Trabajo Final Carrera: Ingeniería de Sistemas

Facultad de Ciencias Exactas-UNICEN

Tema: Heurísticas para la Demostración Automática basado en SNR

Alumno: David Emmanuel López.

Director: Dra. Silvia Schiaffino

Codirector: Dr. Ricardo Oscar Rodríguez

Abstract

El objetivo del presente trabajo es el desarrollo de heurísticas para el mejoramiento de la performance de un método de Demostración Automática de Teoremas. Dicho método consiste en la combinación de otros dos métodos conocidos con los nombres de Secuentes y Resolución No Clausal. El mismo ha experimentado varios cambios y mejoras que serán utilizadas en este trabajo de tesis donde nos proponemos incorporar heurísticas, muchas de ellas inspiradas en el trabajo de Jain, Bartzis y Clarke desarrollado en [JBC06].

Contents

Part I

Introducción y analisis

En la actualidad, una de las aplicaciones más exitosas de la lógica se da en Computación; tanto desde la teoría como al nivel de aplicaciones. De hecho, se ha llegado a comparar su utilidad en Ciencias de la Computación con la de la Matemática en las Ciencias Físicas y el Análisis en Ingeniería [2]. El problema central que enfrenta el uso efectivo y eficaz de la lógica clásica en computación es la alta complejidad computacional que posee su problema de la satisfacibilidad (o validez) de fórmulas. Por ejemplo, en la lógica de primer orden el problema de la satisfacibilidad es indecidible y el problema de chequeo de modelo es PSpace-completo.

Tanto las lógicas clásicas como las no-clásicas son utilizadas principalmente en computación como formalismos para, por un lado, describir objetos y especificar sus propiedades en forma sucinta y con precisión y, por otro, aplicar mecanismos de inferencia como demostradores de teoremas o chequeadores de satisfacibilidad, para hacer explícita la información codificada en la descripción original. Así, la lógica se vuelve cada vez más una ciencia aplicada combinando investigación básica con el desarrollo de herramientas y aplicaciones.

En este contexto, el problema de la satisfacibilidad en general, y en particular el problema de la satisfacibilidad de fórmulas proposicionales clásicas (SAT), es un tema de investigación importante tanto en Informática Teórica como en Inteligencia Artificial (IA) (recordemos que el problema de encontrar un sistema de demostración más potente que Resolución para verificar la satisfacibilidad de una fórmula está en la lista de los diez problemas más importantes todavía no solucionados en Inteligencia Artificial [19]).

La comunidad de IA, ha investigado ampliamente el problema SAT por diversos motivos: (i) es uno de los problemas NP-completos sintáctica y conceptualmente más simples; (ii) captura la esencia de muchos problemas difíciles de IA; y (iii) el descubrimiento de nuevos algoritmos sistemáticos (e.g. [3, 14, 13]) y de búsqueda local (e.g. [18, 21]) ha permitido resolver satisfactoriamente problemas combinatorios difíciles, reduciéndolos al problema SAT, en campos tan diversos como circuitos [14], criptografía [16], planificación [9, 10, 11], y scheduling [1, 5]. Como muestra del interés en SAT, es importante mencionar que en los principales congresos de IA (AAAI, ECAI, IJCAI) hay sesiones dedicadas a este problema. Recientemente también se ha publicado un número especial en los Annals of Mathematics and Artificial Intelligence y en el Journal of Automated

Reasoning. Más recientemente se ha venido aplicado enfoques alternativos a la resolución del problema SAT (ver [4]) que resultan muy interesantes.

La mayoría de las investigaciones y aplicaciones anteriormente mencionadas son basadas en el uso del algoritmo conocido como Davis-Putnam-Logemann-Loveland (DPLL), el cúal requiere que las fórmulas estén en forma normal conjuntiva (CNF). Sin embargo, en dichas aplicaciones, éste no suele ser el formato natural y original en que el conocimeinto se expresa mediante fórmulas. En [23], los autores afirman que el proceso de conversión de una fórmula a CNF incrementa la complejidad computacional y puede destruir la estructura inicial de la misma, la cual podría ser crucial para el chequeo eficiente de SAT. Más específicamente, los aspectos más ampliamente criticados del uso de formas normales son:

- Debido a dicha traducción, la búsqueda es llevada a cabo en un espacio más bien remoto respecto del problema original del usuario, lo cual dificulta que el usuario interactué con el proceso de búsqueda. Esta comunicación es necesaria para preguntar por información de una forma incremental; para mostrarle las explicaciones pertinentes relacionadas con las conclusiones obtenidas, o para pedir ayuda al usuario durante la demostración.

- La conversion de una fórmula bien formada (well-formed fórmula - wff) a CNF puede eliminar información pragmáticamente útil codificada en la elección de conectores lógicos, por ejemplo: "$\neg p \vee q$ " puede sugerir análisis de casos, pero la expresión lógica equivalente "$p \rightarrow q$" sugiere encadenamiento.

- La conversión puede requerir un número grande de cláusulas para representar una wff y una redundancia sustancial en el espacio de búsqueda, por ejemplo: "$p \leftrightarrow q$" en forma clausal es "$(\neg p \vee q) \wedge (p \vee \neg q)$", con dos instancias de los átomos "p" y "q". En el peor caso, cuando "p" y "q" son formadas por equivalencias, la conversión puede resultar en un número de cláusulas que es exponencial en relación a el tamaño de la fórmula. Otro ejemplo puede ser "$(p_1 \wedge p_2 \ldots \wedge p_m) \vee (q_1 \wedge q_2 \ldots \wedge q_n)$ ", esta fórmula se convierte en $M \times N$ cláusulas de la forma "(p_i, q_j)" , entonces cada "p_i" toma lugar (aparece) N veces y cada "q_j" toma lugar M veces.

Como el problema principal de resolución deriva de la conversión a forma clausal, han surgido propuestas que no necesitan de esta conversión. El método de resolución no clausal es una de ellas. Mientras que la resolución clausal obtiene resolventes de cláusulas que contienen literales complementarios explícitos, este método trabaja con wff buscando átomos complementarios usando el concepto de polaridad opuesta (la polaridad es determinada por la paridad del número explícito o implícito de negaciones en el rango en que el átomo aparece).

En la forma clausal, los literales involucrados son eliminados y el nuevo resolvente es obtenido de la disyunción del resto. En resolución no clausal se toma

cada fórmula original y se reemplaza todas las ocurrencias *pares* o *impares* del átomo a resolver por $true$ o $false$. El resolvente no-clausal es la disyunción de las fórmulas resultantes, la cual es simplificada por medio de reducciones funcionales que eliminan las ocurrencias de $true$ y $false$ (ej: $false \vee Q$ es reducido a Q; $true \vee Q$ es reducido a $true$).

Sin embargo, la falta de homogeneidad de la forma no-clausal, tiene algunas características que no son deseables [22]:

- la operación de unificación es más compleja.
- en una wff, un átomo puede aparecer mas de una vez con diferente polaridad, entonces es posible derivar más de un resolvente para el mismo par de wff, aún resolviendo con el mismo átomo, y también resolviendo una wff consigo misma.
- en la forma clausal, si un literal es *puro* (este no puede ser resuelto con ninguna otra cláusula), la cláusula en donde este aparece puede ser eliminada. Este no es el caso de resolución no-clausal, a causa de que no todos los átomos de una wff son esenciales (ej: $(p \wedge q), (\neg p)$ es inconsistente, pero la primer fórmula contiene el átomo puro q.
- la operación de subsumpción debe ser redefinida para notificar que p es subsumido por $p \wedge q$ y que $p \vee q$ es subsumido por p.
- En [15, 17] se sugiere una extensión que consiste en permitir la resolución con fórmulas no atómicas (ej: $p \vee q$ con $(p \vee q) \rightarrow r$). Esto permitía refutaciones más legibles y cortas, pero hay algunas cuestiones de complejidad computacional que no han podido ser superadas hasta ahora, razón por la cual no suele implementarse.

Un método alternativo a resolución (clausal o no) es el llamado Tableaux, introducido por Smullian en [20], el cual puede ser visto como un caso particular de los Sistemas de Secuentes propuesto originariamente por Gentzen [6]. Estas familias de métodos se caracterizan por tener múltiples reglas de inferencias (en contraposición a resolución, clausal o no, que sólo tiene una). Esta multiplicidad trae aparejado un problema de decisión acerca de que regla es más apropiada usar en cada paso. Pero, por otro lado, tienen la ventaja de no requerir que las fórmulas estén en ninguna forma normal y suelen ser muy naturales y flexibles en el caso de tener que agregar nuevos operadores (p.ej. no-estándares). Así, los sistemas basados en múltiples reglas tienen la característica de ser capaces de ser usados directamente. Dado que no hay formas normales, y sólo sub-fórmulas o instancias de sub-fórmulas son usadas en la búsqueda de una demostración, resulta muy fácil involucrar al usuario en la exploración. Además, resulta fácil incorporar heurísticas del dominio del problema de forma directa. Sin embargo, estos métodos tienen varias características ineficientes que el método de resolución resuelve en el preprocesamiento de transformación a las formas normales.

Por todo lo antedicho, parece razonable encontrar algún mecanismo para integrar estos dos paradigmas tan diferentes. En [7], se plantea un estudio de caso basado en el modelo de Gentzen que resultará interesante a los efectos de nuestro trabajo.

1 Objetivos

En [12] se propone un método de prueba que combina el método de Tableaux con resolución no-clausal. Ese método se abandonó por razones no académicas. Sin embargo, las bases y los resultados allí planteados siguen estando vigentes. De hecho, en la literatura existen muy pocas propuestas que no estén basadas en DPLL. El objetivo de este trabajo es retomar aquel sistema SNR propuesto por Kvitca, implementarlo para lógica proposicional clásica utilizando una estrategia de control del tipo "set support" e incorpar heurísticas que lo hagan eficiente. El uso de una estrategía del tipo "set support" consistiría en asumir que el conjunto de premisas originales es consistente y que se incorpora en el soporte la fórmula a demostrar. Sobre dicha fórmula se aplica las reglas de descomposición en subfórmulas de Tableaux hasta que dicho proceso da lugar a la obtención de un átomo (en realidad uno por cada ramificación del árbol de prueba). Para continuar con el proceso de búsqueda de una prueba se incorpora al soporte el resolvente no-clausal entre una fórmula del conjunto de premisas y el átomo obtenido. Y así se continúa hasta que todas las ramas del árbol estén cerradas o en caso contrario hasta que en algún punto no haya más fórmulas para descomponer y la rama quede abierta. Por otro lado, las heurísticas a desarrollar están inspiradas en las propuestas en [8] y pueden ser resumidas de la siguiente manera:

- Elegir la fórmula para continuar la exploración estimando su grado de ramificación. Eso permitirá explorar árboles de prueba más compactos. La idea es desarrollar un algoritmo que determine ese grado de ramificación en tiempo lineal.
- Mantener en una memoria "cache" los átomos encontrados durante el desarrollo de una rama de prueba y usarlos para simplicar las fórmulas candidatas a explorar en la premisas. El objetivo es obtener en forma temprana ramas cerradas o determinar que no pueden serlo.

2 Tópicos preliminares

En esta sección se introducen los tópicos necesarios para los comprensión del trabajo realizado, los mismos se pueden omitir si el lector se encuentra familiarizado con los mismos.

2.1 Lógica proposicional

La lógica se ocupa de estudiar los métodos de razonamiento, de especificarlos y sistematizarlos, la lógica proposicional es es un lenguaje formal para especificar proposiciones (afirmaciones) mediante un conjunto de reglas sintácticas y semánticas. Una proposición es una descripción o hecho que puede ser verdadero o falso, puede ser descripta mediante el lenguaje natural y simbolizarse mediante una letra del alfabeto, o bien puede ser descripta por proposiciones relacionadas mediante conectores lógicos, construida en base a las reglas sintácticas del lenguaje lógico.

Podemos definir la siguiente proposición y simbolizarla con una letra del alfabeto:

$p =$ "*hoy llueve*"

$q =$ "*uso paraguas*"

y construir una proposición compuesta:

$p \rightarrow q$

que se interpreta como, "si hoy llueve, uso paraguas".

Escapa a la lógica evaluar si hoy llueve o no llueve, pero si es incumbencia de la lógica tomar unas premisas o hipótesis (que no son más que proposiciones que se consideran verdaderas) y deducir el valor de verdad de otras proposiciones; es decir interesa el estudio de los sistemas lógicos, de cómo construir razonamientos correctos, de lenguajes y algoritmos que lo realicen de forma correcta y completa.

Sintaxis de la lógica proposicional

En primer lugar, definiremos el alfabeto, es decir el conjunto de símbolos sobre el cual se construye una fórmula o proposición lógica.

- un conjunto de símbolos proposiciones $S = \{p, q, r, ...\}$
- un conjunto de conectores lógicos $C = \{\wedge, \vee, \neg, \rightarrow\}$
- un conjunto de símbolos auxiliares $A = \{(,)\}$.

Luego se define inductivamente el conjunto P que incluye a todas las proposiciones bien construidas.

- $S \subset P$, es decir todos los símbolos proposiciones son una proposición válida.
- dado $A, B \in P$ entonces $A \wedge B \in P$, $A \vee B \in P$, $A \rightarrow B \in P$ y $(A \wedge B) \in P$, $(A \vee B) \in P$, $(A \rightarrow B) \in P$.
- dado $A \in P$ entonces $\neg A \in P$ y $(\neg A) \in P$.

Semántica de la lógica proposicional

La semántica es la interpretación que se le da a una fórmula lógica. Como se dijo, las fórmulas en lógica proposicional pueden tomar un valor de verdad $true$ o $false$. A continuación se definirá cuáles son las reglas para dar valor de verdad a las fórmulas, tambien llamada valuaciónes.

Las variables proposicionales $p \in S$ mediante una valuación v toman un valor de verdad $true$ o $false$. La valuación de las variables proposicionales se especifican manualmente y se toma como hipótesis; se deducen a partir de las fórmula(s) lógica(s), o bien son variables libres, sin ninguna asignación de valor de verdad.

El valor de verdad o valuación de una fórmula lógica compuesta, se especifica mediante tablas de verdad. Una tabla de verdad consta de dos proposiciones valuadas, un operador y la valuación resultante de la fórmula.

Tabla de verdad del operador $\wedge$:

A	B	$A \wedge B$
t	t	t
t	f	f
f	t	f
f	f	f

donde $A, B \in P$.

Coloquialmente, el operador $\wedge$ representa "y" en lenguaje natural, en donde la fórmula resultante se cumple si ambas fórmulas que componen la fórmula se cumplen.

Tabla de verdad del operador $\vee$:

A	B	$A \vee B$
t	t	t
t	f	t
f	t	t
f	f	f

donde $A, B \in P$.

Coloquialmente, el operador $\vee$ representa "o" en lenguaje natural, en donde la fórmula resultante se cumple si una de las fórmulas que la componen se cumple.

Tabla de verdad del operador $\rightarrow$:

A	B	$A \rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

donde $A, B \in P$.

Coloquialmente, el operador $\rightarrow$ representa una implicación, en donde la fórmula resultante se cumple si al cumplirse a se cumple b.

Tabla de verdad del operador $\neg$:

A	$\neg A$
t	f
f	t

donde $A \in P$.

Coloquialmente, el operador $\neg$ representa una negación, en donde la fórmula resultante se cumple sólo si no se cumple A.

Fórmulas tautológicas y contradictorias

- Una tautología ocurre cuando una fórmula es siempre verdadera, ya que no existe una valuación que la haga falsa. Por ejemplo $p \vee \neg p$ es una tautología.
- Una contradicción ocurre cuando una fórmula es siempre falsa, ya que no existe una valuación que la haga verdadera. Por ejemplo $p \wedge \neg p$ es una contradicción.
- La negación ($\neg$) de una fórmula tautológica es una fórmula contradictoria, y viceversa.

Satisfacibilidad

El problema de la satisfacibilidad de una fórmula lógica, tambien llamado SAT, es ampliamente conocido y de extensa investigación por su facil formulación y porqué existen muchos problemas que puede reducirse a este. Consiste en poder determinar si dada una fórmula lógica, la misma tiene al menos una valuación que la satisfaga, o si no la tiene. Está clasificado dentro de la categoria de algoritmos **NP-completo**.

Cuando una fórmula tiene una valuación que la hace verdadera, se dice que es satisfacible, cuando no existe valuación posible se dice insatisfacible.

Deducción de nuevas fórmulas

Es de interés poder deducir nuevas fórmulas a partir de otras. A aquellas fórmulas que se suponen verdaderas se las denomina hipótesis, y las nuevas fórmulas

deducidas a partir de las hipótesis se las denomina tesis. En el presente trabajo, se utilizará sólo una tesis objetivo.

Supongamos que tenemos un conjunto inicial de hipótesis $H_1, H_2, ..., H_n$ y una fórmula T tesis, para denotar que se deduce del conjunto de hipótesis la tesis, se escribe:

$H_1, H_2, ..., H_n \Rightarrow T$

Usamos el operador $\Rightarrow$ conocido como "consecuencia semántica". Una tesis se deduce de un conjunto de hipótesis si, para toda valuación v que satisface a $H_1 \wedge H_2 \wedge ... \wedge H_n$ también satisface a T, es decir, la fórmula $(H_1 \wedge H_2 \wedge ... \wedge H_n) \rightarrow T$ es tautología, formalmente descripto como $\models (H_1 \wedge H_2 \wedge ... \wedge H_n) \rightarrow T$.

2.2 Forma Normal Conjuntiva

Una forma normal describe una forma en la que una fórmula se puede expresar, cumpliendo ciertos requisitos. Esto es útil ya que al estar una fórmula en una forma normal dada se cumplen ciertas propiedades que pueden ser útiles. Por lo general, se toma una fórmula escrita en wff y se aplican transformaciones que mantienen la equivalencia lógica de la misma para obtener una fórmula en forma normal.

Es importante mencionar qué fórmula tiene una fórmula equivalente en donde solo se usan los operadores logicos $\{\wedge, \vee, \neg\}$.

A continuación se describe la Forma Normal Conjuntiva:

Definiciones

- Un literal es una variable proposicional p o su negación $\neg p$.
- Una cláusula es una fórmula $C = l_1 \vee l_2 \vee ... l_n$ donde $l_i$es un literal.
- Una fórmula esta forma normal conjuntiva si $F = C_1 \wedge C_2 \wedge ... \wedge C_m$donde C_i es una cláusula.

Lógicamente hablando, debe satisfacerse al menos un literal de cada cláusula para que una valuación haga a la fórmula verdadera.

2.3 Resolución clausal

El problema de deducir fórmulas a partir de otras, dio origen a métodos especificos para realizar este proceso. Así surge resolución.

Como $H_1, H_2, ..., H_n \Rightarrow T$ es equivalente a fórmular que $\{H_1, H_2, ..., H_n, \neg T\}$ es insatisfacible, se puede fórmular el problema de decidir si una deducción es válida en términos del problema SAT.

Forma Clausal

Es posible redefinir la forma normal conjuntiva en términos de conjuntos; una cláusula que es de la forma $C = l_1 \vee l_2 \vee ... \vee l_n$ donde l_i es un literal, puede escribirse como $C = \{l_1, l_2, ..., l_n\}$ manteniendo la misma semántica. De igual manera una fórmula que es una conjunción de cláusulas $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ puede escribirse como $F = \{C_1, C_2, ..., C_m\}$.

Resolvente

Tomemos la fórmula $F = (p \vee \neg q) \wedge (q \vee r)$ en forma clausal:

$C = \{\{p, \neg q\}, \{q, r\}\}$

que también puede ser escrita para mayor claridad como $C = \{p \vee \neg q, q \vee r\}$.

Si una valuación v hace $v(q) = true$ entonces debe hacer $v(p) = true$ para que la fórmula sea verdadera. Si $v(q) = false$ entonces $v(r) = true$. Si la fórmula es válida, se deduce que la fórmula $F_r = p \vee r$ también lo es, cualquier valuación que haga verdadera la fórmula F también hace verdadera F_r. A su vez, una valuación que no satisface a F_r tampoco satisface a F. Lo que ocurre aqui es que se ha eliminado la variable q.

Como lo que es de interés en un método por refutación es buscar la contradicción, si bien no toda valuación que satisface F_r satisface F, toda valuación que no satisface a F_r tampoco satisface a F, por lo tanto si F_r es contradictoria, F tambien lo es.

Se define la operación Resolvente entre dos cláusulas con literales complementarios:

$Res(C_1,C_2) = (C_1 - \{l\}) \cup (C_2 - \{l_c\})$.

donde l_c denota al literal complementario de l, que es el literal mismo negado ($\neg$).

Por ejemplo, para la fórmula en forma clausal $C = \{p \vee \neg q, q \vee r\}$,

$Res(p \vee \neg q, q \vee r) = (\{p, \neg q\} - \{\neg q\}) \cup (\{q \vee r\} - \{q\}) = \{p, r\}$.

Mediante la operación resolvente que toma dos cláusulas y un par de literales complementarios, es posible deducir nuevas fórmulas mas reducidas a partir de otras, y de esta forma obtener una refutación (o la imposibilidad de encontrarlo).

Encontrar una refutación equivale a alcanzar dos cláusulas unitarias, con literales complementarios:

$Res(\{l\}, \{l_c\}) = \perp$

Algoritmo DPLL (Davis-Putnam-Logemann-Loveland)

Si bien la fórmulación del problema es facil, realizar el proceso de refutación es complejo si disponemos de una entrada grande. El algoritmo DPPL sistematiza la búsqueda de la refutación y garantiza completitud y correctitud.

2.4 Resolución no-clausal

La resolución no-clausal consiste en aplicar el enfoque de resolución en fórmulas que no están representadas en forma clausal. La ventaja de operar con las fórmulas en su estado original, o a lo sumo nnf, se mencionó en la introducción. La base de la resolución no-clausal es una regla de inferencia general:

$$\frac{F[G] \quad F'[G]}{F[G/\bot] \vee F'[G/\top]}$$

donde F y F' son dos fórmulas y G una subfórmula (puede ser un átomo). $G/\bot$ simboliza que G es reemplazada por $\bot$ (símbolo contradicción), y $G/\top$ simboliza que G es reemplazada por $\top$ (simbolo tautología).

En una regla de inferencia las hipótesis se escriben en la parte superior y la tesis o conclusíon en la parte inferior de la definición. Por lo tanto, su interpretación es si $F[G], F'[G]$ son verdaderas, se deduce que $F[G/\bot] \vee F'[G/\top]$ es verdadera. Si F y F' son dos cláusulas en cnf entonces la regla de inferencia es equivalente a resolución clausal en cnf. Un proceso de demostración mediante resolución no-clausal comparte similitudes con resolución clausal, que es más fácil de comprender. Ambos utilizan la regla de inferencia para generar nuevas fórmulas válidas en el sistema, buscando obtener una refutación. Sin embargo en la resolución no-clausal hay más decisiones que tomar: entre dos fórmulas se puede hacer resolución sobre cualquiera de los átomos en común, y por cada par de átomos se debe elegir en que fórmula ocurrirá $\bot$ y respectivamente, en cual ocurrirá $\top$.

A continuación se cita un ejemplo de [24]:

Resolución general

1. $a \wedge c \leftrightarrow b \wedge d$ (hipótesis)
2. $a \wedge c$ (hipótesis)
3. $\neg[b \wedge d]$ (tesis) - negada para buscar refutación
4. $[a \wedge \bot] \vee [a \wedge \top]$(resolvente de (2), (2) en c) $\Rightarrow a$
5. $[a \wedge \bot] \vee [a \wedge \top \leftrightarrow b \wedge d]$((2), (1) en c) $\Rightarrow a \leftrightarrow b \wedge d$
6. $\bot \vee [\top \leftrightarrow b \wedge d]$ ((4), (5) en a) $\Rightarrow b \wedge d$

7. $\bot \wedge d \vee \top \wedge d$ ((6), (6) en b) $\Rightarrow d$
8. $b \wedge \bot \vee b \wedge \top$ ((6), (6) en d) $\Rightarrow b$
9. $\bot \vee \neg[\top \wedge d]$ ((8), (3) en b) $\Rightarrow \neg d$
10. $\bot \vee \neg\top$((7), (9) en d) $\Rightarrow \bot$ (refutación)

El método de resolución general permite trabajar con las fórmulas originales y construir una refutación matemáticamente legible, pero no resulta fácil construir un algoritmo que cumpla ciertas garantías como eficiencia y completitud. En lo que respecta al presente trabajo de tesis, se utilizará resolución no-clausal entre una fórmula y un átomo, por lo tanto solo hay una fórmula G posible a utilizar, y una sola alternativa para asignar $\bot$. Por ejemplo, sea $F = a \wedge \neg c$ y $F' = G = c$ las hipótesis, $\Rightarrow F[\top] \vee F'[\bot]$, reemplazando las ocurrencias de G por los operadores respectivos obtenemos $a \wedge \neg\top \vee \bot \cong \bot$.

Al aplicar resolución no-clausal entre fórmula y átomo, inferimos una nueva fórmula que es la fórmula original con el átomo reemplazado por su valor, y reducida si aplica alguna de las siguientes reglas de equivalencias lógicas:

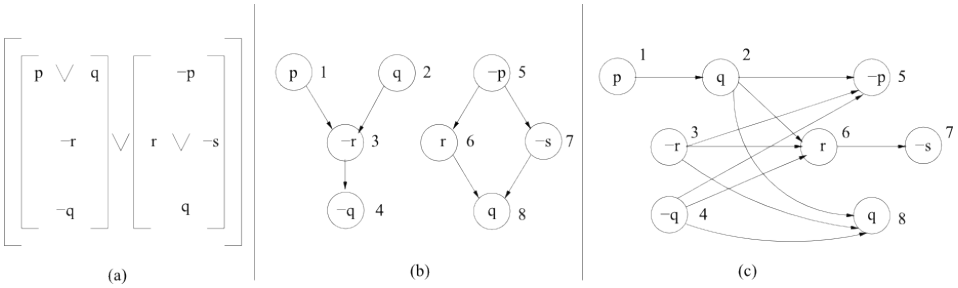
1. $F \wedge \top \cong F$
2. $F \wedge \bot \cong \bot$
3. $F \vee \top \cong \top$
4. $F \vee \bot \cong F$
5. $\neg\bot \cong \top$
6. $\neg\top \cong \bot$

Lo mismo ocurre al aplicar resolución clausal entre una fórmula (cláusula) y un átomo, la cláusula inferida será la original con el átomo reemplazado por su valor y reducida, sólo que no se consideran los casos **1**. y **2.** si las cláusulas estan en cnf. Este caso particular da lugar a la resolución unitaria.

2.5 General Matings

El enfoque General Matings aplicado al problema SAT desarrollado en [8] no requiere la conversión a forma clausal de las fórmulas de entrada, y es uno de los tres principales métodos en los que se basa el desarrollo del presente trabajo. Utiliza una representación matricial de la fórmula a evaluar. La fórmula es translada en un formato de dos dimensiones llamado **vertical-horizontal path form** (vhpform)**.** En esta forma las disjunciones (operandos de $\vee$) se disponen horizontalmente y las conjunciónes (operandos de $\wedge$) se disponen verticalmente.

Por ejemplo, para la fórmula $F = ((p \vee q) \wedge \neg r \wedge \neg q) \vee (\neg p \wedge (r \vee \neg s) \wedge q)$ se tiene la siguiente representación matricial:



(a) representacion matricial, (b) vpgraph, (c) hpgraph

Un vertical path a través de la vhpform es una secuencia de literales resultante al elegir un camino entre el ámbito izquierdo o derecho de cada ocurrencia de $\vee$. Los **vertical paths** de F son $\{\langle p, \neg r, \neg q\rangle, \langle q, \neg r, \neg q\rangle, \langle \neg p, r, q\rangle, \langle \neg p, \neg s, q\rangle\}$. De forma análoga, un horizontal path a través de la vhpform es una secuencia de literales resultante al elegir un camino entre el ámbito izquierdo o derecho de cada ocurrencia de $\wedge$. Los **horizontal paths** son $\{\langle p, q, \neg p\rangle, \langle p, q, r, \neg s\rangle, \langle \neg r, r, \neg s\rangle, \langle \neg r, q\rangle, \langle \neg q, q\rangle\}$. La fórmula es satisfacible si y solo si, existe un **vertical path** a través de este orden que no contenga dos literales opuestos (l y $\neg l$). La fórmula de entrada no requiere ninguna forma normal, sin embargo internamente se usa **negation normal form** (nnf).

Desde una perspectiva de alto nivel, el algoritmo enumera todos los posibles vertical paths en la vhpform de una fórmula dada, hasta que un vertical path sin literales opuestos es encontrado. Si cada vertical path contiene literales opuestos entonces la fórmula es insatisfacible. El número de vertical paths puede ser exponencial en función del tamaño de la fórmula de entrada. Por lo tanto, el desafio para obtener un procedimiento eficiente es prevenir la enumeración de vertical paths tanto como sea posible.

Los vertical paths codifican la forma normal disjuntiva (dnf) y los horizontal paths la forma normal conjuntiva (cnf). El algoritmo utiliza el espacio de búsqueda determinado por el conjunto de posibles vertical paths para buscar una valuación que satisfaga la fórmula de entrada, y utiliza el espacio de búsqueda determinado por los horizontal paths como base para detectar conflictos y realizar deducciones unitarias.

En base a la representación matricial se construyen dos grafos sobre los cuales el método trabaja realmente, como se observa en (b) y (c):

Grafo de vpaths: Es un grafo aclíclico dirigido (GDA) compuesto por todos los vertical paths de una fórmula, se simboliza con $G_v(\varphi)$. Es equivalente a la forma normal disjuntiva (DNF). Cada path del grafo es equivalente a una cláusula de la forma normal.

Grafo de hpaths: Es un grafo aclíclico dirigido (GDA) compuesto por todos los horizontal paths de una fórmula, se simboliza con $G_h(\varphi)$. Es equivalente a la forma normal conjuntiva (CNF). Cada path del grafo es equivalente a una cláusula de la forma normal.

Vertical path (vpath): Un vertical path en vhpform es una secuencia de literales que resulta de elegir ya sea el alcance izquierdo o derecho de cada ocurrencia del operador $\vee$.

Horizontal path (hpath): Un horizontal path en vhpform es una secuencia de literales que resulta de elegir ya sea el alcance izquierdo o derecho de cada ocurrencia del operador $\wedge$.

Cada rl-path π del grafo de vertical paths representa una valuación σ candidata para satisfacer a la fórmula φ. A su vez equivale a una cláusula en DNF.

A continuación se citan dos corolarios de [8], sobre los cuales se opera en los grafos:

Corolario 1. Una asignacion σ satisface φ si existe un rl-path π en $Gv(\varphi)$ tal que σ satisface π.

Corolario 2. Una asignacion σ falsifica φ si existe un rl-path π en $Gh(\varphi)$ tal que σ falsifica π.

donde σ simboliza una valuación, φ una fórmula en nnf, $Gv(\varphi)$ grafo de vertical paths de la fórmula φ, $Gh(\varphi)$ grafo de horizontal paths de la fórmula φ y π un path.

Mediante esta representación el algoritmo opera utilizando poda en el espacio de búsqueda, apredizaje dirigido por conflictos, non-cronológical backtracing y heurísticas. Para mayor detalles del método se recomienda leer la publicación original. En una sección posterior de la presente investigación, se citan y realizan las equivalencias que dan origen a la heurística propuesta; mejoras por fuera del algoritmo heurístico, que facilitan la información usada por la heurística y hacen mas eficiente su ejecución, y tambien aquellos aspectos que SNR ya considera en relación a las falencias de General Matings.

2.6 Secuentes

Un secuente es un par (Γ, Δ) de secuencias finitas $\Gamma =< A_1, ..., A_m >, \Delta =< B_1, ..., B_n >$ de proposiciones o fórmulas, pudiendo estar vacias. En el presente trabajo, en lugar de especificar a un secuente mediante la notación (Γ, Δ), usaremos la notación $\Gamma \Rightarrow \Delta$.

La secuencia Γ se denomina antecedente del secuente y la secuencia Δ consecuente. Una valuación v satisface al secuente si $v \models A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow B_1 \vee B_2 \vee ... \vee B_n$. Asimismo, una valuación hace falso al secuente si hace

verdaderas a $A_1, A_2, ..., A_m$ y falsas a $B_1, B_2, ..., B_n$. El de método secuentes trabaja por refutación, busca una valuación que haga verdaderas las fórmulas de Γ y falsas las fórmulas de Δ, si tiene éxito entonces el sistema es inconsistente, pero si no encuentra ningún contraejemplo, entonces el sistema es consistente.

Como se mencionó, Γ o Δ pueden estar vacios, Γ representa las fórmulas que son verdaderas, o requieren serlo, y Δ las fórmulas que se necesita falsificar. Γ tambien simboliza el conjunto de hipótesis y Δ el conjunto de tesis, de tal manera que si busca demostrar $\Gamma \Rightarrow \Delta$, es decir si a partir de las hipótesis se deducen las tesis. Como caso particular se puede utilizar el método para decidir si una fórmula F es tautológica (o contradictoria), para tal caso existe una sola tesis a refutar y ninguna hipótesis, $\Gamma = \emptyset$ y $\Delta = F$.

El método de secuentes, al igual que resolución no-clausal, trabaja aplicando reglas de inferencias, construyendo un árbol de deducción.

Una regla de inferencia esta compuesta de una o varias premisas y una conclusión, y se denota como sigue:

$$\frac{P_1, P_2}{C}$$

donde P_i simboliza una premisa y C la conclusión.

En el presente trabajo las fórmulas pertenecen a la lógica proposicional, es decir son variables proposicionales $\{p, q, r...\}$ relacionadas mediante los operadores $\{\wedge, \vee, \rightarrow, \neg\}$, por lo tanto las reglas de inferencia se construyen sobre esos mismos operadores.

Existen varios sistemas de inferencia, En [12] se analizan los sistemas NK, LK y G y se toma como base el sistema G (Gentzen). A continuación se especifican sus reglas:

$li : \wedge \frac{T1,A,B,T2 \Rightarrow U}{T1,A \wedge B,T2 \Rightarrow U}$

$ri : \wedge \frac{T \Rightarrow U1,A,U2 \quad T \Rightarrow U1,B,U2}{T1 \Rightarrow U1,A \wedge B,U2}$

$ri : \vee \frac{T \Rightarrow U1,A,B,U2}{T \Rightarrow U1,A \vee B,U2}$

$li : \vee \frac{T1,A,T2 \Rightarrow U \quad T1,B,T2 \Rightarrow U}{T1,A \vee B,T2 \Rightarrow U}$

$li : \vee \frac{T1,T2 \Rightarrow A,U}{T1,\neg A,T2 \Rightarrow U}$

$ri : \neg \frac{A,T \Rightarrow U1,U2}{T1 \Rightarrow U1,\neg A,U2}$

$li : \rightarrow \frac{T1,T2 \Rightarrow A,U \quad T1,B,T2 \Rightarrow U}{T1,A \rightarrow B,T2 \Rightarrow U}$

$ri : \rightarrow \frac{T1,A \Rightarrow U1,B,U2}{T1 \Rightarrow U1,A \rightarrow B,U2}$

$ri : \neg \frac{A,T \Rightarrow U1,B,U2}{T1 \Rightarrow U1,A \rightarrow B,U2}$

$axiom : T1, A, T2 \Rightarrow U1, A, U2$

Reglas de inferencia del sistema G: li (left inference) denota a las reglas que aplican en el antecedente del secuente de la conclusión de la regla de inferencia (secuente que se encuentra abajo de la línea horizontal); mientras que ri (right inference) denota a las reglas que aplican al consecuente del secuente de la conclusión.

Las reglas pueden clasificarse en aquellas que tienen sólo una premisa y las que tiene dos. El proceso de refutación en un sistema G consiste en tomar el secuente inicial e intentar falsificarlo aplicando reglas de inferencia, generando un árbol de deducción.

A continuación un ejemplo simple obtenido de [12], supongamos que se quiere demostrar: $\{r, s, (s \wedge p) \vee (r \wedge s) \rightarrow k\} \Rightarrow \{k\}$, u omitiendo la notación de conjuntos: $r, s, (s \wedge p) \vee (r \wedge s) \rightarrow k \Rightarrow k$.

- $r, s, (s \wedge p) \vee (r \wedge s) \rightarrow k \Rightarrow k$ (aplico $li :\rightarrow$)
- r1: $r, s \Rightarrow k, (s \wedge p) \vee (r \wedge s)$ (aplico $ri : \vee$)
  - $r, s \Rightarrow k, (s \wedge p), (r \wedge s)$ (aplico $ri : \wedge$)
  - r1: $r, s \Rightarrow k, s, (r \wedge s)$ (aplico $axiom$)
    - $\perp$(rama cerrada)
  - r2: $r, s \Rightarrow k, p, (r \wedge s)$ (aplico $ri : \wedge$)
    - r1: $r, s \Rightarrow k, p, r$ (aplico $axiom$)
      - $\perp$(rama cerrada)
    - r2: $r, s \Rightarrow k, p, s$ (aplico $axiom$)
      - $\perp$(rama cerrada)
- r2: $r, s, k \Rightarrow k$ (aplico $axiom$)
- $\perp$(rama cerrada)

Como todas las ramificaciones del secuente original culminan en una regla axiom, que simboliza la imposibilidad de satisfacer un secuente que tiene la misma fórmula atómica en el antecedente y en el consecuente, queda demostrado que la deducción $r, s, (s \wedge p) \vee (r \wedge s) \rightarrow k \Rightarrow k$ es válida.

La ocurrencia de un axioma, simboliza una afirmación que siempre es verdadera en el sistema.

Si existiese una rama abierta, en donde no es posible aplicar axiom ni ninguna otra regla de inferencia, es posible satisfacer las hipótesis y fasificar la tesis, por lo tanto se encontró un contraejemplo, refutando la deducción que se intentaba demostrar. A este suceso se le denomina "rama abierta".

Top-down o Bottom-up

Las reglas de inferencia pueden utilizarse para, a partir de las hipótesis deducir una conclusíon; o bien, para a partir de una conclusión obtener las hipótesis que la pueden hacer verdadera. La estrategia Top-down es la utilizada en el método SNR que será descripto en la sección siguiente. Consiste en buscar las hipótesis en base a una conclusión, es de gran utilidad cuando no se conocen las hipótesis y se quiere buscar una demostración.

El método de secuentes tiene la ventaja de no trabajar directamente con un espacio de búsqueda de valuaciones, así como también una simplicidad estructural en sus reglas de inferencia, trabajando con las fórmulas originales y permitiendo añadir heurísticas facilmente, pero el sistema tiene los siguientes problemas:

- la operación de eliminación de fórmulas en los secuentes consume mucho tiempo y destruye la base para uso futuro.
- Sin una adecuada heurística el sistema se expande a muchas fórmulas, cuando solo fórmulas emparentadas con el objetivo actual deben ser descompuestas.
- En muchos casos el árbol de demostración es redundante.

2.7 Spliting Non-clausal Resolvent

Debido a los problemas mencionados del sistema de inferencia G, se ha propuesto una modificación que sólo permite demostraciones lineales y que es completa si el conjunto de hipótesis es consistente. La modificación consiste en aplicar las reglas de inferencia sólo al consecuente. Cuando obtenemos una fórmula atómica, esta es agregada al conjunto de fórmulas en el antecedente y otra fórmula es propuesta como objetivo. La nueva fórmula objetivo debe incluir al átomo alcanzado con polaridad opuesta para generar una demostración lineal, luego se aplica resolución no-clausal entre el átomo y la fórmula seleccionada, obteniendo una nueva fórmula reducida, que será la nueva fórmula objetivo.

Reglas de SNR:

$gt : \wedge \frac{P,B{=}t \Rightarrow A{=}t \mid P,A{=}t \Rightarrow B{=}t}{P \Rightarrow (A \wedge B){=}t}$

$gf : \wedge \frac{P \Rightarrow A{=}f \quad P \Rightarrow B{=}f}{P \Rightarrow (A \wedge B){=}f}$

$gf : \vee \frac{P,B{=}f \Rightarrow A{=}f \mid P,A{=}f \Rightarrow B{=}f}{P \Rightarrow (A \vee B){=}f}$

$gt : \vee \frac{P \Rightarrow A{=}t \quad P \Rightarrow B{=}t}{P \Rightarrow (A \vee B){=}t}$

$gf : \rightarrow \frac{P,A{=}t \Rightarrow B{=}t \mid P,B{=}f \Rightarrow A{=}t}{P \Rightarrow (A \rightarrow B){=}f}$

$gt :\rightarrow \frac{P \Rightarrow A{=}f \quad P \Rightarrow B{=}t}{P \Rightarrow (A \rightarrow B){=}t}$

$axiom\, P, X = t \Rightarrow X = f \mid P, X = f \Rightarrow X = t$ cuando X es una fórmula atómica.

$gt : \neg \frac{P \Rightarrow A{=}f}{P \Rightarrow \neg A{=}t}$

$gf : \neg \frac{P \Rightarrow A{=}t}{P \Rightarrow \neg A{=}f}$

$exchange : \neg \frac{P,Z{=}M,X{=}N \Rightarrow Z{=}M}{P,Z{=}M \Rightarrow X{=}N}$

A causa de que las fórmulas no son eliminadas en esta versión del sistema, es necesario incorporar una prueba de detección de bucle para garantizar completitud. Esta prueba no consume tiempo a causa de que puede ser desarollada por la regla 'axiom'. La elección de la fórmula 'Z' [fórmula a intercambiar] debería no ser arbitraria. Por el contrario, el átomo 'X' debe aparecer con polaridad opuesta en la fórmula de modo que esta pueda ser seleccionada.

A continuación se utilizará la misma fórmula usada Secuentes para desarrollar un ejemplo del método.

Ejemplo

- $r = t$, $s = t$, $((s \wedge p) \vee (r \wedge s) \rightarrow k) = t \Rightarrow k = f$ (aplico $exchange$)
- $r = t$, $s = t$, $k = f \Rightarrow ((s \wedge p) \vee (r \wedge s)) = f$ (aplico $gf : \vee$)
- $r = t$, $s = t$, $k = f$, $(s \wedge p) = f \Rightarrow (r \wedge s) = f$ (aplico $gf : \wedge$)
- r1: $r = t$, $s = t$, $k = t$, $(s \wedge p) = f, \Rightarrow r = f$ (aplico $axiom$)
- $\perp$
- r2: $r = t$, $s = t$, $k = t$, $(s \wedge p) = f, \Rightarrow s = f$ (aplico $axiom$)
- $\perp$

Se observa una mejora en el proceso de demostración (se aplicaron menos reglas de inferencias). A continuación un ejemplo extraído de [12], en donde se muestra cuando el demostrador SNR explora ramificaciónes abiertas.

Exploración de ramas abiertas

- $a \wedge b$, $a \rightarrow c = t$, $c \rightarrow d = t$, $d \rightarrow e = t \Rightarrow b$
- $a \wedge b = t$, $a \rightarrow c = t$, $c \rightarrow d = t$, $d \rightarrow e = t \Rightarrow b = t$ (aplico $exchange$ con $a \wedge b = t$)
- $b = f$, $a \wedge b = t$, $a \rightarrow c = t$, $c \rightarrow d = t$, $d \rightarrow e = t \Rightarrow a \wedge b = t$ (aplico $gt : \wedge$)

- r1: $b = f, a \wedge b = t, a \rightarrow c = t, c \rightarrow d = t, d \rightarrow e = t, b = t \Rightarrow a = t$ (aplico *exchange*)
  - $a = t, b = f, a \wedge b = t, a \rightarrow c = t, c \rightarrow d = t, d \rightarrow e = t, b = t \Rightarrow c = t$ (aplico *exchange*)
  - $c = t, a = t, b = f, a \wedge b = t, a \rightarrow c = t, c \rightarrow d = t, d \rightarrow e = t, b = t \Rightarrow d = t$ (aplico *exchange*)
  - $d = t, c = t, a = t, b = f, a \wedge b = t, a \rightarrow c = t, c \rightarrow d = t, d \rightarrow e = t, b = t \Rightarrow e = t$
  - $\top$ (rama abierta)
- r2: $b = f, a \wedge b = t, a \rightarrow c = t, c \rightarrow d = t, d \rightarrow e = t, b = t \Rightarrow b = t$
- $\bot$

La deducción era válida y se obtuvo una refutación, pero se exploró una rama del árbol que hizo más lento el proceso. Sin ninguna guia, el algoritmo toma la primer rama de la regla y la explora. Ademas, no se verificó que la fórmula B que se agrega al conjunto de hipótesis era átomica ($b = t$), y que por lo tanto podía hacerse una verificación de consistencia similar al realizado por la regla axiom, haciendo mas eficiente el proceso.

3 SNR

La investigación consistió en el análisis del método de General Matings, SNR, y resolución no-clausal, buscando obtener una integración y equivalencia entre los tres métodos y sintetizarla en términos de una función heurística que guie el proceso de búsqueda de SNR. De los métodos expuestos, General Matings es el único que trabaja explícitamente con heurísticas (operando en su propia estructura de datos). Si bien la formulación del problema y su estructuración difiere respecto a SNR, es posible extraer estas heurísticas y traducirlas a los términos de SNR para aplicarlas en estos. El objetivo es seleccionar fórmulas que aceleren el proceso de refutación, de tal manera que se alcance en menos pasos un contrajemplo, o bien se determine la imposibilidad de encontrarlo.

A continuación se realiza una síntesis de los razonamientos mas importantes y las exploraciones realizadas para alcanzar el objetivo propuesto.

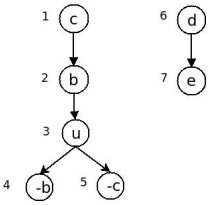
3.1 Diferencias y equivalencias entre General Matings y SNR

Para poder integrar los métodos y realizar equivalencias entre las ventajas que cada uno tiene, buscando la integración de estas, es necesario en primer lugar tener en claro la diferencias entre los dos enfoques.

El método de General Matings en escencia, realiza una búsqueda de prueba y error. El espacio de búsqueda se acota mediante el uso de estructuras vgraph y hgraph, y el proceso de búsqueda se refina con técnicas de poda, aprendizaje dirigido por conflictos, non-cronológical backtracking y deducción unitaria. Para un estudio en profundidad del método, que se hace totalmente necesario para la completa comprensión del presente trabajo, es necesario la lectura de la versión extendida de la especificación de General Matings.

3.1.1 Limitaciones de Resolución no-clausal

El tercer método utilizado en SNR es resolución no-clausal, a continuación se evalúan las limitaciones que tiene al aplicarse en SNR, de la forma especificada. Al aplicar resolución no-clausal entre un átomo y una fórmula, las reglas de simplificación pueden dar lugar a una fórmula que sea contradictoria o con términos falsos, que pasan inadvertidas. Por ejemplo, sea una fórmula $F = c \wedge b \wedge u \wedge (\neg a \vee \neg b \vee \neg c) \vee (d \wedge e)$ y un átomo $a = t$, la operación de resolución no-clausal resulta en $Res(F, a) = c \wedge b \wedge u \wedge (\neg b \vee \neg c) \vee (d \wedge e)$. Si creamos el grafo de vertical paths, se obtienen dos componentes conexas:



De la observación de los vpaths, se evidencia que al intentar extender el CRP a partir del nodo 1 (en notación del método General Matings, CRP<1>), se obtiene siempre una valuación contradictoria. Esta situación es descripta y detectada como conflicto local en General Matings. Semanticamente $c \wedge b \wedge u \wedge (\neg b \vee \neg c)$ es una subfórmula lógicamente equivalente a $\perp$, que puede ser reemplazada en la fórmula completa y aplicando las regla de reducción $rdp(\perp \vee F) = F$ se obtiene $F = (d \wedge e)$. Sin embargo, el algoritmo SNR no realiza esta simplificación.

Queda en evidencia que la resolución no-clausal no realiza todas las simplificaciones posibles, y que a su vez, no simplifica lo que en General Matings se denomina conflicto local.

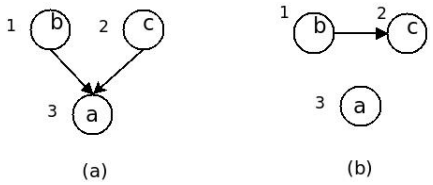
Una gran diferencia entre General Matings y SNR es que en General Matings la asignación de átomos es en base a la estructura de grafos, siempre comenzando con nodos raíz y buscando extender ese camino hasta un nodo hoja. Bajo este esquema, el algoritmo de detección de conflicto local funciona, ya que fue diseñado para esa estructura de datos particular. Sin embargo en SNR ocurren asignaciones atómicas que no siempre siguen el orden estructural de las fórmulas, por lo tanto un objetivo es evitar que la resolvente de una resolución no-clausal sea una fórmula con conflicto local, o bien detectarlo y aplicar la simplificación que corresponda.

3.1.2 Asignación atómica

El principal análisis que llevó a la construcción de la heuristíca se dio en la forma de creación del vgraph y su equivalencia en SNR cuando es necesario elegir una nueva fórmula a descomponer. En primer lugar, definamos la equivalencia mas evidente:

equivalencia 1: una asignacion atómica en SNR es equivalente a la valuación de un literal en General Matings.

En primera instancia, esta es una equivalencia conceptual entre estas dos operaciones, ya que algorítmicamente hay que tener en cuenta el uso de toda la información disponible por el sistema SNR para realizar las asignaciones atómicas que se deduzcan en un estado de demostración, para ser realmente equivalente a General Matings, que considera esta cuestión. Por ejemplo, si se tiene en el consecuente la siguiente fórmula $[P][PA] \Rightarrow (b \vee c) \wedge a = t$, al aplicar $gt : \wedge$ se obtiene $[a = t, P][..] \Rightarrow (b \vee c) = t$, como primer alternativa a explorar. Sin embargo $a = t$ es agregada a la lista P y tratada como subfórmula, siendo que es atómica y debería ser agregada a PA. En General Matings, la asignacion atómica se realiza mediante la deducción unitaria sobre la estructura hgraph que codifica una cnf.



(a) vgraph, hgraph (b) de la fórmula $(b \vee c) \wedge a$

La conclusión es que, para que la **equivalencia 1** sea verdadera, es necesario verificar si una fórmula que pasa a las hipótesis al aplicar una regla de inferencia es átomica. Esta verificación se puede realizar sin problemas, su costo computacional no afecta la complejidad algorítmica.

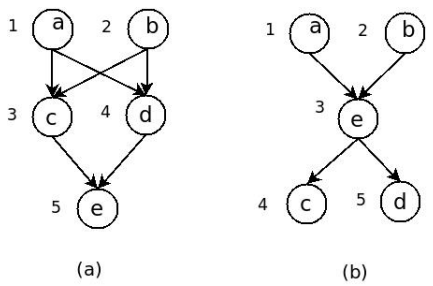
3.1.3 Vgraph en SNR

En SNR, todas las fórmulas del sistema en un estado de demostración se consideran verdaderas, la demostración por refutación está dada por la asignación del valor de verdad =f a la fórmula inicial o goal, y luego en la asignación del valor de verdad de cada subfórmula (=t o =f). Existen dos tipos de reglas de inferencia: strech o split, directamente relacionadas con los operadores lógicos $\wedge$ y $\vee$ en su polaridad positiva (=t) respectivamente, de esta forma se evidencia

que no existe gran diferencia entre los sistemas en interpretación de operadores y ramificación de alternativas, en este punto podemos pensar en la demostración prescindiendo del hecho de que SNR es un método por refutación y General Matings no, salvo por la condición de corte.

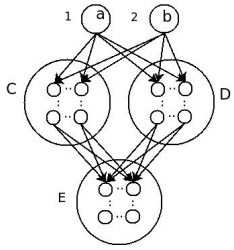
En efecto, SNR busca una contradicción, rama cerrada o valuación insatisfacible, y retorna exitosamente al encontrarla; mientras que General Matings busca una valuación satisfacible (rl-path) y retorna exitosamente al encontrarla. Por lo tanto el uso de la estructura vgraph (y su representación equivalente) varía ya que tienen objetivos de búsqueda ortogonales. Al ser SNR un método por refutación, buscará la insatisfacibilidad de cada rl-path del vgraph (en términos de General Matings). Por lo tanto, es de interés preguntarse como codifica SNR una estructura de vgraph. Para comenzar con este ánalisis se especifican dos características distintivas entre los dos métodos.

Es necesario notar que, SNR es un método Top-down, que comienza con la fórmula completa y la descompone, mientras que General Matings es esencialmente un método Botton-up, ya que parte de asignar valuaciones de literales y busca encontrar una valuación que satisfaga la fórmula completa. Es importante notar que en General Matings, la estructura vgraph es dependiente del orden de los términos en una fórmula. Por ejemplo sea $F_1 = (a \vee b) \wedge (c \vee d) \wedge (e)$ y $F_2 = (a \vee b) \wedge (e) \wedge (c \vee d)$ sus respectivos vgraph son:



vgraph de F_1(a) y vgraph de F_2(b)

Donde F_1y F_2 son lógicamente equivalentes. ¿En que afecta el orden?: en como se expande el CRP y se asignan valuaciones; y si se analiza en términos de subfórmulas o subgrafos, determina el subgrafo que se explorará para expandir el CRP. El ejemplo particular no es representativo de las implicancias mencionadas, ya que la valuación $v(e) = t$ se deduce mediante deducción unitaria sobre hgraph y el orden de los términos $(a \vee b)$ y $(c \vee d)$ no altera significativamente las posibilidades de exploración. Sin embargo, si c, d y e son subfórmulas denominadas C, D, y E obtenemos $F = (a \vee b) \wedge (C \vee D) \wedge (E)$. Al aplicar la lógica de creación del vgraph, obtenemos:



En este caso, no hay deducción unitaria posible, ya que se desconocen las subfórmulas. Si bien siempre se conoce la fórmula completa, no es posible (por la complejidad computacional que conlleva) realizar ciertos análisis o verificaciones, lo que en términos prácticos equivale a que el algoritmo debe tomar decisiones prescindiendo de cierta información, o desconociendo la composición de subfórmulas en un estado dado de la demostración.

De la observación de este caso, se nota que podría ser conveniente evaluar primero E, y luego eligir C o D para expandir el CRP, ya que E será evaluada independientemente de si se transita por el subgrafo C o D. Obviamente, también puede ocurrir que E añada información que no es utilizada para obtener la refutación, y que la refutación suceda con la información de C y D, de manera independiente, en tal sentido vale recordar que se esta buscando una función heurística, y no un algoritmo deterministico de mayor eficiencia.

3.1.4 Asociatividad en SNR

Una cuestión importante, que si bien no se aplicó como modificación al algoritmo original de SNR, es la asociatividad de las reglas. Se mostrará con un ejemplo la importancia e implicación, señalando que es una etapa del algoritmo en donde se podría aplicar la heurística diseñada para obtener demostraciones mas cortas.

En demuba2, al tomar como entrada la fórmula $F = (a \vee b) \wedge (C \vee D) \wedge (E)$, se realiza el siguiente proceso de demostración (asociatividad a izquierda):

- $[] : [] \Rightarrow (a \vee b) \wedge (C \vee D) \wedge (E) = t$
- $[(C \vee D) \wedge (E) = t] : [] \Rightarrow (a \vee b) = t$
  - $[(C \vee D) \wedge (E) = t] : [] \Rightarrow a = t$
  - $[(C \vee D) \wedge (E) = t] : [a = t] \Rightarrow (C \vee D) \wedge (E) = t$
  - $[E = t, (C \vee D) \wedge (E) = t] : [a = t] \Rightarrow (C \vee D) = t$
  - ...

con asociatividad a derecha del operador $gt : \wedge$, se realiza la siguiente descomposición:

- $[] : [] \Rightarrow (a \vee b) \wedge (C \vee D) \wedge (E) = t$
- $[E = t] : [] \Rightarrow (a \vee b) \wedge (C \vee D) = t$
- $[C \vee D = t, E = t] : [] \Rightarrow (a \vee b) = t$
  - $[C \vee D = t, E = t] : [] \Rightarrow a = t$

En el caso de asociatividad a derecha, se da el caso en que el algoritmo tiene que decidir una nueva fórmula, y dispone de dos alternativas $C \vee D = t$ y $E = t$. Este caso es ilustrativo de la importancia de la asociatividad de los operadores, ya que definen el espacio de búsqueda o decision en donde posteriormente se aplica la regla elegir, y por ende, la heurística.

Aqui el demostrador toma la opción de descomponer primero $C \vee D$, es decir evaluar esta ramificación. De manera análoga al método de General Matings, este criterio es el orden de aparición de los términos en la fórmula.

En donde no resulta trivial que subfórmula o subgrafo conviene explorar primero. Sin embargo podemos explorar E primero sabiendo que es un subgrafo obligatorio. Pero si $F = (a \vee b) \wedge (C \vee D) \wedge (E \vee F)$, el criterio anterior no aplica.

Esta problemática tiene una gran similitud a la problemática de elección de la siguiente fórmula a descomponer en SNR. Otra forma de definir una mejora en la búsqueda, es plantear la elección de la subfórmula basada en el criterio heurístico en lugar de usar una asociatividad determinada por el orden de escritura de las subfórmulas.

3.2 Propuesta de mejora 1

En primer lugar se pensó en un sistema de identificación de subfórmulas, que permita especificar su interrelación con las demas fórmulas, es decir, cual es la fórmula que contiene a una subfórmula, y su nivel de profundidad. En este esquema, a cada subfórmula se le asigna el conjunto de átomos válidos. El proceso de asignacion se realizaría de forma estática y en complejidad temporal **O(n)**.

Por ejemplo, tomando la fórmula $F = c \wedge b \wedge u \wedge (\neg a \vee \neg b \vee \neg c) \vee (d \wedge e)$ se la recorre recursivamente mediante un algoritmo top-down y se generan identificaciones para todas las subfórmulas (atómicas y no-atómicas).

1. $F = c \wedge b \wedge u \wedge (\neg a \vee \neg b \vee \neg c) \vee (d \wedge e) : \ F_1 \vee F_2$
2. $F_1 = c \wedge b \wedge u \wedge (\neg a \vee \neg b \vee \neg c)$: $F_{11} \wedge F_{12} \wedge F_{13}$
3. $F_{11} = c$, $F_{12} = b$, $F_{13} = u$
4. $F_{13} = (\neg a \vee \neg b \vee \neg c) : F_{131} \vee F_{132} \vee F_{133}$

5. $F_{131} = \neg a$, $F_{132} = \neg b$, $F_{133} = \neg c$
6. $F_2 = (d \wedge e)$: $F_{21} \wedge F_{22}$
7. $F_{21} = d$, $F_{22} = e$

Lista final de identificadores de fórmulas atómicas:

$TF_{11} = c$

$TF_{12} = b$

$TF_{13} = u$

$TF_{131} = \neg a$

$TF_{132} = \neg b$

$TF_{123} = \neg c$

$TF_{21} = d$

$TF_{22} = e$

Basicamente, se trata de construir un árbol, que representa la información de una manera conveniente a los fines propuestos, y permite un conjunto de operaciones sobre el mismo.

3.2.1 Estructura de árbol

Cada nodo del árbol representa una subfórmula, y contiene como datos un conjunto de átomos válidos para esa subfórmula y una lista de conjunciones o disyunciones (excluyente) que componen a la fórmula, además de la información inherente a la estructura de datos árbol. Para evitar confusión, se aclara que un átomo válido tiene un valor de verdad constante y que es deducción lógica de la fórmula en cuestión a la cual pertenece, es decir se utiliza la definicion estricta de validez.

En primer lugar, todo átomo es válido en el ámbito de la subfórmula mas simple que lo contiene, a partir de esta premisa se busca aplicar una estrategia buttom-up de deducciones, utilizando las reglas de inferencia. Luego se busca propagar de forma top-down los átomos válidos en las fórmulas adyacentes.

Dada la estructura, definimos reglas de propagación de átomos:

1. Si una subfórmula esta añadida en conjunción desde su padre, por ejemplo $F = a \wedge (\neg b)$ tomando la subfórmula $\neg b$, entonces los átomos válidos para esa subfórmula son propagados hacia el padre. Es decir, el alcance de $\neg b$ es F.

2. Si una fórmula F esta compuesta por una disyunción de otras subfórmulas, entonces los átomos válidos para F son propagados hacia las fórmulas hijo. Por ejemplo: $F = a \wedge (\neg a \vee b)$ recibe el átomo a por la **regla 1,** y la subfórmula $F_2 = (\neg a \vee b)$ recibe el átomo a, que luego es propagado hacia la subfórmula $\neg a$ y la subfórmula b. Es decir, el alcance de a es F, que incluye a todas sus subfórmulas.

3. Una contradicción se da cuando en una fórmula tengo ambos átomos con distinta polaridad.

4. Si una fórmula F_1 es contradictoria, y $F_p = F_1 \wedge F_2$ donde F_2 es otra subfórmula cualquiera, entonces el alcance de la contradicción es F_p.

El algoritmo consiste en recorrer todos los nodos hojas, que representan siempre fórmulas atómicas, aplicar las reglas de propagación para determinar el alcance de los mismos, y encontrar contradicciónes que permitan la reducción o falsificacion de la fórmula.

3.2.2 Relación con reglas de inferencia de Secuentes

Las reglas de propagación recuerdan a las reglas de inferencias del sistema G de secuentes. La diferencia es que Secuentes usa una estrategia Top-down mientras que en las reglas de propagación se aplica también una estrategia Bottom-up. El uso de la deducción local/reglas propagación posibilita al algoritmo del demostrador "mirar hacia adelante" ya que el procesamiento que realiza la deducción local transforma la fórmula en una fórmula lógica y gramaticamente equivalente pero más chica, antes de seguir descomponiendo la fórmula con las reglas de inferencia, haciendo más corto el árbol de demostración que es el objetivo principal.

3.2.3 Especificacion del algoritmo

A alto nivel, el algoritmo esta compuesto por tres rutinas, **propagarArriba**, **propagarAbajo** y **propagarContradicción**.

- n denota a un nodo
- *conjunción(n)* es verdadero si n es un nodo del tipo conjunción

Algorithm 1 propagarArriba(n)

```
if conjuncion(n.padre) then
    if (n.tomos ∩ ¬(n.padre.tomos) = ∅) then
        n.padre.tomos = n.tomos ∪ n.padre.tomos;
        propagarArriba(n.padre);
        propagarAbajo(n.padre);
    else
        propagarcontradiccion(n.padre);
    end if
end if
```

PropagarArriba(n) verifica que el nodo padre de n sea del tipo conjunción ($\wedge$), esto implica que los átomos de n serán válidos también en *n.padre*. Luego se verifica si algun átomo de n existe con polaridad opuesta en *n.padre*, caso positivo se alcanzó una contradicción, y se procede a propagar la misma.

Algorithm 2 propagarAbajo(n)

```
for all m ∈ n.subfrmulas do
    if (m.tomos ∩ ¬(n.tomos) = ∅) then
        m.tomos = m.tomos ∪ n.tomos;
        propagarAbajo(m);
    else
        propagarcontradiccion(m);
    end if
end for
if size(n.subfrmulas) == 1 then
    n.tipo = conjuncion;
    propagarArriba(n.subfrmula);
end if
```

PropagarAbajo(n) actualiza recursivamente los átomos válidos del subárbol con raiz en n, independientemente del tipo de cada nodo. Este método es llamado cuando ocurre una nueva asignación atómica en n, dado que la propagación de la/las asignaciones atómicas puede dar lugar a deducciones en las subfórmulas de n. Si una subfórmula tiene algun átomo válido con polaridad opuesta en relación al padre, entonces ocurre una contradicción que será propagada. Si luego del proceso de propagación y eliminación de subfórmulas contradictorias, el nodo queda con un sola subfórmula, se la tratará como conjunción (si fuese disyunción) y se propagaran hacia arriba los átomos válidos de esa subfórmula.

Algorithm 3 propagarcontradicción(n)

```
if conjuncion(n.padre) then
    propagarcontradiccion(n.padre)
else
    eliminar(n);
end if
```

Propagarcontradicción(n) propaga una contradicción hacia arriba si n.padre es del tipo conjunción, ya que esto significa que un término de un operador $\wedge$ es $\bot$, por lo tanto la fórmula completa es $\bot$. Si el n.padre es un nodo del tipo disyunción ($\vee$), el término o subfórmula $n \cong \bot$ es eliminado de la subfórmula *n.padre* ya que la subfórmula no puede ser verdadera.

3.2.4 Etapas

Al pensar en una posible implementación de los algoritmos descriptos, surge la idea de dividir la funcionalidad en tres etapas, sin describir ninguna implementación particular.

Preprocesamiento Se parsea la fórmula generando la estructura de árbol, dicho algoritmo es independiente de los otros procedimientos descriptos a continuación y puede ser intercambiado/mejorado.

Procesamiento Se toma la lista de nodos hojas y se aplica las reglas de propagación. Se van eliminado nodos contradictorios y registrandolos para el postprocesamiento.

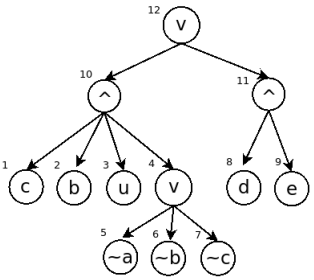
Si tomamos como base el contexto de la resolucion no-clausal, podemos acotar dicha propagación desde la(s) subfórmula(s) afectadas por la ocurrencia del átomo con el que se aplica la resolución, haciendo mas eficiente el procedimiento.

Posprocesamiento Consiste en tomar la lista de nodos eliminados y realizar las alteraciones de la fórmula escrita en lógica proposicional, para que de esta forma se pueda seguir utilizando en el algoritmo de demostración.

3.2.5 Ejemplo

$F = c \wedge b \wedge u \wedge (\neg a \vee \neg b \vee \neg c) \vee (d \wedge e)$

A continuación se muestra una representación gráfica de la estructura de árbol generada por la etapa de preprocesamiento. Inicialmente los nodos internos no contienen átomos valuados, solo los nodos hoja y un átomo por nodo.



Luego, en la etapa de procesamiento se realiza la propagación de átomos:

1. PropagarArriba(1), PropagarArriba(2), PropagarArriba(3)
   átomos(10) = [c,b,u]
2. PropagarAbajo(10)
   átomos(4) = [c,b,u]
3. PropagarAbajo(4)
   (a) PropagarAbajo(5)
   (b) PropagarAbajo(6), eliminar(6)
   (c) PropagarAbajo(7), eliminar(7)
   (d) 4.tipo = conjunción
   (e) PropagarArriba(5)
   (f) átomos(10) = [c,b,u,-a]
4. PropagarArriba(8)
5. PropagarArriba(9)

Obteniendose una estructura que representa la fórmula reducida, en la etapa de posprocesamiento:
$F = c \wedge b \wedge u \wedge \neg a \vee (d \wedge e)$

3.2.6 Análisis

A continuación se realizará un análisis de la propuesta de mejora 1.

Equivalencia con conflicto local Un vpgraph (vertical path graph) codifica en su estructura las valuaciones potenciales para una fórmula dada; codificando las satisfacibles como también aquellas que son contradictorias, por contener al menos un átomo con polaridad opuesta. El algoritmo de detección de conflicto local trabaja sobre esta estructura para detectar contradicciones en los vertical paths. Dado un conjunto de asignaciones atómicas (SNR) o CRP (General Matings), que determina el estado actual de demostración y una nueva asignacion atómica, el algoritmo determina si existe al menos un vertical path safisfacible que contenga a los átomos previos y el nuevo átomo (es decir, prueba el path actual); caso contrario ocurre un conflicto local. Es importante aclarar que el hecho de que exista conflicto local no implica que la fórmula sea insatisfacible en el estado actual del demostrador, sino que la combinación de ramas obtenida no puede ser extendida con el átomo propuesto y mantener satisfacibilidad en la construcción de un path especifico, pero sí puede existir otra valuación independiente de esta que haga satisfacible la fórmula.

Un hpgraph codifica en su estructura valuaciones que hacen contradictoria la fórmula. Un horizontal path de un solo átomo es equivalente a una deducción unitaria, ya que ese átomo debe ser asignado de lo contrario hay una contradicción global. Esta situación también se ve reflejada de forma equivalente en el vpgraph, como se describe a continuación.

Estructuralmente, es de interés analizar aquellos nodos de las subfórmulas que son puntos de inflexión, es decir que cualquier valuación que satisfasga a la subfórmula los debe contener. Encontrarlos es simple ya que son aquellos átomos que estan en conjunción con otra subfórmula,

simbolicamente $F = A \wedge F_1$

donde A es un átomo y F_1es una subfórmula.

El problema de este análisis es que si la subfórmula $F = A \wedge F_1$ es parte de una fórmula $F_0 = (A \wedge F_1) \vee F_2$ el alcance del átomo A no es F_0 , A pierde su propiedad de punto de inflexión para la fórmula F. Sin embargo bajo el ámbito de la subfórmula F_1 el átomo A no pierde validez, es decir es punto de inflexión. Este caso sugiere trabajar con subfórmulas explicitamente para realizar deducciones locales y simplificar las fórmulas. Esta aproximación se realiza con la detección de conflicto local sobre el vhpath. La diferencia radica en que el método de General Matings trabaja con la subfórmulas de forma implicita: toma un conjunto de valuaciones y aplica los algoritmos sobre las estructuras. Conceptualmente y algoritmicamente, el método de General Matings trabaja siempre con la fórmula entera y un conjunto de valuaciones.

Deducción local En el contexto de este trabajo, se denomina deducción local al enfoque inspirado en la detección de conflicto local que a su vez busca operar explicitamente con la subfórmula, encontrando si es posible una fórmula

lógicamente equivalente simplificada. Esto es utilizable porqué el estado alcanzado por el algoritmo de demostración es realizado siempre por inferencias y no por backtracking, por lo tanto siempre se profundizará por una rama de demostración, a diferencia de General Matings que realiza una combinatoria (guiada) de valuaciones. A su vez, esto es posible por que se usa la estrategia de conjunto de soporte, en donde se presuponen las hipótesis consistentes y que la causa de inconsistencia se encuentra en la tesis (goal).

Conflicto global o deducción unitaria El conflicto global en General Matings, es un caso particular de una deducción unitaria en la fórmula entera, que genera contradicción con la asignación previa de átomos. Por lo tanto, conceptualmente no va mas allá que una deducción unitaria.

Otros casos El caso que contempla el conflicto local y que no contempla el conflicto global es irrelevante, por que se trata de valuaciones que no afectan a la fórmula derivada al aplicar las valuaciones anteriores. Son casos en donde se intenta valuar una subfórmula que ya es contradictoria, por lo tanto esa valuación no aporta nada (asigna una variable libre).

Si se examina la composición de subfórmulas que ocurre en la construcción del vpgraph, se nota que dos subfórmulas en conjunción $F_1 \wedge F_2$ generan $l_1.r_2$ aristas en el grafo donde l1 es la cantidad de nodos leaf(hoja) de F1 y r2 la cantidad de nodos raíz de F2. Semanticamente, para satisfacer la fórmula compuesta, debe existir un path que satisfasga F1 y uno que satisfaga F2 que a su vez no tengan átomos opuestos. La eleccion del orden de F1 y F2, y por lo tanto la forma del vhpgraph generado es arbitraria.

3.2.7 Conclusión

La propuesta desarrollada resulta interesante como integración de los métodos base, pero excede al alcance del trabajo de tesis, ya que da lugar a un enfoque diferente a SNR, en donde se plantear nuevas dificultades y la necesidad soluciones a las mismas. El algoritmo propuesto podria ser util como base de desarrollo, pero necesita seguir profundizandose ya que es un planteo inicial de alto nivel.

4 Síntesis

Una definición util para formular la heurística desarrollada es la **cantidad de ramas** de una fórmula lógica, definida como el numero máximo de valuaciones posibles que hacen la fórmula satisfacible, o equivalentemente una cota superior al numero de valuaciones que hacen la fórmula satisfacible.

Es deseable que la ramificación sea lo mas reducida posible, como se plantea en SNR, por lo tanto la propuesta de heurística consiste en la elección de la fórmula con la menor cantidad de ramas, en principio del subconjunto de fórmulas que contengan el átomo alcanzado en el consecuente, con polaridad inversa respecto a este.

En escencia, el método se resume en:

Utilizar la función **cantidad de ramas** como aproximación de la cantidad de valuaciones que hacen verdadera a una fórmula, en complemento con resolución no-clausal para obtener fórmulas reducidas en base al conjunto de átomos asignados en un estado particular de la demostración.

El problema de calcular la cantidad de valuaciones que hacen a una fórmula satisfacible equivale al problema de satisfacibilidad de la fórmula lógica (SAT). Sin embargo es posible realizar una aproximación, considerando sólo la ocurrencia de variables proposicionales y conectores lógicos, con independencia de su valor e identificación, analizando aspectos estructurales de la fórmula.

La aproximación se vuelve utilizable en el demostrador al complementarla con resolución no-clausal, idea central del método SNR, pero en este caso ha de aplicarse a todas las fórmulas candidatas en el consecuente del secuente, o desarrollar una técnica equivalente.

Si bien esta aproximación parece poco útil, ya qué de esta manera la cantidad de valuaciones calculadas puede distar mucho del valor real, ya que esta cantidad es condicionada por las asignaciones de las variables proposicionales y las contradicciónes inherentes en la fórmula; vale resaltar que las fórmulas están reducidas a priori por las operaciones de resolución no-clausal con base a los átomos asignados previamente en el proceso de refutación; factor que reduce considerablemente la fórmula y consecuentemente la cantidad de valuaciones posibles que la satisfacen, por eliminación de variables.

De esta manera el cálculo se vuelve trivial, ya que una fórmula con operadores $\wedge$ y $\vee$ se transforma en una expresión algebraica de producto y suma de números 1. Por ejemplo:

$(a \vee b) \wedge (b \vee c) \equiv (p \vee p) \wedge (p \vee p) \equiv (1 + 1) * (1 + 1) = 4$

$(a \wedge b \wedge c) \vee (a) \equiv (p \wedge p \wedge p) \vee (p) \equiv (1 * 1 * 1) + 1 = 2$

Part II

Implementación final

La implementación final para la tesis de grado, consiste en la mejora del Demostrador UBA version 2 (Demuba2) que implementa el método SNR, introduciendo la función heurística desarrollada, y la implementación de todo lo

requerido para su eficiente y eficaz funcionamiento, dando origen a Demuba3. Por tal motivo, en la siguiente sección se desarrolla una explicación del funcionamiento de Demuba2, escrito en Prolog.

A partir de ahora, se hará distinción entre estas dos definiciones:

Se define a la función que calcula el valor "cantidad de ramas" como "aproximación" o "función de aproximación", ya que establece una cota superior para la cantidad de valuaciones que hacen una fórmula dada verdadera.

Se define a la función que selecciona la siguiente fórmula a intercambiar en la aplicación de la regla replace del demostrador, como "función heurística".

La función heurística usa la estructura de índices para obtener la fórmula con menor valor en la función de aproximación, de aquellas que contienen el átomo que ocurre en el consecuente con polaridad opuesta. Si ninguna fórmula tiene el átomo, se considera el conjunto total de fórmulas, esta modificación a SNR es necesaria a partir del fenomeno "indexación de átomos oscurecidos" y la imposibilidad de implementarlo sin realizar un efoque diferente en el tratamiento de fórmulas. Esta cuestión mencionada hace que tenga sentido separar conceptualmente función heurística y función aproximación, y probablemente de lugar a futuros desarrollos.

A partir de analizar los diferentes demostradores y técnicas, surge la propuesta de heurística como una posible síntesis de las ventajas que cada uno ofrece. La heurística para ser eficientemente aplicada requiere de una serie de modificaciónes y mejoras al demostrador Demuba2 (SNR), dando origen a Demuba3.

5 Demuba2

Demuba2 es la implementación en Prolog propuesta por el método SNR. Consta de tres grupos de predicados: reglas de demostración, resolución no-clausal y predicados auxiliares.

Un estado de demostración, o estado del demostrador se define como sigue:

P:PA => (T=V)

donde P es una lista de hipótesis, PA es una lista de asignaciones atómicas y $(T = V)$ una fórmula con una polaridad determinada V ($V = t$ o $V = f$). El operador => es utilizado para separar las hipótesis de la tesis, o en secuentes, el antecedente y el consecuente.

5.1 Reglas de demostración

regla 1:

P:PA => [(A & B=t)]:Z:strecht(regla_1,A & B=t >> A=t y B=t,T)

:- !,([B=t|P]:PA => [(A=t)]:Z:T; P:PA => [(B=t)]:Z:T).

regla 2:

P:PA => [(A & B=f)]:Z:split(regla_2,A & B=f >> A=f o B=f,T1,T2)

:- !,(P:PA => [(A=f)]:Z:T1, P:PA => [(B=f)]:Z:T2).

regla 3:

P:PA => [(A v B=t)]:Z:split(regla_3,A v B=t >> A=t o B=t,T1,T2)

:- !,(P:PA => [(A=t)]:Z:T1, P:PA => [(B=t)]:Z:T2).

regla 4:

P:PA => [(A v B=f)]:Z:strecht(regla_4,A v B=f >> A=f y B=f,T)

:- (!,[B=f|P]:PA => [(A=f)]:Z:T; !,P:PA => [(B=f)]:Z:T).

regla 5:

P:PA => [(- A =t)]:Z:strecht(regla_5, - A =t >> A=f,T)

:- !,P:PA => [(A=f)]:Z:T.

regla 6:

P:PA => [(A -> B=f)]:Z:strecht(regla_6,A -> B=f >> A=t y B=f,T)

:- (!,[A=t|P]:PA => [(B=f)]:Z:T; !,P:PA => [(A=t)]:Z:T).

regla 7:

P:PA => [(A -> B=t)]:Z:split(regla_7,A -> B=t >> B=t o A=f,T1,T2)

:- !,P:PA => [(B=t)]:Z:T1; !,P:PA => [(A=f)]:Z:T2.

regla 8:

P:PA => [(A <-> B=t)]:Z:split(regla_8,A<->B=t >>(A=t y B=t) o (A=f y B=f),T1,T2)

:- !,([B=t|P]:PA => [(A=t)]:Z:T1,[B=f|P]:PA => [(A=f)]:Z:T2).

regla 9:

P:PA => [(A <-> B=f)]:Z:split(regla_9,A<->B=f>>(A=t y B=f) o (A=f y B=t),T1,T2)

:- !,([A=t|P]:PA => [(B=f)]:Z:T1; P:PA => [(A=t)]:Z:T1), !,([B=t|P]:PA => [(A=f)]:Z:T2 ; P:PA => [(B=t)]:Z:T2).

regla 10:

P:PA => [(A=N)]:Z:contradicción(regla_10,A=N)

:- opuesto(A,N),!.

regla 11:

P:PA => [(A=N)]:Z:replace(regla_11,A=N,EX,FX,T)

:- !,elegir1(A=N,P,FX,EX,PA), \+member(FX,P),\+member(FX,PA), P:[A=N|PA] => [FX]:Z:T.

5.2 Resolución no-clausal

```
r1(A & t,A):- !.
r1(A & f,f):- !.
r1(t & A,A):- !.
r1(f & A,f):- !.
r1(A v t,t):- !.
r1(A v f,A):- !.
r1(t v A,t):- !.
r1(f v A,A):- !.
r1(A -> t,t):- !.
r1(A -> f,- A):- !.
r1(t -> A,A):- !.
r1(f -> A,t):- !.
r1(- t,f):- !.
r1(- f,t):- !.
r1(t <-> A,A):- !.
r1(A <-> t,A):- !.
r1(f <-> A,- A):- !.
r1(A <-> f,- A):- !.
r(A & B,R):- !,r(A,RA),r(B,RB),r1(RA & RB,R),!.
r(A v B,R):- !,r(A,RA),r(B,RB),r1(RA v RB,R),!.
r(A -> B,R):- !,r(A,RA),r(B,RB),r1(RA -> RB,R),!.
r(- A,R):- !,r(A,RA),r1(- RA,R),!. r(X,R):- r1(X,R),!.
rs(A=V,NA=NV):- r(A,X),!,rdp(X=V,NA=NV),!.
rdp(- A = V, Z):- !,opuesto(V,OV),rdp(A = OV,Z). rdp(X,X).
```

6 Función de aproximación

En la parte I del documento se introdujo una noción de la función de aproximación, ahora se desarrollará completamente el algoritmo que implementa la función.

En el cálculo de la función de aproximación solo se procesan aspectos estructurales de la fórmula de entrada; conectores lógicos y ocurrencias de variables, sin ningún procesamiento semántico referido a las valuaciones. Explorando la fórmula respetando la precedencia de operadores y la composición en subfórmulas, mediante la aplicación de las siguientes reglas:

si $F = F_a \vee F_b$ entonces $r(F) = r(F_a) + r(F_b)$.

si $F = F_a \wedge F_b$ entonces $r(F) = r(F_a) * r(F_b)$.

si $F = p_x$ entonces $r(p_x) = 1$.

Obtenemos el valor **r** deseado. Incorporando el operador logico "$-$" se obtiene:

si $F = F_a \vee F_b$ entonces $r(F) = r(F_a) + r(F_b)$.

si $F = -(F_a \vee F_b)$ entonces $r(F) = r(-F_a) * r(-F_b)$.

si $F = F_a \wedge F_b$ entonces $r(F) = r(F_a) * r(F_b)$.

si $F = -(F_a \wedge F_b)$ entonces $r(F) = r(-F_a) + r(-F_b)$.

si $F = p_x$ entonces $r(p_x) = 1$.

si $F = -p_x$ entonces $r(-p_x) = 1$.

La lógica del cálculo no presenta dificultad, es análoga a la definición recursiva y a las equivalencias lógicas en lógica proposicional. Si se quiere incorporar los operadores " $\rightarrow, \leftrightarrow$ " basta con aplicar la equivalencia lógica de estos en relación a los operadores básicos y escribir las nuevas reglas de cálculo. En esta implementación estos operadores están descartados ya que los casos de tests generados no los utilizan, y al no incluirlos se implementa un algoritmo de cálculo simplificado que evita consideraciones de precedencia de operadores.

Implementación del algoritmo

Para procesar una fórmula aplicando las reglas mencionadas, existen a grandes rasgos dos aproximaciones: o bien se comienza desde la fórmula completa, descomponiendo recursivamente la fórmula, o bien se procesan los tokens de la fórmula de izquierda a derecha mediante un recorrido lineal y un stack. Mas información sobre este problema se puede encontrar en el funcionamiento de los analizadores sintaxicos de los compiladores. Se opto por la segunda manera, a continuación el pseudocodigo del algoritmo:

Algorithm 4 Cantidad de ramas

```
while t = nextToken do
    if t =' ∨' then
        s.top.operator = +;
    end if
    if t =' ∧' then
        s.top.operator = ∗;
    end if
    if t =' p∗' then
        saveAtom(AtomsList);
        if s.top.operator! = NULL then
            makeOperation(s, 1);
            s.top.operator ← NULL;
        else
            s.top.operand ← 1;
        end if
    end if
    if t =' (' then
        s.push();
    end if
    if t =')' then
        tmpOperand ← s.top.operand;
        s.pop();
        if s.top.operator! = NULL then
            makeOperation(s, tmpOperand);
        else
            s.top.operand ← tmpOperand;
        end if
        if t =' −' then
            change(s.top.polarity);
        end if
    end if
end while
```

La implementación en C++ se encuentra en **program/src/extern_predicates/h1.cpp**. La consistencia con la estructura y asociatividad de la fórmula de entrada, queda simplificada por la naturaleza de las reglas a aplicar para el cálculo del valor heurístico. Para visibilizar esto hay que notar que dada $F = A \wedge B$, el único caso en donde $r(B)$ altera, multiplica o incrementa $r(F)$ es si $r(B) > 1$. Si $(a) : B = p_x \rightarrow r(B) = 1$, si $(b) : B = p_1 \wedge p_2 \wedge p_n ... \rightarrow r(B) = 1$. Solo si $B = B_1 \vee B_2 \rightarrow (r(B) > 1)$, y en tal caso debe ser $F = A \wedge (B)$ (debe tener paréntesis) por precedencia de operadores, resultando en $F = A \wedge (B_1 \vee B_2)$.

Por lo tanto, si $F = A \wedge B$ (sin paréntesis), debe ser de la forma (a) o (b), lo cual implica que será una sucesión de átomos que no alteran el resultado

$(r(B) = 1)$. Esta simplificacion es posible debido a que todos los operandos son átomos, no subfórmulas.

El cálculo de la función de aproximación es equivalente a reemplazar los operadores lógicos por sus operaciones matemáticas y los átomos por 1. Las negaciones invierten la asignación de operaciones, sin mayor complejidad algorítmica. Por ejemplo:

$$(a \vee b) \wedge (b \vee c) = (p \vee p) \wedge (p \vee p) = (1 + 1) * (1 + 1) = 4$$

$$a \wedge b \wedge c \vee a = p \wedge p \wedge p \vee p = 1 * 1 * 1 + 1 = 2$$

Por este motivo no se requiere una lógica adicional en la interpretación de la fórmula para los diferentes operadores, sólo basta con realizar la operación asociada.

7 Función heurística

El objetivo de la función heurística es ofrecer una fórmula a intercambiar que haga el proceso de demostración mas corto. Esta fórmula es elegida considerando la función de aproximación sobre el total de fórmulas candidatas del consecuente.

8 Reducción de hipótesis

Para que la aplicación del método heurístico sea eficiente, las fórmulas deben estar reducidas mediante resolución no-clausal, aplicando los átomos asignados en un estado particular del proceso de demostración. Esta es una de las modificaciones necesarias al demostrador. Sin esta modificación, no se estaría utilizando información muy valiosa para guiar el proceso de búsqueda. Reducir las fórmulas permite un valor aproximado de mejor calidad, y un menor costo computacional para aplicar la función de aproximación en una fórmula.

Para obtener la fórmula con menor cantidad de ramas, es necesario aplicar la función de aproximación en todas las fórmulas candidatas, y por ende aplicar la reducción de hipótesis a cada fórmula. A priori, efectuar este procesamiento conlleva el aumento de la complejidad computacional, ya que hay que procesar n fórmulas cada vez que se busca aplicar la regla *replace*. Sin embargo con técnicas complementarias se puede reducir considerablemente la carga computacional requerida.

Hay que notar que, solo las fórmulas que contienen a cierto átomo A, serán reducidas y se recalculará el valor "cantidad de ramas". Las fórmulas restantes no intervienen en el proceso de elegir una fórmula substituta para A=N. Es posible modificar la función de aproximación para que devuelva el valor cantidad de

ramas y la lista de átomos que intervienen en la fórmula. Mediante una estructura de índices se puede acceder en un tiempo computacionalmente eficiente a las fórmulas que contienen un átomo especifico.

Predicados involucrados

El proceso de reducción de hipótesis se realiza cada vez que es necesario elegir una nueva fórmula a descomponer (es decir, se intenta aplicar la regla *replace*), mediante el predicado *reduceHs, fórmulasAfectadas, dHs, y rHs.* A continuación se especifican los mismos.

reduceHs(PA,P,PR,I)

Predicado que define como se realiza la reducción en términos de otros predicados auxiliares. *PA* es la lista de átomos con la que reducir, *P* el conjunto de fórmulas sin reducir, *PR* el conjunto reducido e *I* un puntero a la estructura de índices.

```
reduceHs(_,[],[],I).

reduceHs(PA,P,PR,I):- fórmulasAfectadas(I,FA),

dHs(P,PE,FA,0), rHs(I,P,PE,PA,FA,PR).
```

fórmulasAfectadas(I,FA)

Predicado externo escrito en C++ que retorna en *FA* las fórmulas afectadas por las asignaciones atómicas despues de la última reducción de hipótesis. Por motivos de eficiencia, *FA* es una lista de índices que referencia hipótesis en la lista *P*. El objetivo de este predicado es obtener las fórmulas que requieren reducción mediante resolución no-clausal.

dHs(P,PE,FA,O)

Elimina de la lista *P* las fórmulas referenciadas por sus índices en *[F|FA]*. La lista resultante es *PE*. La variable *O* (offset) es utilizada para corregir los índices en la lista al eliminar elementos en la lista *P*.

```
dHs(P,PE,[F|FA],O):- N is F+(-O),rem(N,P,PI),

NO is O+1,dHs(PI,PE,FA,NO).

dHs(P,P,[],_).
```

rHs(I,P,PE,PA,FA,PR)

Reduce todas las fórmulas de *P* que son indexadas en la lista de fórmulas afectadas *[FI/FA]*. Inserta la fórmula reducida *FR* en *PE*, y prosigue con las fórmulas indexadas en *FA* restantes. Si *FR* es tautología "*FR=(V=V)*", se elimina de la estructura de índices, caso contrario actualiza la información de indexación de la fórmula, recalculando la función de aproximación y obteniendo los átomos que componen la fórmula.

```
rHs(I,P,PE,PA,[FI|FA],PR):- nth1(FI,P,F), rs(F,FR,PA),
if_then_else(FR=(V=V), eliminarF(I,FI), actualizarF(I,FI,FR)),
ins(FR,PE,FI,PRL), rHs(I,P,PRL,PA,FA,PR).
rHs(I,_,PRL,_,[],PRL).
```

eliminarF(I,FI)

Elimina de la indexación la fórmula referenciada por *FI*. Se utiliza cuando una fórmula es reducida a tautología en la forma *t=t* o *f=f*, y no requiere indexación por no aportar información al proceso de refutación.

actualizarF(I,FI,FR)

Actualiza la información de indexación de la fórmula referenciada por el índice *FI*, donde *FR* es la nueva fórmula reducida que se utilizará para la actualización. Los únicos predicados que invocan a la función de aproximación son *actualizarF* e *indexarF*.

9 Estructura de índices

La estructura de índices se hace necesaria como complemento a la heurística, ya que realizar el cálculo heurístico para cada fórmula candidata de las hipótesis, y localizar todas las fórmulas candidatas es muy costoso en complejidad computacional, recordando que este proceso debe realizarse cada vez que aplicando reglas de descomposición se obtiene una fórmula atómica.
Los principales requerimientos son:

1. Dado un átomo, obtener el conjunto de fórmulas que contienen al átomo.
2. Dada una fórmula, obtener el valor heurístico.
3. Dado un conjunto de átomos, obtener el conjunto de fórmulas que contienen al menos uno de los átomos, con su polaridad original o invertida.

El caso (1) se requiere para obtener el conjunto de fórmulas candidatas, de las cuales se seleccionara aquella con menor valor heurístico. El caso (2) no requiere mayor explicación. El caso (3), se requiere para una optimización complementaria que consiste en mantener las hipótesis reducidas con base a la asignacion actual de átomos en el estado de demostración, la cual se detallara en la siguiente sección.

9.1 El Problema de la indexación de átomos "oscurecidos"

Al proponer las modificaciones al demostrador, se deducen las limitaciones del esquema, a continuación se detalla una limitación importante en relación a las posibilidades de Demuba 2, y se evalúa su impacto.

Una problematica aparente que ocurre en Demuba 3 es la siguiente: al aplicar resolución no-clausal entre una fórmula y un átomo, la resolvente puede eliminar sub-fórmulas tautológicas o contradictorias. Por ejemplo $Res(p1 = f, (p1 \vee p2) \wedge p2 \wedge (p3 \rightarrow p4) \vee p5) = p5$, eliminando $(p1 \vee \ p2) \wedge p2 \wedge (p3 \rightarrow p4) \equiv \bot$, y eliminando los átomos $p2$, $p3$ y $p4$. He denominado a este suceso oscurecimiento de átomos, ya que realmente los átomos forman parte de la fórmula original como variables libres, y su asignación no afecta en ningun caso a la tabla de verdad de la fórmula.

Si bien es cierto que la valuación de estos átomos no afecta la satisfacibilidad de la fórmula, si en algun punto del proceso de demostración reduzco una fórmula que oscurece átomos, y en un punto posterior obtengo en el consecuente uno de estos átomos con polaridad opuesta, no podría elegir tal fórmula reducida, que en su estado original contiene al átomo con polaridad opuesta, al respetar la restriccion original de demuba2.

Primera solución

Una primera solución propuesta consistió en registrar los átomos oscurecidos o eliminados cada vez que se reduce una fórmula y se vuelve a indexar. La operación consiste en realizar una diferencia entre conjuntos de átomos para obtener el resultado. Sin embargo este método funciona hasta que una fórmula pasa al consecuente y es descompuesta, ya que en la descomposición se trata como nuevas fórmulas a las fórmulas derivadas de la original, y no hay manera trivial de mantener trazabilidad en relación a cuáles átomos oscurecidos corresponden a cuales subfórmulas de la fórmula base. Por otro lado hay que notar que Demuba 3, al mantener todas las hipótesis reducidas con todos los átomos asignados en cualquier estado del demostrador, nunca tendrá dos veces el mismo átomo en el consecuente, reduciendo el alcance de este problema.

Se realizaron experimentos para comparar Demuba3 con y sin indexación de átomos oscurecidos, y si bien hay algunas mejoras para casos concretos, también hay varias perdidas de desempeño, medido tanto el tiempo de cómputo

como cantidad de reglas que aplicó el demostrador. A continuación se muestra el speedup logrado con indexación de átomos oscurecidos para el conjunto de fórmulas c5315.

Los experimentos corroboran la hipótesis de que no existe mejora real en la indexación de átomos oscurecidos, al menos de esta manera, en la que no mantengo la asociación de los átomos con las subfórmulas asociadas. Ademas, al priorizar átomos oscurecidos, que en verdad no aportan ninguna información relevante, se dejan de lado fórmulas que dan menor valor heurístico (con tendencia a refutaciones más cortas); esto explica la pérdida de desempeño en algunos casos importantes, por pertenecer a los casos procesados mas grandes.

Tambien se puede observar en los casos **c7552** que si bien la indexación no mejora el desempeño (speedup reglas=1), el desempeño en tiempo de computo empeora, producto del overhead de indexación de átomos oscurecidos.

Por lo tanto se decide descartar la indexación de átomos oscurecidos para está version del Demuba 3.

9.2 Implementación

La estructura de índices está implementada como predicado externo en C++. Para la implementación de las estructuras internas se usa la estructura de datos map de STL c++, que tiene una interfaz análoga a una estructura de hash table.

DirectMap

map<atomo, set<int>>

Una de las estructuras principales, que asocia a un átomo un conjunto de fórmulas afectadas por ese átomo, referenciadas por su índice entero.

ReverseMap

map<int, set<atomo>>

Estructura que codifica los átomos que contiene una fórmula, especificada por su índice.

atomosEliminados

map<int, set<atomo>>

Cada fórmula tiene un conjunto de átomos eliminados, desde su estado sin asignación de valores de verdad.

atomosEliminadosInvertido

map<atomo, set<int>>

Por cada átomo, mantiene un conjunto de fórmulas que contuvieron al átomo y que luego fue eliminado. (para un funcionamiento eficiente de elegir).

cacheAtomos

set<atomos>

Conjunto de átomos asignados para el estado actual del proceso de demostración.

HeuristicMap

map<int, unsigned_long>

Estructura que almacena por cada fórmula el valor heurístico asociado (cantidad de ramas).

formulas

map<int,term_t>

mapeo entre índices y fórmulas(puntero Prolog).

9.3 Métodos

mkEliminacion(int i, Eliminacion &e):

IndexF(PlTerm t):

Toma la fórmula en formato término de prolog, la parsea (aplica heurística), e introduce los valores en las estructuras. Inserta el valor heurísco (HeuristicMap), indexa por átomos encontrados (DirectMap) e inserta la lista de átomos para la fórmula (ReverseMap).

ActualizarF(int i, PlTerm t):

Vuelve a parsear la fórmula, obteniendo la cantidad de ramas y los átomos. Aplica diferencia entre conjuntos para obtener los átomos que han sido eliminados. Obtiene el conjunto anterior de átomos eliminados de la fórmula, y aplica union con los nuevos átomos eliminados para actualizar *atomosEliminados*. Actualiza los átomos actuales de la fórmula. Actualiza el valor heurístico en *HeuristicMap*. Quita los nuevos átomos eliminados en la indexación DirectMap. Inserta en *atomosEliminadosInvertido* cada átomo eliminado, asociado con la fórmula. Actualiza la cantidad de ramas en *HeuristicMap*. Por ultimo crea registro de actualización.

10 Modificaciónes al demostrador

10.1 Verificacion de fórmulas atómicas

Un requerimiento importante para la nueva versión del demostrador es, para cualquier estado del proceso de demostración, poder determinar el conjunto de átomos valuados. En el programa en Prolog, el consecuente está compuesto por dos listas: *P, PA*; Premisas, y premisas atómicas respectivamente. Al aplicar la regla *replace*, el átomo $A{=}N$ es reemplazado por una nueva fórmula del conjunto de premisas, y el átomo $A{=}N$ es agregado a la lista *PA*. De esta manera todas las fórmulas atómicas obtenidas mediante la regla *replace* se encuentran en *PA*, pero pueden ocurrir asignaciones atómicas por aplicación de otras reglas, por ejemplo:

Se tiene una fórmula con estructura (A & B=T) en el consecuente, la regla a aplicar es la siguiente:

P:PA => [(A & B=T)] :-

([B=T|P]:PA => [(A=T)]:Z:T; P:PA => [(B=T)]:Z:T)

En el caso de que B=T sea atómica, es agregada a la lista P y se buscará una refutación para ese sistema con (A=T) en el consecuente, perdiendo de esta forma información valiosa que es utilizable por la heurística y la reducción de hipótesis mediante resolución no-clausal.

Demuba 3 sin heurística

El demostrador sin heurística elige la primer fórmula con el átomo con polaridad opuesta, si no existe, simplemente la primer fórmula. Asi las fórmulas mas propensas a ser elegidas son las del conjunto de hipótesis inicial. Se modifico el algoritmo para darle prioridad a las últimas fórmulas añadidas, reduciendo asi la demostración y poder realizar una comparacion mas ajustada(y real) con Demuba 3.

10.2 Nueva regla replace

11 Generación de casos de tests

Es importante que los casos de tests sean representativos de la complejidad de la problemática, de lo contrario las experimentaciones perderían eficacia y no cumplirían la función de poner a prueba el funcionamiento del demostrador. Por esta razón se descartó generar fórmulas aleatoreas que podían no representar ninguna complejidad real para el demostrador, y se optó usar como base para la generación de fórmulas del conjunto de casos *ISCAS Benchmark Combinational Circuits 85'*.

Archivos ISCAS

La especificación de cada circuito está dada en un archivo de texto formato *Bench;* cada circuito posee un conjunto de entradas, un conjunto de salidas, buffers, operadores AND, NAND, OR, NOR, NOT, BUFF; especificados mediante identificadores. A continuación un ejemplo de un caso muy simple con tres entradas y dos salidas:

```
INPUT(1)
INPUT(2)
INPUT(3)
OUTPUT(4)
OUTOUT(5)
6 = NAND(1,2)
4 = OR(6,3)
5 = NOR(1,2,3)
```

Estos casos de prueba tienwen la propiedad de no retroalimentar estados mediante Flip-Flops, por lo que su conversión a una fórmula lógica es directa, cada fórmula de salida depende exclusivamente de una entrada dada, sin considerar "memoria" de estados anteriores. Los circuitos includos en *ISCAS 85'* son *c17, c432, c499, c880, c1355, c1908, c2670, c3540, c6288, c5315 y c7552.* Como cada circuito tiene un conjunto de salidas, se generan varias fórmulas lógicas por circuito.

Generador

Para obtener fórmulas lógicas en el formato requerido por el demostrador a partir de la especificación de circuitos se desarrolló un programa escrito en C++ que realiza esta tarea. Recibe dos parametros: *–infile* y *–outdir;* para indicar el caso de entrada en formato .bench y especificar el directorio de salida de los casos generados, respectivamente. Por ejemplo: *"generator –infile=./iscas85/c432.bench –outdir=./tests/c432"*. El programa generará un archivo por salida con el formato *[nombre_caso]_[id_salida]*, donde *id_salida* es el identificador que tiene la salida en el archivo .bench.

Funcionamiento

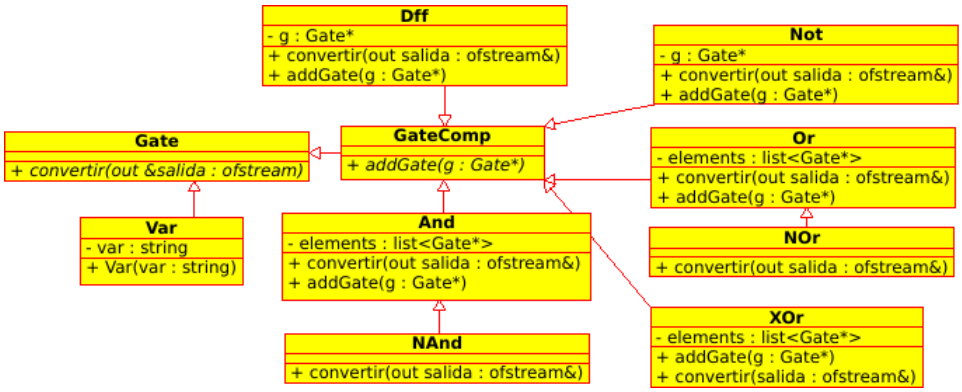
El funcionamiento del generador consta de tres etapas:

1. parseo de entradas y salidas.
2. parseo y construcción de estructura de las fórmulas.

3. generación de fórmulas.

Los pasos uno y dos son simples, consisten en reconocer la operación (AND, NAND, OR, NOR, NOT, BUFF) o especificación (INPUT, OUTPUT), y el conjunto de identificadores involucrados para generar una instancia de la clase correspondiente e incorporarlo a la estructura que representa la fórmula.

A continuación se presenta el diagrama de clases de la estructura de fórmulas:



El paso tres consiste en recorrer la estructura de fórmulas, generando la salida correspondiente por el stream de salida provisto como parametro.

Fórmulas generadas

Las fórmulas generadas abarcan un amplio espectro en relación al tamaño de entrada. Se obtienen fórmulas del orden de algunos bytes hasta gigabytes. Algunos fórmulas por ser muy pequeñas no son aplicables para realizar benchmarks, como las fórmulas obtenidas por *c499* (varias fórmulas del orden de 32 bytes) y *c17* (dos fórmulas, de 52 bytes y 128 bytes respectivamente). Tambien se obtienen casos mayores a 10gb en *c6288* que exceden el alcance de los demostradores a evaluar. Los casos mas acordes a los demostradores ejecutandose en una PC con prestaciones normales son *c880, c5315, c2670, c7552*.

12 Experimentación

Para una experimentación sistematizada y reproducible se programó un tester, que toma como entrada un conjunto de casos de test (especificado con un nombre de directorio en ./tests), un demostrador y si se quiere probar tautología o tomar las fórmulas tal como fueron generadas.

12.1 Medición de performance

Para la medición de la performance se utilizan dos métricas, tiempo de ejecución y cantidad de reglas que componen la demostración (si existe). La métrica "tiempo de ejecución", obviamente depende del hardware utilizado y de mejoras de performance que puedan realizarse sin alterar el algoritmo de demostración, pero resulta util para realizar experimentos con fórmulas no-tautológicas y analizar la constante multiplicativa de dos algoritmos con la misma complejidad **O(f(n))**.

Para la medición de tiempo se utiliza la librería *Chrono* de c++, que dispone de tres tipos básicos de relojes:

- System clock
- Steady clock

System clock utiliza el reloj interno de la BIOS, y no es util para medir intervalos de ejecución precisos. Utilizando este reloj los tiempos requeridos para tareas del sistema operativo, cambios de contexto, interrupciones y ejecución de otros procesos son considerados.

Steady clock es un reloj ajustado para la medición real en tiempo ejecución dentro de un mismo proceso, con independencia del sistema operativo y otros procesos. La aclaración de que este es el reloj utilizado para la medición es importante, ya que le da mayor precisión a los experimentos.

La implementación de la medición consta de las siguientes lineas, donde *query->next_solution()* es la invocación a la consulta al programa en Prolog desde C++ (tester.cpp).

```
auto start = chrono :: steady_clock :: now();
int ns = query-> next_solution();
auto end = chrono :: steady_clock :: now();
auto diff = end - start;
```

12.2 Prueba de correctitud

Es altamente importante que el demostrador determine si una fórmula es tautología, o si una deducción es válida. Como los casos de prueba generados son aleatorios, y la generación de casos tautológicos es una problemática aparte, se ideó una manera simple de generar una fórmula tautológica: $F\, v - F$, donde F es una fórmula cualquiera. En términos del demostrador, es probar que /- $F\, v - F$. El demostrador al no trabajar con comparacion de subfórmulas, debe verificar rama por rama que no hay manera de contradecir la fórmula completa.

12.3 Interfaz del tester

El tester es un programa escrito en C++, encargado de facilitar las experimentos con el demostrador, permitiendo definir el demostrador, el conjunto de casos de test, la librería de indexación y el tipo de demostración a ejecutar (demostración de las fórmulas originales o prueba de correctitud).

Argumentos:

- **–dem**=<demostrador> //nombre del demostrador, sin extensión ".pl"
- **–indexlib**=<libreria> //nombre de la librería, sin extensión ".so" o ".dll"
- **–set**=<conjunto> //nombre del conjunto de casos de prueba a utilizar
- **–type**=<tipo> //tipo de demostración: demostrar(fórmula original) o demostrart(prueba de correctitud)
- **–limit**=<limite> //valor entero, establece el limite de casos consecutivos a ejecutar
- **–basefile**=<archivo> //establece un archivo de salida previo, para continuar con las demostraciones.

La configuración de parametros permite la ejecución de pruebas sistematicas, facilitando el proceso de experimentación que suele requerir muchas comparaciones, variaciones de configuraciones, obteniendo los resultados de forma ordenada. El objetivo de **–limit** y **–basefile** es disponer de la capacidad de interrumpir, reanudar, y realizar pruebas incrementalmente, por ejemplo de a un caso por vez (con limit=1).

12.4 Organización

Dentro del subdirectorio tests, se encuentran los set de tests generados por *test_generator*. Cada test está identificado por un nombre con prefijo [idset] seguido de una numeración interna.

En el subdirectorio resultados, se encuentran los resultados de aplicar el tester a diversos sets con los diferentes demostradores. El formato de nombre de archivo es [idset]_[demostrador].txt y [idset]_[demostrador]_taut.txt si se probó correctitud.

Dentro de cada archivo:

- una prueba por linea, indicando: [test],[bytes],[0-1], [tiempo en milisegundos][cantidad reglas aplicadas]

0 indica falso, 1 indica verdadero.

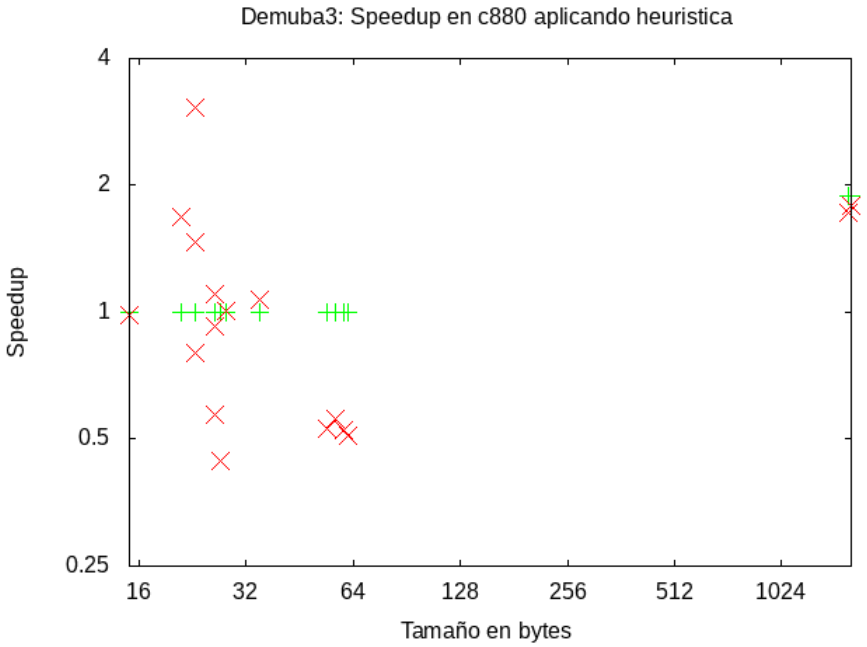
12.5 Resultados

La métrica utilizada es $S_p = t_{dem3_{nh}}(c)/t_{dem3}(c)$ para evaluar la mejora en tiempo de ejecución, y $S_p = r_{dem2}(c)/r_{dem3}(c)$ para evaluar la mejora en cantidad de reglas aplicadas por el demostrador. Observando las mejoras, se nota una clara interrelación entre ambas mejoras.

12.5.1 Set c880

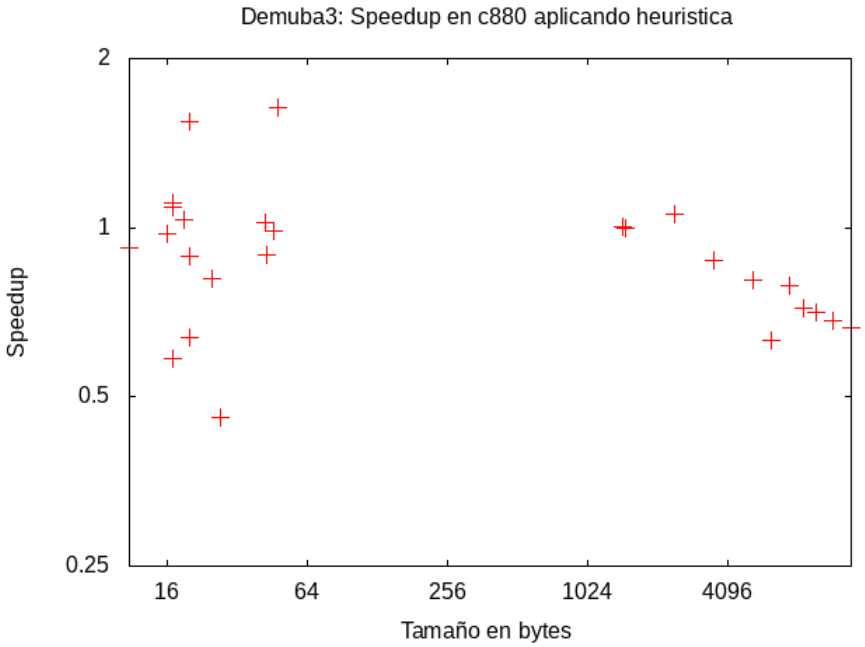
Prueba tautológica:

caso	bytes	tiempo	reglas
c880_391	11	1.13726	1
c880_390	17	0.979797	1
c880_389	17	0.707244	1
c880_388	17	1.53746	1
c880_423	19	1.39822	0.777778
c880_422	20	2.17614	1.28571
c880_421	20	0.802084	1
c880_420	20	1.31147	1
c880_447	22	1.34082	1.22222
c880_450	25	1.4584	1
c880_418	27	0.83986	1
c880_448	42	1.25272	1
c880_419	43	1.19514	1
c880_446	46	1.00382	1
c880_449	48	0.900777	1
c880_767	1440	1.73373	1.89368
c880_768	1472	1.77773	1.89368
promedio		1.27	1.12



Prueba con fórmulas originales:

caso	bytes	tiempo	reglas
c880_388	17	0.5844	
c880_423	19	1.031	
c880_422	20	1.547	
c880_421	20	0.8915	
c880_420	20	0.6377	
c880_450	25	0.8106	
c880_418	27	0.4596	
c880_448	42	1.022	
c880_419	43	0.897	
c880_446	46	0.9894	
c880_449	48	1.636	
c880_767	1440	1.006	
c880_768	1472	0.9966	
c880_850	2416	1.059	
c880_865	3556	0.8726	
c880_864	5250	0.8062	
c880_866	6259	0.631	
c880_863	7481	0.7904	
c880_874	8655	0.7178	
c880_880	9804	0.7063	
c880_879	11508	0.6822	
c880_878	13767	0.6642	
promedio		0.9018	



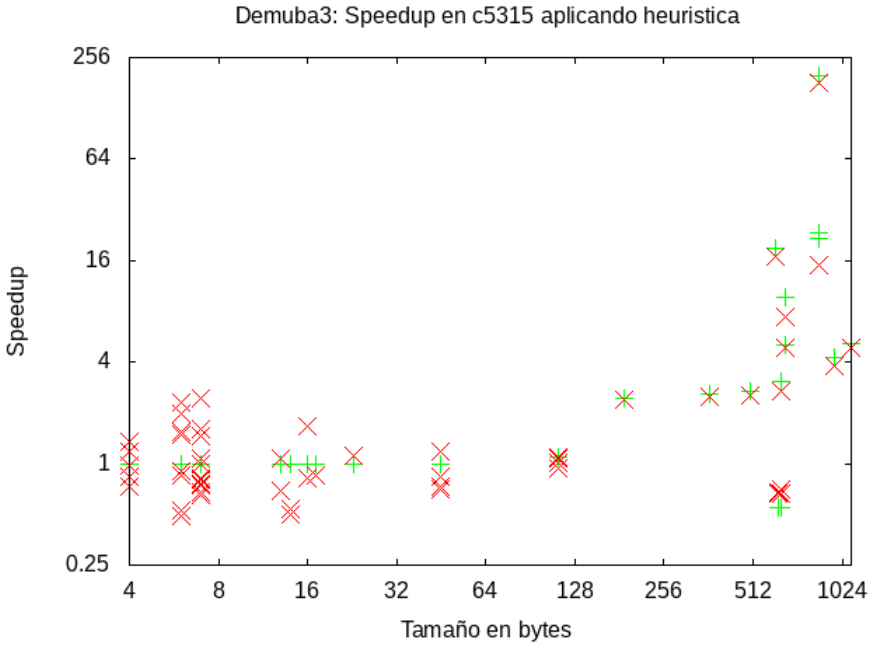
12.5.2 Set c5315

Prueba tautológica:

caso	bytes	tiempo	reglas
c5315_3360	4	0.965952	1
c5315_3359	4	0.828438	1
c5315_3358	4	1.3417	1
c5315_3357	4	0.733606	1
c5315_2309	4	1.16949	1
c5315_816	6	0.526044	1
c5315_709	6	1.49541	1
c5315_3604	6	2.32283	1
c5315_2527	6	0.85593	1
c5315_2387	6	0.895678	1
c5315_2142	6	1.55265	1
c5315_2139	6	0.484196	1
c5315_1066	6	1.96491	1
c5315_3613	7	0.799218	1
c5315_2584	7	1.0625	1
c5315_1155	7	0.745519	1

caso	bytes	tiempo	reglas
c5315_1154	7	0.682835	1
c5315_1153	7	0.756282	1
c5315_1152	7	1.60268	1
c5315_1145	7	0.645332	1
c5315_1144	7	0.821324	1
c5315_1143	7	0.998453	1
c5315_1142	7	0.681483	1
c5315_1141	7	0.65447	1
c5315_1139	7	1.46123	1
c5315_1138	7	2.429	1
c5315_1137	7	0.818183	1
c5315_1147	13	1.07249	1
c5315_1140	13	0.694446	1
c5315_2623	14	0.540983	1
c5315_2054	14	0.496696	1
c5315_2061	16	1.66813	1
c5315_2060	16	0.822964	1
c5315_1972	17	0.850731	1
c5315_2590	23	1.1126	1
c5315_4279	45	0.698811	1
c5315_4278	45	0.728921	1
c5315_4275	45	0.829251	1
c5315_4272	45	1.18316	1
c5315_4740	112	1.00982	1.08696
c5315_4739	112	1.09058	1.08696
c5315_4738	112	1.0683	1.08696
c5315_4737	112	0.932963	1.08696
c5315_7015	188	2.41102	2.42169
c5315_7365	366	2.48661	2.59954
c5315_7363	500	2.55753	2.68557
c5315_5240	608	16.8151	19.0573
c5315_7469	625	0.660723	0.550113
c5315_7449	625	0.680549	0.550113
c5315_7511	636	0.663573	0.55026
c5315_7506	636	0.704722	0.55026
c5315_7473	638	2.71159	3.06573
c5315_6646	661	4.81949	5.03333
c5315_6643	661	7.39676	9.71269
c5315_5388	854	180.287	199.783
c5315_6648	858	14.9205	23.444
c5315_6641	858	15.0579	21.7057
c5315_7467	960	3.80376	4.23013
c5315_7472	1098	4.8691	5.13913

caso	bytes	tiempo	reglas
promedio		91.35800	12.0614



Se observan grandes mejoras, recordando que speedup=2 significa que se reduce a la mitad el tiempo de computo necesario, o las reglas aplicadas para obtener una demostración, según el indicador que se esté utilizando para evaluar el speedup. También se notan casos extremos donde se obtiene speedup=16 y speedup=199.

Prueba con fórmulas originales:

caso	bytes	tiempo	reglas
c5315_3360	4	0.960301	
c5315_3358	4	0.787803	
c5315_2309	4	0.947948	
c5315_709	6	0.56458	
c5315_2527	6	1.32664	
c5315_2142	6	1.44508	
c5315_1066	6	1.24365	
c5315_2584	7	0.671992	
c5315_1154	7	0.718709	
c5315_1152	7	0.931708	
c5315_1144	7	0.979806	
c5315_1142	7	1.56438	
c5315_1139	7	1.25061	
c5315_1137	7	0.995382	
c5315_1140	13	0.935066	
c5315_2054	14	0.921614	
c5315_2060	16	1.36249	
c5315_2590	23	1.04827	
c5315_4278	45	0.862272	
c5315_4272	45	0.98864	
c5315_4739	112	0.98772	
c5315_4737	112	1.03877	
c5315_7365	366	1.08633	
c5315_5240	608	1.21234	
c5315_7449	625	1.10012	
c5315_7506	636	1.06888	
c5315_6646	661	1.10672	
c5315_5388	854	1.19112	
c5315_6641	858	1.19762	
c5315_7472	1098	0.916986	
promedio		1.0471	

Los resultados sobre el set de casos *c5315* tautológicos sobre los demostradores Demuba3 y demuba3_nh (no heurística) son muy buenos.

12.5.3 Set c2670

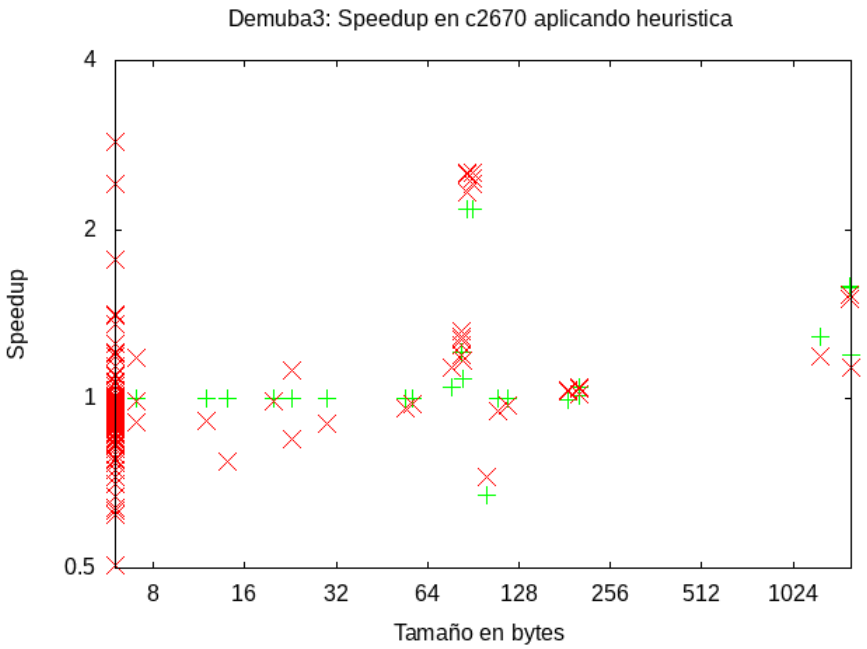
Prueba tautológica:

caso	bytes	tiempo	reglas
c2670_805	6	1.103	1
c2670_493	6	1.068	1

caso	bytes	tiempo	reglas
c2670_491	6	0.5066	1
c2670_490	6	1.025	1
c2670_489	6	1.028	1
c2670_487	6	0.7713	1
c2670_458	6	0.9153	1
c2670_457	6	1.404	1
c2670_456	6	1.207	1
c2670_420	6	0.8157	1
c2670_419	6	0.8431	1
c2670_401	6	0.9019	1
c2670_400	6	0.9031	1
c2670_398	6	1.099	1
c2670_218	6	0.8499	1
c2670_217	6	1.096	1
c2670_216	6	0.9542	1
c2670_215	6	0.9339	1
c2670_214	6	1.143	1
c2670_213	6	0.831	1
c2670_212	6	0.9983	1
c2670_211	6	1.415	1
c2670_210	6	0.7468	1
c2670_209	6	1.358	1
c2670_208	6	0.7256	1
c2670_207	6	0.7458	1
c2670_206	6	0.8895	1
c2670_205	6	0.9677	1
c2670_204	6	1.213	1
c2670_203	6	1.251	1
c2670_202	6	0.9033	1
c2670_201	6	0.8625	1
c2670_200	6	0.9387	1
c2670_199	6	1.19	1
c2670_198	6	0.8239	1
c2670_197	6	1	1
c2670_196	6	0.6329	1
c2670_195	6	1.058	1
c2670_194	6	0.7075	1
c2670_193	6	0.8129	1
c2670_192	6	0.933	1
c2670_191	6	0.9372	1
c2670_190	6	0.9309	1
c2670_189	6	1.008	1
c2670_188	6	0.7776	1

caso	bytes	tiempo	reglas
c2670_187	6	0.9108	1
c2670_186	6	0.9833	1
c2670_185	6	0.9899	1
c2670_184	6	0.8809	1
c2670_183	6	0.942	1
c2670_182	6	1.099	1
c2670_181	6	1.014	1
c2670_180	6	1.002	1
c2670_179	6	0.9423	1
c2670_178	6	1.07	1
c2670_177	6	0.945	1
c2670_176	6	0.9198	1
c2670_175	6	0.7972	1
c2670_174	6	0.6224	1
c2670_173	6	0.8929	1
c2670_172	6	0.8681	1
c2670_171	6	0.9367	1
c2670_170	6	0.915	1
c2670_169	6	2.864	1
c2670_168	6	1.772	1
c2670_167	6	0.8546	1
c2670_166	6	0.9786	1
c2670_165	6	1.002	1
c2670_164	6	0.8829	1
c2670_163	6	0.853	1
c2670_162	6	0.8858	1
c2670_161	6	0.9345	1
c2670_160	6	0.9197	1
c2670_159	6	0.7832	1
c2670_158	6	0.9378	1
c2670_157	6	0.9507	1
c2670_156	6	0.976	1
c2670_155	6	0.6406	1
c2670_154	6	0.8966	1
c2670_153	6	0.9618	1
c2670_152	6	1.071	1
c2670_151	6	1.038	1
c2670_150	6	0.8347	1
c2670_149	6	0.6686	1
c2670_148	6	0.9095	1
c2670_147	6	0.8733	1
c2670_146	6	1.037	1
c2670_145	6	2.409	1

caso	bytes	tiempo	reglas
c2670_144	6	0.9683	1
c2670_143	6	0.9054	1
c2670_494	7	0.9061	1
c2670_492	7	0.9877	1
c2670_488	7	1.182	1
c2670_1026	12	0.9147	1
c2670_1028	14	0.7721	1
c2670_799	20	0.9881	1
c2670_1269	23	0.8489	1
c2670_1029	23	1.127	1
c2670_792	30	0.9014	1
c2670_1277	54	0.9618	1
c2670_1448	57	0.9811	1
c2670_2018	77	1.134	1.048
c2670_2022	83	1.29	1.211
c2670_2020	83	1.243	1.211
c2670_2016	83	1.278	1.211
c2670_2014	83	1.321	1.211
c2670_2012	83	1.195	1.211
c2670_2010	83	1.248	1.211
c2670_1726	84	1.172	1.083
c2670_1821	86	2.331	2.182
c2670_1820	86	2.531	2.182
c2670_1819	86	2.506	2.182
c2670_1818	90	2.531	2.182
c2670_1817	90	2.417	2.182
c2670_1816	90	2.466	2.182
c2670_1969	100	0.7283	0.6743
c2670_1970	109	0.9518	1
c2670_1971	118	0.9738	1
c2670_2390	186	1.035	0.9979
c2670_2389	186	1.038	0.9979
c2670_2388	186	1.032	0.9979
c2670_2387	186	1.038	0.9979
c2670_2644	202	1.042	1.015
c2670_2643	202	1.018	1.015
c2670_2496	202	1.052	1.051
c2670_2970	1257	1.189	1.293
c2670_2971	1569	1.503	1.583
c2670_2925	1577	1.531	1.58
c2670_2891	1589	1.137	1.198
promedio		1.0796	1.0766



Prueba con fórmulas originales:

caso	bytes	tiempo	reglas
c2670_805	6	0.987524	
c2670_491	6	0.894731	
c2670_489	6	0.857077	
c2670_458	6	1.29449	
c2670_456	6	1.31786	
c2670_419	6	1.0924	
c2670_400	6	1.21089	
c2670_218	6	0.954076	
c2670_216	6	1.47271	
c2670_214	6	0.892935	
c2670_212	6	0.92964	
c2670_210	6	1.15346	
c2670_208	6	0.678893	
c2670_206	6	1.53656	
c2670_204	6	0.606271	
c2670_202	6	1.14692	
c2670_200	6	1.04174	
c2670_198	6	0.657041	
c2670_196	6	1.29253	

caso	bytes	tiempo	reglas
c2670_194	6	1.29941	
c2670_192	6	1.05049	
c2670_190	6	1.10554	
c2670_188	6	1.1225	
c2670_186	6	0.888707	
c2670_184	6	0.937644	
c2670_182	6	0.921629	
c2670_180	6	0.561871	
c2670_178	6	0.920725	
c2670_176	6	0.800172	
c2670_174	6	1.39065	
c2670_172	6	0.877354	
c2670_170	6	0.874355	
c2670_168	6	1.12605	
c2670_166	6	0.794197	
c2670_164	6	1.63181	
c2670_162	6	1.1463	
c2670_160	6	1.14871	
c2670_158	6	1.09617	
c2670_156	6	1.05407	
c2670_154	6	1.23885	
c2670_152	6	1.06349	
c2670_150	6	0.844191	
c2670_148	6	1.03353	
c2670_146	6	0.975382	
c2670_144	6	0.973934	
c2670_494	7	1.32422	
c2670_488	7	0.889344	
c2670_1028	14	1.11331	
c2670_1269	23	1.0236	
c2670_792	30	1.13696	
c2670_1448	57	0.85969	
c2670_2022	83	0.973427	
c2670_2016	83	0.980838	
c2670_2012	83	0.975675	
c2670_1726	84	1.06072	
c2670_1820	86	0.987309	
c2670_1818	90	1.34009	
c2670_1816	90	1.21019	
c2670_1970	109	0.828893	
c2670_2390	186	0.954936	
c2670_2388	186	0.954633	
c2670_2644	202	0.992242	

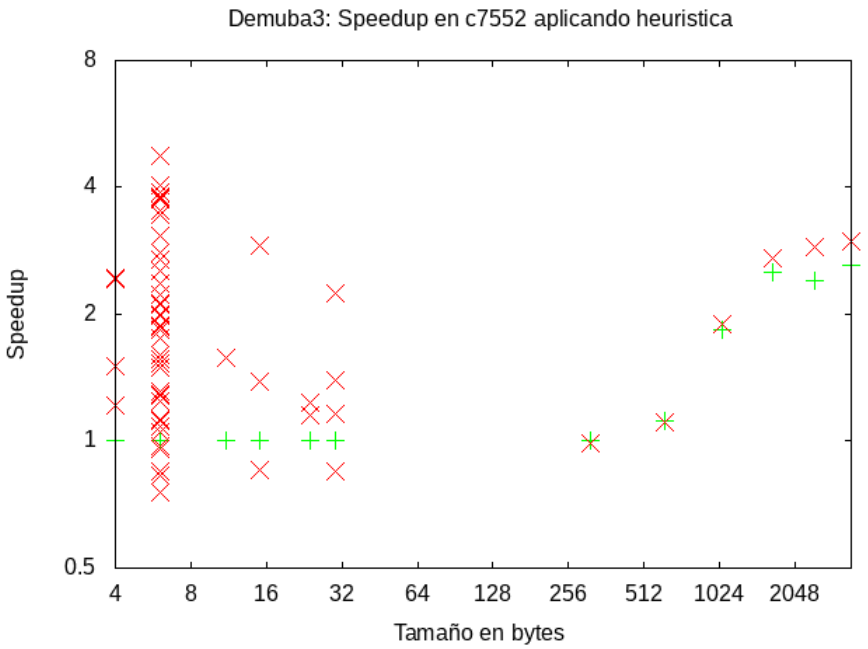
caso	bytes	tiempo	reglas
c2670_2496	202	1.01192	
c2670_2971	1569	1.01334	
c2670_2891	1589	1.25079	
promedio		1.04273	

12.5.4 Set c7552

Prueba tautológica:

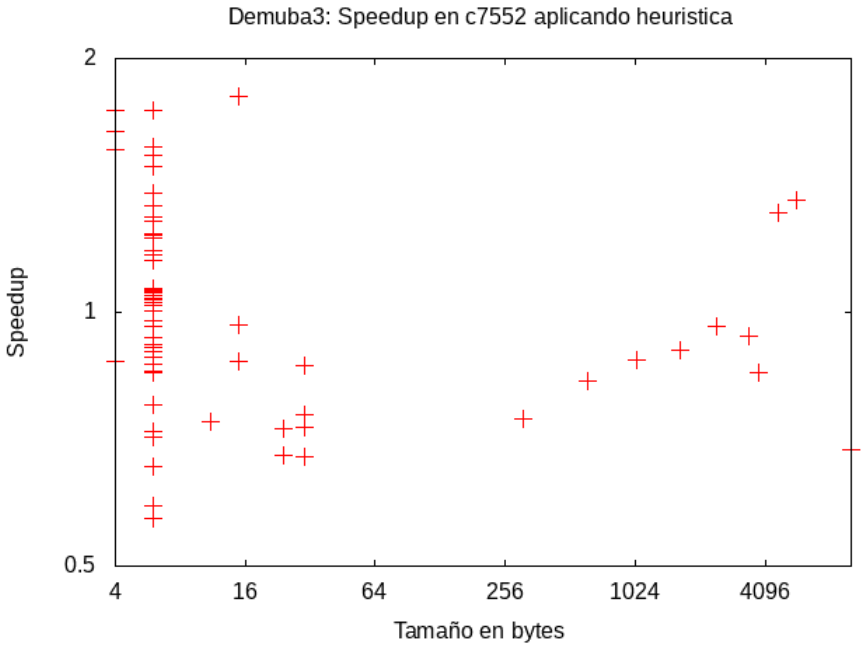
caso	bytes	tiempo	reglas
c7552_889	4	1.51199	1
c7552_388	4	2.44902	1
c7552_387	4	2.42754	1
c7552_1490	4	1.21868	1
c7552_945	6	0.828586	1
c7552_813	6	2.0016	1
c7552_707	6	3.81903	1
c7552_643	6	1.12923	1
c7552_582	6	1.12081	1
c7552_573	6	1.49974	1
c7552_571	6	3.75341	1
c7552_569	6	1.31168	1
c7552_567	6	3.43281	1
c7552_565	6	1.6033	1
c7552_563	6	4.73588	1
c7552_561	6	3.06827	1
c7552_559	6	0.95561	1
c7552_556	6	2.21975	1
c7552_553	6	3.53345	1
c7552_551	6	1.87364	1
c7552_549	6	0.972541	1
c7552_547	6	2.37039	1
c7552_545	6	2.54082	1
c7552_543	6	1.88017	1
c7552_541	6	0.753855	1
c7552_539	6	1.83528	1
c7552_537	6	1.525	1
c7552_535	6	2.69313	1
c7552_519	6	1.08096	1
c7552_517	6	1.56417	1
c7552_515	6	1.76154	1

caso	bytes	tiempo	reglas
c7552_513	6	2.10371	1
c7552_511	6	1.08006	1
c7552_509	6	0.849033	1
c7552_507	6	2.12309	1
c7552_505	6	3.77124	1
c7552_501	6	1.29196	1
c7552_492	6	0.957074	1
c7552_489	6	2.79884	1
c7552_486	6	1.24119	1
c7552_484	6	3.89757	1
c7552_482	6	1.28152	1
c7552_478	6	3.75799	1
c7552_241	6	1.99212	1
c7552_1114	6	4.04791	1
c7552_1111	6	1.05349	1
c7552_1781	11	1.58249	1
c7552_881	15	1.38869	1
c7552_1112	15	2.90825	1
c7552_1110	15	0.856643	1
c7552_1489	24	1.2334	1
c7552_1113	24	1.15205	1
c7552_885	30	2.23906	1
c7552_884	30	0.847516	1
c7552_883	30	1.39211	1
c7552_882	30	1.16049	1
c7552_10025	312	0.985807	1
c7552_10112	616	1.10585	1.12121
c7552_10111	1048	1.89517	1.83526
c7552_10110	1658	2.71673	2.51749
c7552_10109	2450	2.88405	2.39766
c7552_10353	3429	2.98884	2.62412
promedio		1.98477	1.08864



Prueba con fórmulas originales:

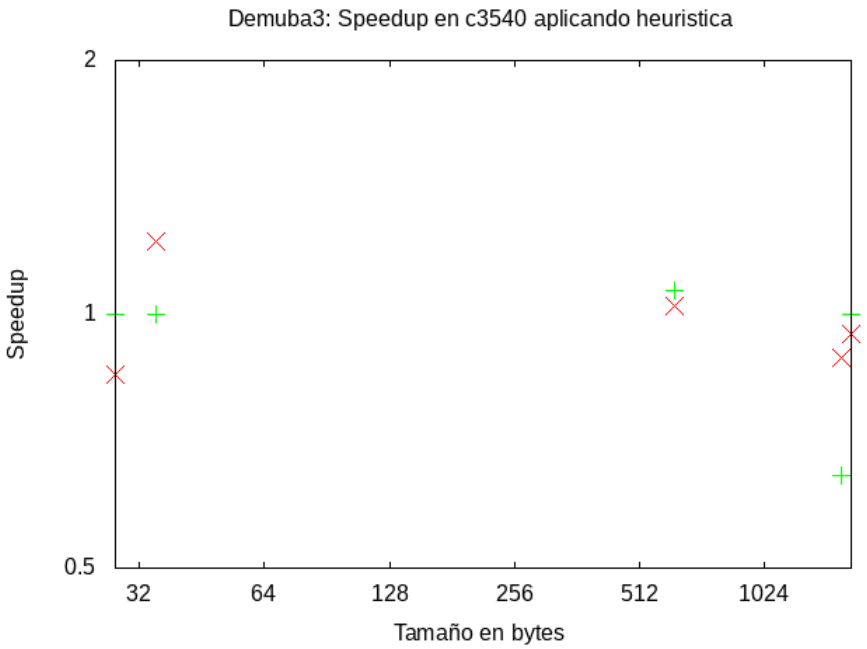
caso	bytes	tiempo	reglas
c7552_889	4	0.8754	
c7552_388	4	1.737	
c7552_387	4	1.559	
c7552_1490	4	1.639	
c7552_945	6	0.5897	
c7552_813	6	1.572	
c7552_707	6	0.961	
c7552_643	6	1.239	
c7552_582	6	0.9083	
c7552_573	6	1.019	
c7552_571	6	1.491	
c7552_569	6	1.226	
c7552_567	6	1.537	
c7552_565	6	0.8993	
c7552_563	6	1.062	
c7552_561	6	1.734	
c7552_559	6	1.285	
c7552_556	6	1.026	
c7552_553	6	0.8471	
c7552_551	6	1.384	
c7552_549	6	1.049	
c7552_547	6	1.295	
c7552_545	6	0.7224	
c7552_543	6	0.6559	
c7552_541	6	0.8526	
c7552_539	6	1.183	
c7552_537	6	1.151	
c7552_535	6	0.5703	
c7552_519	6	1.049	
c7552_517	6	1.056	
c7552_515	6	1.172	
c7552_513	6	1.002	
c7552_511	6	0.8673	
c7552_509	6	1.038	
c7552_507	6	1.068	
c7552_505	6	1.061	
c7552_501	6	0.9771	
c7552_492	6	1.234	
c7552_489	6	0.886	
c7552_486	6	0.9154	
c7552_484	6	1.339	
c7552_482	6	0.9323	
c7552_478	6	0.7753	
c7552_241	6	0.7114	
c7552_1114	6	1.037	
c7552_1111	6	1.227	
c7552_1781	11	0.741	
c7552_881	15	1.804	
c7552_1112	15	0.9678	
c7552_1110	15	0.8733	
c7552_1489	24	0.6759	
c7552_1113	24	0.7291	
c7552_885	30	0.7562	



12.5.5 Set c3540

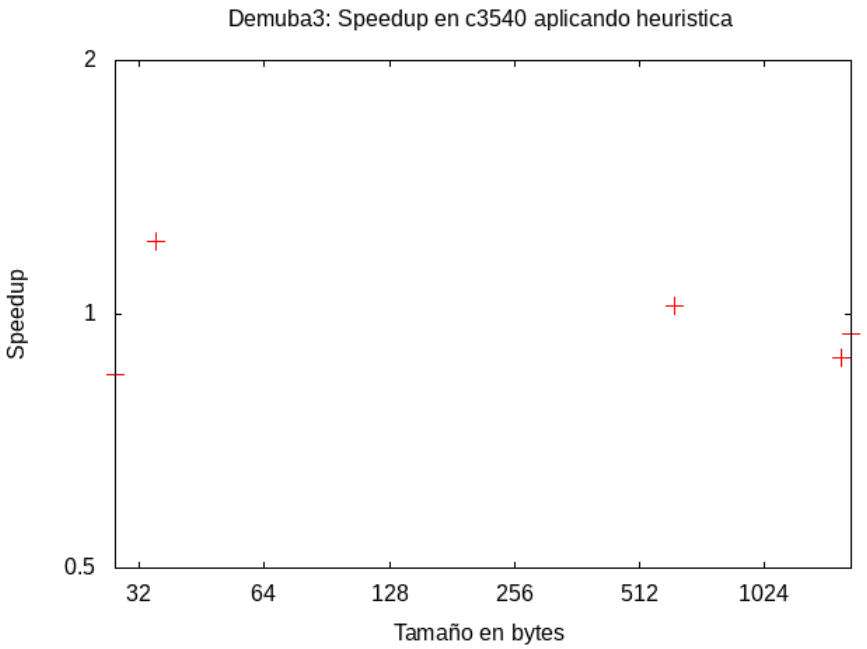
Prueba tautológica:

caso	bytes	tiempo	reglas
c3540_1947	28	0.848468	1
c3540_1713	35	1.22085	1
c3540_3195	622	1.02517	1.0686
c3540_3987	1562	0.887302	0.642567
c3540_3833	1652	0.948963	1
promedio		0.986	



Prueba con fórmulas originales:

caso	bytes	tiempo	reglas
c3540_1947	28	0.9651	
c3540_1713	35	0.7284	
c3540_3195	622	0.8816	
c3540_3987	1562	0.9778	
c3540_3833	1652	0.9685	
c3540_4028	17387	0.9338	
promedio		0.909	



En este set de prueba la heurística no mejoró el desempeño, si bien en la mayoría de los casos hay mejoras significativas.

13 Código fuente y compilación

Para compilar los predicados externos y el tester se debe instalar Swi-prolog, ya que el software contiene la utilidad swipl-ld que facilitar la compilación y linkeo de predicados externos, de otra manera deberían resolverse los simbolos especificando manualmente los paths, haciendo la tarea mas difícil.

El generador de fórmulas y el calculador de speedup estan programados en C++ standard, por lo tanto no requieren dependencias adicionales.

13.0.1 Linux (Debian/Ubuntu)

Para instalar Swi-prolog en Debian/ubuntu se deben seguir las siguientes instrucciones (tomadas de https://www.swi-prolog.org/build/PPA.html):

1. sudo apt-add-repository ppa:swi-prolog/stable
2. sudo apt-get update
3. sudo apt-get install swi-prolog

También se debe tener instalado el compilador GCC, de lo contrario se puede instalar mediante:

- sudo apt install build-essential

Luego se debe ejecutar el script */program/compile.sh*, la librería compilada se ubicará en */program/index.so*. El script tambien compila el generador de formulas y el tester.

13.0.2 Windows

Los requerimientos en Windows son los mismos, Swi-prolog y compilador GCC. Para instalar GCC utilice Mingw-w64, un port de GCC para Windows, el mismo se puede descargar de http://mingw-w64.org.

Es recomendable no modificar la variable de entorno PATH en las instalaciones, y usar un script para especificar los paths antes de ejecutar la compilación. Para hacerlo modifique el archivo *win_ env.bat*, y especifique los paths de instalación de Swi-Prolog y Mingw-w64. Luego ejecute *compile.bat*.

14 Uso

14.1 Generación de formulas

Para la generación de fórmulas se utilizan los archivos *ISCAS85*, ubicados en */program/iscas85*. Los binarios utilizados son *espacios* y *formula_ generator*. *Espacios* elimina los espacios en blanco de los archivos de definición de circuitos, para simplificar el parseo que realiza *formula_ generator*. El script f*ormula_ generator.sh/formula_ generator.bat* es provisto para facilitar la invocacion de los ejecutables. Su uso es "*formula_ generator.sh [nombre_ prueba]*", donde *[nombre_ prueba]* es el nombre de prueba de un archivo iscas, ubicado en */program/iscas85/[nombre_ prueba].bench.* Las fórmulas generadas se ubican en */program/tests/[nombre_ prueba],* por ejemplo:

./formula_ generator.sh c2670

14.2 Demostrador

Para ejecutar el demostrador demuba2 es necesario ejecutar swi-prolog con el parametro consult para cargar el programa del demostrador, y establecer límites de memoria y pila necesarios para la ejecución de grandes casos de prueba. Para facilitar esta tarea, se puede utilizar el script *demuba2.bat/demuba2.sh*, el cual contiene la línea:

swipl --stack-limit=32g -g 'consult('demuba2').'

Para ejecutar el demostrador demuba3, es necesario tambien cargar la librería con los predicados externos, por lo tanto se invoca:

swipl --stack-limit=32g -g ' load_foreign_library('index'), consult('demuba3').
'

Una vez cargado el demostrador, ya se puede consultar directamente, por ejemplo:

/- a v -a:L.

/- -(a & b) v (a & b):L.

donde L es la cantidad de reglas aplicadas para obtener la demostración, si es que existe (si prolog contesta true).

14.3 Tester

La interfaz del tester se especificó anteriormente. A continuación se muestran dos invocaciones de ejemplo. Ambas ejecutan el mismo conjunto de pruebas, con diferentes demostradores: demuba3 sin usar heurística, y demuba3 usando heurística.

./tester --dem=demuba3_nh --indexlib=index --set=c2670 --type=demostrart --limit=130 --basefile=dem3_nh.txt

./tester --dem=demuba3 --indexlib=index --set=c2670 --type=demostrart --limit=130 --basefile=dem3.txt

14.4 Calculador de Speedup

A continuación se utilizan los resultados del tester para calcular el speedup.

./calc_speedup -a dem3_nh.txt -b dem3.txt -c speedup.txt

15 Conclusiones

Si siguio la sugerencia del Ricardo Rodriguez y se pudo desarrollar una heurística que efectivamente acelere el proceso de demostración, tanto en tiempo de cómputo como en cantidad de pasos. Se alcanzaron muy buenos resultados, comprobados en los experimentos, principalmente en pruebas tautológicas y con tamaños de fórmulas grandes.

Se adquirieron nuevos conocimientos en los tópicos de demostración automatica y SAT. Tambien se desarrolló una herramienta complementaria, el generador de casos de prueba, que es util para realizar experimentos con cualquier demostrador en forma no-clausal o SAT-solver no-clausal.

16 Futuros trabajos

A continuación se especifican las ideas mas importantes que exceden el ámbito del presente trabajo, tanto por ser suceptibles a abordarse con métodos diferentes SNR, o bien proponiendo mejoras al mismo.

16.1 Deducción unitaria

Usando el concepto de deducción unitaria se puede desarrolar mejoras al método. La deducción unitaria ocurre cuando todos los literales salvo uno de una cláusula son fasificados, entonces el restante debe ser verdadero.

La fórmula $F = \neg a \vee \neg b \vee c$ en forma clausal es $F_c = (\neg a, \neg b, c)$.

Dada una valuación v para F,

si $v(a = t, b = t) \Rightarrow c = t$

Este concepto se puede aplicar con resolucion no-clausal, cuando obtengo una fórmula sin disyunciones.

Por ejemplo:

$C = \neg a \vee \neg b \vee (c \wedge d)$

si $v(a = t, b = t) \Rightarrow (c \wedge d) = t \equiv (c = t) \wedge (d = t)$

Son las deducciones que son válidas para cualquier rama (son "factor comun"), o solo tengo una sola rama (una sola valuación posible que satisface la fórmula).

$C = \neg a \vee \neg b \vee (c \wedge d \wedge (e \vee f)$

dada una valuación v

si $v = (a = t, b = t)$,entonces se reduce a $F = e \vee f$

Futuros trabajos:

Entonces, si registro una fórmula y el conjunto de átomos preasignados que me generan una deducción de un átomo para esta fórmula (también puedo registrar contradicciónes), puedo utilizar esa información para guiar la búsqueda y también acelerar cálculos. Este planteo se asemeja al aprendizaje realizado en el método de General matings, los diferentes perfiles indican cuando y que registrar.

Descubrimiento de literales unitarios desde hpgraph

Los SAT solvers modernos que operan sobre una representacion en *CNF* emplean la regla de literal unitario para una propagación de restricciones booleana. La regla de literal unitario determina que si todos, salvo un literal de una

cláusula son falsos, entonces el literal sin valor de verdad debe ser verdadero. En nuestro contexto la fórmula de entrada no esta necesariamente representada en *CNF*, sin embargo es posible obtener la regla de literal unitario por el uso del hpgraph de la fórmula de entrada.

Este hecho en *Secuentes* es equivalente a obtener una fórmula atómica a partir de una hipótesis y un conjunto de átomos preasignados.

$H1, A_1, A_2, ..., A_n \Rightarrow B$

donde H_1es una hipótesis, $A_1, A_2, ..., A_n$una asignación de átomos y B un átomo deducido.

En [8] la deducción unitaria se comprueba valuando los nodos del hpgraph y realizando una búsqueda de camino minimo desde cualquier nodo raiz hacia cada nodo hoja. Así es posible comprobar tanto si hay deducción unitaria o contradicción. A continuación una cita de [8] en donde se menciona dos optimizaciones necesarias para la aplicabilidad del algoritmo.

> Thus, we use two optimizations which are crucial for efficiency: 1) incremental shortest path computation: whenever a variable is (un)assigned, instead of computing the shortest path estimate for every node in the hpgraph, we only examine the nodes whose shortest path estimate can get affected due to this assignment. 2) by limiting the range of δ(n) to only {0,1,∞}.

Teniendo una implementación incremental del algoritmo, y almacenando estados de la búsqueda para reutilizarlos al volver a un estado anterior, puede ser posible forma eficiente de este esquema de optimización para cada hipótesis del secuente.

References

[1] R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In Proceedings of the 17th National Conference on Artificial Intelligence, pages 262-266. AAAI Press, 2000.

[2] P. Blackburn, M. de Rijke and Y. Venema. Modal Logic, Cambridge University Press, 2002.

[3] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Proceedings of the 14th National Conference on Artificial Intelligence, pages 203-208. AAAI Press, 1997.

[4] José M. Castaño. A Parsing Approach to SAT. IBERAMIA 2014: 3-14.

[5] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In Proceedings of the 12th National Conference on Artificial Intelligence, pages 1092-1097, 1994.

[6] G.Gentzen. The collected Papers of Gerhard Gentzen. North-Ho1land, 1969.

[7] H.Gao, Y Goto, and J. Cheng. Automatic Theorem Finding by Forward Reasoning based on Strong Relevant Logic: A Case Study in Graph Theory. In Advanced Multimedia and Ubiquitous Engineering: Future Information Technology edited by James J. Jong Hyuk Park, Han-Chieh Chao, Hamid Arabnia. Lecture Note in Electrical Engineering 352, 23-30. Springer. 2015

[8] H. Jain, C. Bartzis and E. Clarke. Satisfiability Checking of Non-clausal fórmulas Using General Matings. LNCS 4121. 2006.

[9] H. Kautz, D. McAllester and B. Selman. Encoding plans in propositional logic. In Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, 1996.

[10] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Proceedings of the 14th National Conference on Artificial Intelligence, pages 1194-1201. AAAI Press, 1996.

[11] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In Proceedings of IJCAI'99, pages 318-325, 1999.

[12] A.Kvitca. Demostrador de teoremas= reglas de inferencia + estructura de datos + control. XVIII JAIIO, Argentina. 1989.

[13] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In Proceedings of IJCAI'97, pages 366-371. 1997.

[14] C. M. Li. Integrating equivalence reasoning into Davis-Putnam procedure. In Proceedings of the 17th National Conference on Artificial Intelligence, pages 291-296. AAAI Press, 2000.

[15] Z.Manna. A deductive approach to program synthesis. R.Waldinger.Prog.Languages and Systems ACM 2, 1982.

[16] F. Massacci, F. and L. Marraro. lógical cryptanalysis as a SAT-problem. Journal of Automated Reasoning, 24(1/2):165-203, 2000.

[17] N.Murray. Completely non-clausal theorem proving. Artificial Intelligence 18,1 1982.

[18] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In Proceedings of the 12th National Conference on Artificial Intelligence, pages 337-343. AAAI Press, 1997.

[19] B. Selman, H. Kautz and D. McAllester. Ten challenges in propositional reasoning and search. In Proceedings of IJCAI'97, pages 50-54, 1997.

[20] R.M.Smullyan. First-order logic. Springer 1968.

[21] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In Proceedings of the 17th National Conference on Artificial Intelligence, pages 297-302. AAAI Press, 2000.

[22] M.E. Stickel. A non-clausal connection-graph resolution theorem-proving program. AAAI-82, 1982.

[23] Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving Non-clausal fórmulas with DPLL Search. In SAT, 2004.

[24] Hashim Habiballa, Non-clausal resolution theorem prover. In Empirically Successful Automated Reasoning Workshop (ESCAR'2005) (CADE), Tallinn, Estonia