Proyecto 2 Inteligencia Artificial

Saldaña Hernandez Norman Brandon Guerrero Ferrusca Francisco

González Lorenzo Daniela Michel Tenorio Veloz Alisson Dafne

Octubre 2021

1 Planteamiento del Problema

Implementar una solución en un lenguaje de programación lógica para el problema de satisfacibilidad.

2 Solución basada en lógica proposicional

Para solucionar el problema decidimos utilizar los siguientes operadores y relacionarlos con cada una de las operaciones:

- -: Negación
- : Conjunción
- v: Disyunción
- =>: Implicación
- <=>: Doble implicación

2.1 LOGICA PROPOSICIONAL

La lógica proposicional es una parte de la lógica clásica que estudia las variables proposicionales, sus posibles implicaciones, los valores de las proposiciones o de conjuntos de ellas formadas a partir de los conectores lógicos, a través de tablas de verdad que nos indican lo verdadero o falso.

2.1.1 Proposiciones

El Diccionario RAE define proposición como "enunciación de una verdad demostrada o que se trata de demostrar" la cual utiliza un lenguaje exacto que no da lugar a imprecisiones.
Existen dostipos de proposiciones:

- Proposiciones simples:
  Para que un enunciado que hacemos sea una proposición el único requisito es que podamos definirla como verdadera o falsa. A estas afirmaciones verdaderas o falsas las llamamos proposiciones simples y, para trabajar con ellas, las representamos con letras del alfabeto.
- Proposiciones complejas:
  Una proposición compleja es la unión de dos o más proposiciones simples que están unidas por un conector lógico.

2.1.2 Conectores lógicos

Los conectores lógicos sirven para enlazar dos o más proposiciones simples. Los cuales se muestran a continuación:

- Conjunción:
  Equivalente a la letra "y". En este caso, para que la proposición compuesta sea verdad las dos proposiciones simples deben ser verdaderas.
- Disyunción débil:
  Equivalente a la letra "o", para que la proposición compuesta sea verdad alguna de las dos proposiciones será verdadera o las dos. La proposición compuesta sólo será falsa si las dos proposiciones simples que la componen son falsas.

- Disyunción fuerte:
  Equivalente a la frase "o ... , o ...". En este caso para que la proposición compuesta sea verdadera una de las dos simples que la componen debe ser verdadera y la otra falsa, si las dos son verdaderas o falsas, la proposición compuesta es falsa.

- Condicional:
  Equivalente a la expresión "sí ... entonces ...", la primera proposición simple es el antecedente y la segunda el consecuente. En este caso la proposición siempre será verdadera, a excepción de cuando la primera es verdadera y la segunda falsa.

- Bicondicional:
  por la expresión "sí y sólo sí", como en la frase "el animal ladra sí y sólo sí es un perro": en este caso la proposición compuesta es verdadera si las dos simples son, a la vez, verdaderas o las dos falsas, si una fuera verdadera y la otra falsa la compuesta sería falsa.

- Negación:
  Equivalente a la expresión "no" o "no es cierto que". Para que la compuesta sea verdadera, al menos una de las dos que la componen ha de ser falsa.

2.1.3 Tablas de verdad

Para saber si una proposición compleja es verdadera o falsa necesitamos saber si las proposiciones simples que la componen son verdaderas o falsas.
Dependiendo del resultado final según la combinación de estas proposiciones simples, la tabla de verdad de la compleja puede ser de tres tipos:

- Tautológica:
  Cuando cualquier combinación de verdadero o falso de sus componentes da siempre como resultado que la proposición compleja es verdadera.

- Contradictoria:
  Si cualquier combinación de verdadero o falso de los componentes da siempre como resultado que la proposición compleja es falsa.

- Contingente:
  Cuando existen distintas posibilidades de resultados según la combinación de verdadero y falso de los componentes.

2.2 SATISFACIBILIDAD

Una proposición es satisfacible si la última columna de su tabla de verdad contiene al menos un valor verdadero.En otras palabras se dice que si una estructura M satisface (da verdadero al aplicarla a la formula) a una formula ϕ entonces la formula es satisfacible

2.3 Desarrollo del código

Para resolver el problema declaramos los operadores:

```
:- op(1, fy, -). % negaci n
:- op(2, xfy, &). % conjunci n
:- op(2, xfy, v). % disyunci n
:- op(3, xfy, =>). % implicaci n
:- op(3, xfy, <=>). % Doble Implicaci n
```

Si evaluamos la fórmula en M, el resultado 1, se dice que M es modelo de la fórmula

```
es_modelo(ESTRUCTURA,FORMULA) :-valor(FORMULA,ESTRUCTURA,V),nl,write(ESTRUCTURA),write("  "),write(V),V
                                                    = 1.
```

Declaramos eval, que contiene las definciones de disyunción, conjunción, implicación, doble implicación y negación

```
eval(v, 0, 0, 0) :- !.%Definici n de disyuncion (or)
eval(v, _, _, 1).
eval(&, 1, 1, 1) :- !.%Definici n de conjuncion (and)
eval(&, _, _, 0).
eval(=>, 1, 0, 0) :- !.%definici n de implicaci n (=>)
eval(=>, _, _, 1).
eval(<=>, X, X, 1) :- !.%definici n de doble implicaci n (<=>)
eval(<=>, _, _, 0).
eval(-, 1, 0). %definici n de negaci n (-)
eval(-, 0, 1). % cual es esta?
```

Definimos las reglas al obtenemos las estructuras de M para el cojunto de variables proposicionales; Al ingresar un cojunto de varibales [p,q], devuelve las posibles estructuras: [(p,0),(q,0)] , [(p,0),(q,1)], [(p,1),(q,0)], [(p,1),(q,1)]

```
getEstructuras([],[]).
getEstructuras([VARIABLE|VARIABLES_COLA],[(VARIABLE,VALOR)|ML]) :-
valor_verdad(VALOR),
getEstructuras(VARIABLES_COLA,ML).
```

Obtención de las estructuras.

```
Los valores que pueden tomar las variables son 0 y 1, si ponemos valor_verdad(X), puede regresar 0 y 1
valor_verdad(0).
valor_verdad(1).
```

Para cumplir la condición de satisfacción de una formula, se necesita que tenga al menos 1 modelo (Un modelo es una estructura que al aplicar a la formula hace que de verdadero) Por otro lado, se revisará estructura por estructura (00,01,10,11) (si es que fueran dos variables).

```
es_satisfacible(FORMULA,VARIABLES) :-
getEstructuras(VARIABLES,ESTRUCTURA), %obtiene las posibles estructuras
es_modelo(ESTRUCTURA,FORMULA).% evalua las estructuras para comprobar si alguna es modelo, si lo es
                                                    entonces la formula es satisfacible
```

Código Final

```
:- op(1, fy, -). % negacion
:- op(2, xfy, &). % conjuncion
:- op(2, xfy, v). % disyuncion
:- op(3, xfy, =>). % implicacion
:- op(3, xfy, <=>). % Doble Implicacion

%si al evaluar la formula en M el resultado es 1, se dice que M es modelo de la formula
es_modelo(ESTRUCTURA,FORMULA) :-valor(FORMULA,ESTRUCTURA,V),nl,write(ESTRUCTURA),write("  "),write(V),V
                                                    = 1.

% descompone la formula y evalua. p.e (p => q) v (q => p) se descompone en A = (p =>q) v B = (q => p).
% %Luego descompone A = (p) => B= (q). Con memberchk obtiene el valor de p y de q, finalmente
%evalua de adentro hacia afuera (por orden de operadores) las definiciones de las funciones, en este
                                                    caso v y => (definidas en eval)
valor(F, ESTRUCTURA, V) :-memberchk((F,V), ESTRUCTURA).
valor(-A, ESTRUCTURA, V) :-valor(A, ESTRUCTURA, VA),eval(-, VA, V).
valor(F, ESTRUCTURA, V) :-F =..[Op,A,B], valor(A, ESTRUCTURA, VA),valor(B, ESTRUCTURA, VB),
eval(Op, VA, VB, V).

%eval contiene las definiciones de
eval(v, 0, 0, 0) :- !.%Definici n de disyuncion (or)
eval(v, _, _, 1).
eval(&, 1, 1, 1) :- !.%Definici n de conjuncion (and)
eval(&, _, _, 0).
eval(=>, 1, 0, 0) :- !.%definici n de implicaci n (=>)
eval(=>, _, _, 1).
eval(<=>, X, X, 1) :- !.%definici n de doble implicaci n (<=>)
eval(<=>, _, _, 0).
eval(-, 1, 0). %definici n de negaci n (-)
eval(-, 0, 1).

%obtenci n de las estructuras M. p.e para el conjunto de variables proposicionales
% ingresas un conjunto de variables p.e [p,q], devuelve las posibles estructuras:
% [(p,0),(q,0)] , [(p,0),(q,1)], [(p,1),(q,0)], [(p,1),(q,1)]
getEstructuras([],[]).
getEstructuras([VARIABLE|VARIABLES_COLA],[(VARIABLE,VALOR)|ML]) :-
valor_verdad(VALOR),
getEstructuras(VARIABLES_COLA,ML).

%Los valores que pueden tomar las variables son 0 y 1, si ingresamos valores en valor_verdad(X),
                                                    obtendremos 0 y 1
valor_verdad(0).
valor_verdad(1).
%La condici n de satisfacci n de una formula es que tenga al menos 1 modelo. Un modelo es una
                                                    estructura que al aplicar a la formula hace que de
                                                    verdadero. Para saber si una formula es
                                                    satisfacible es conveniente revisar estructura por
                                                    estructura (00,01,10,11; para dos variables por
                                                    ejemplo) hasta que una de ellas sea un modelo es
                                                    decir
es_satisfacible(FORMULA,VARIABLES) :-
getEstructuras(VARIABLES,ESTRUCTURA), %obtiene las posibles estructuras
es_modelo(ESTRUCTURA,FORMULA).% evalua las estructuras para comprobar si alguna es modelo, si lo es,
                                                    entonces la formula es satisfacible
```

3 Experimentos

3.1 Experimento de baja dificultad

3.1.1 Experimento de 1 variable

-p & q

p	-p	p & -p
0	1	0
1	0	0

No es satisfacible (insatisfacible).

3.1.2 Experimentos de 2 variables

-p v q

p	-p	q	-p v q
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1

Si es satisfacible.

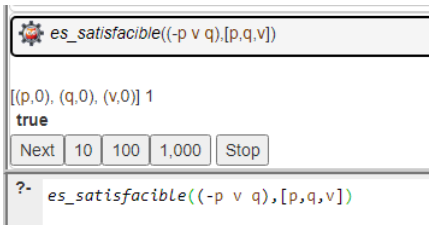

Figure 1: Resultado en Prolog

-(p & q) & (p v q)

p	q	p & q	-(p & q)	(p v q)	-(p & q) & (p v q)
0	0	1	0	1	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	0	1	0	0

Si es satisfacible.

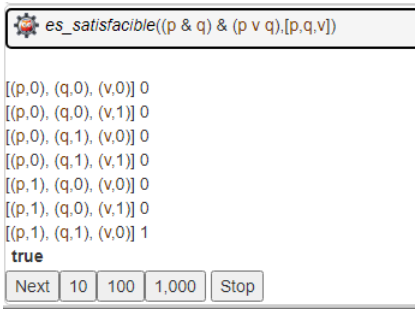

Figure 2: Resultado en Prolog

3.2 Experimentos de media dificultad

3.2.1 Experimentos de 3 variables

(p v q) & (-q v r)

p	q	-q	r	p v q	-q v r	(p v q) & (-q v r)
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	0	1	1	1	1

Si es satisfacible.

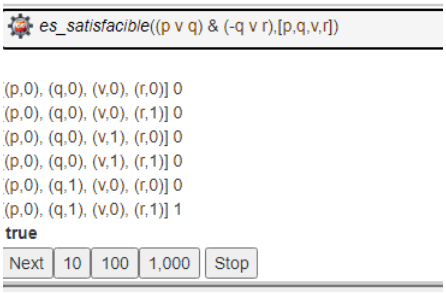

Figure 3: Resultado en Prolog

(r & q) =¿ p

r	q	p	r & q	(r & q) => p
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Si es satisfacible.

-(p =¿ q) ¡=¿ (p & r)

p	q	r	(p => q)	-(p => q)	(p & r)	-(p => q) <=> (p & r)
0	0	0	1	0	0	1
0	0	1	1	0	0	1
0	1	0	1	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Si es satisfacible.

3.3 Experimentos de alta dificultad

3.3.1 Experimentos de 4 variables

((p & -q) & r) => s

p	q	r	s	-q	(p & -q)	((p & -q) & r)	((p & -q) & r) => s
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	1
0	0	1	0	1	0	0	1
0	0	1	1	1	0	0	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	0	0	0	1
0	1	1	1	0	0	0	1
1	0	0	0	1	1	0	1
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	1
1	1	0	1	0	0	0	1
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	1

Si es satisfacible.

(p & q) ¡=¿ (r & s)

p	q	r	s	(p & q)	(r & s)	(p & q) <=> (r & s)
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	1	1

Si es satisfacible.

3.3.2 Experimento de 5 variables

(p =>q) & (r => (t => s))

p	q	r	s	t	p =>q	t => s	r => (t => s)	(p =>q) & (r => (t => s))
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	0	1	1
0	0	0	1	0	1	1	1	1
0	0	0	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1
0	0	1	0	1	1	0	0	0
0	0	1	1	0	1	1	1	1
0	0	1	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1
0	1	0	0	1	1	0	1	1
0	1	0	1	0	1	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	0	0	1	1	1	1
0	1	1	0	1	1	0	0	0
0	1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	1	1	0
1	0	0	1	1	0	1	1	0
1	0	1	0	0	0	1	1	0
1	0	1	0	1	0	0	0	0
1	0	1	1	0	0	1	1	0
1	0	1	1	1	0	1	1	0
1	1	0	0	0	1	1	0	1
1	1	0	0	1	1	0	0	1
1	1	0	1	0	1	1	1	1
1	1	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0	1
1	1	1	0	1	1	0	0	0
1	1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0	1

Si es satisfacible.

3.4 Experimento sin solución

Proposición vacía.
No se obtiene solución.

4 Conclusiones

Se concluye que se logró identificar exitosamente cuando dos operadores cumplen la condición de satisfacción y cuando son inválidas. Ademas de haberlo de desarollado manualmente, se logró implementarlo en código, mediante SWISH.

5 Referencias

- Lógica proposicional y Teoría de conjuntos, 2016. Recuperado el 28 de octubre de 2021 de: https://repository.eafit.edu.co/har
- Lógica proposicional ¿Qué es?, 2019, Software DELSOL. Recuperado el 28 de octubre de 2021 de: https://www.sdelsol.com/g proposicional/