Universidad Nacional Autonoma de México Facultad de Ingeniería

Inteligencia Artificial

Proyecto 2

Saldaña Hernandez Norman Brandon
Guerrero Ferrusca Francisco
González Lorenzo Daniela Michel
Tenorio Veloz Alisson Dafne

October 28, 2021

Planteamiento del problema

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

Solución basada en lógica proposicional

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

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

LÓGICA 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.

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.

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.

CONECTORES LÓGICOS

- 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.

TABLAS DE VERDAD

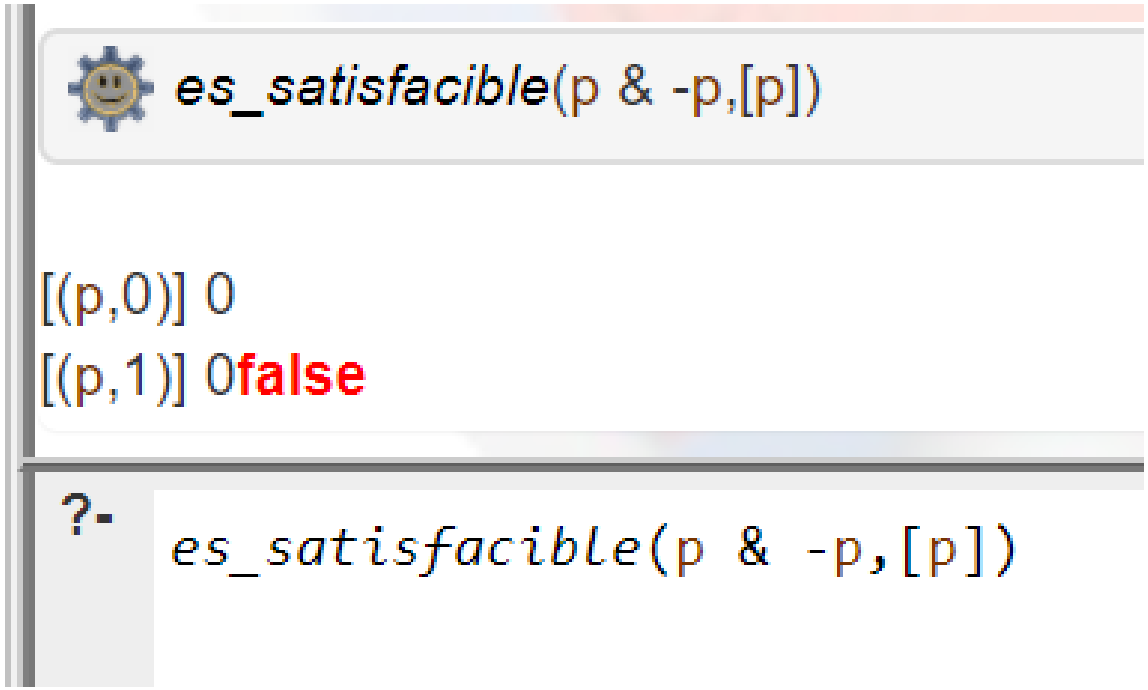
Para saber si una proposición compleja es verdadera o falsa necesitamos saber si las proposiciones simples que la componen son verdaderas o falsas. Dependiendo del resultado final según la combinación de estas proposiciones simples, la tabla de verdad de la compleja puede ser de tres tipos:

- 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.

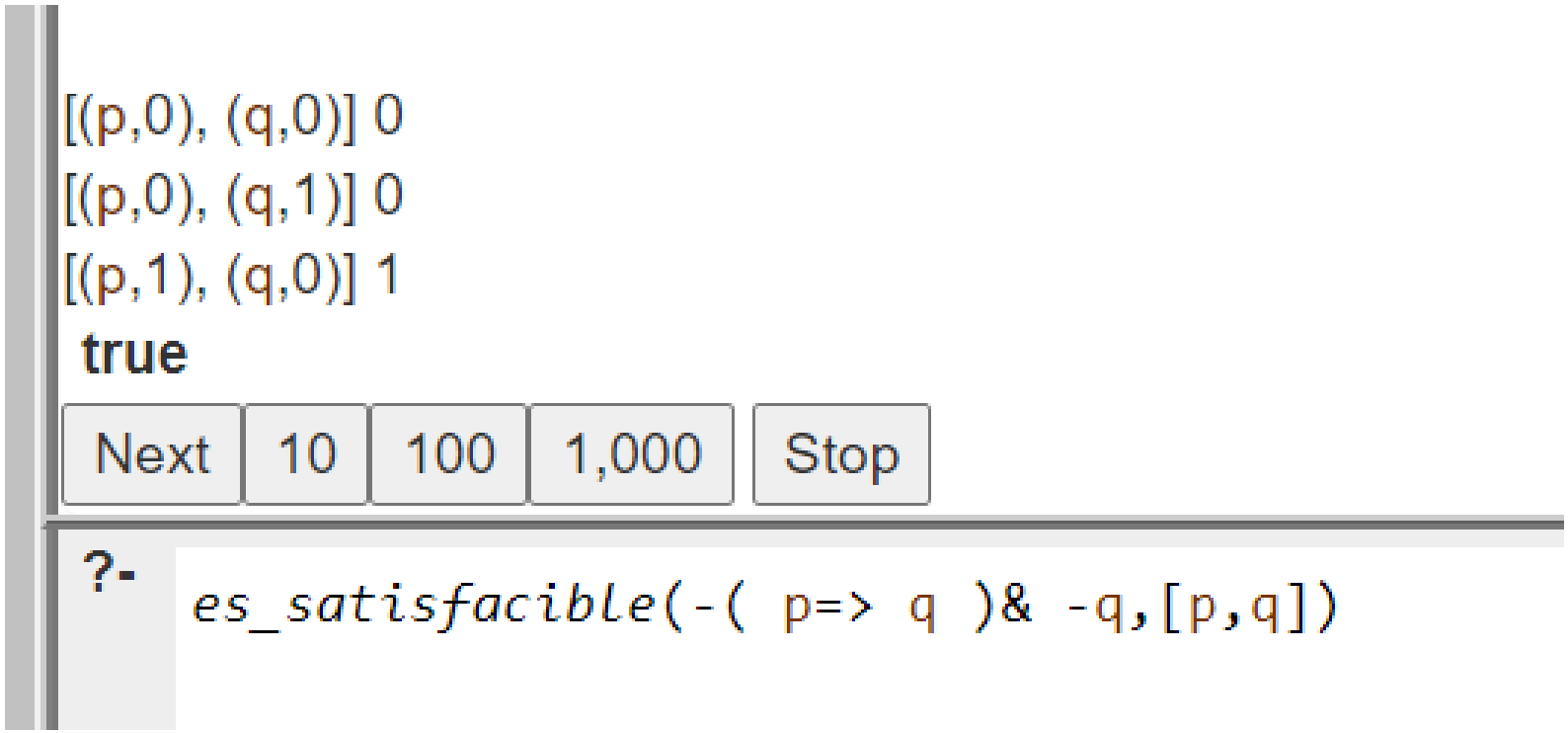
SATISFACIBILIDAD

Una fórmula es satisfacible si la última columna de su tabla de verdad contiene al menos un valor verdadero. O en otras palabras, si existe al menos un modelo de la formula.

Desarrollo del código. Como usar el código



Desarrollo del código. Como usar el código



Desarollo del código. Estrategia de resolución

Dada la fórmula -p v q, con la tabla de verdad:

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

La idea más intuita es probar todas las posibles estructuras, y con que alguna de ellas dé verdadero tenemos un modelo, y por lo tanto la formula es satisfacible.

Desarrollo del código. ¿La formula es satisfacible?

```
41 %La condición de satisfacción de una formula es que tenga al menos 1 modelo. Un modelo es una
42 %estructura que al aplicar a la formula hace que de verdadero. Para saber si una formula
43 %es satisfacible es conveniente revisar estructura por estructura (00,01,10,11; para dos variables por
44 % hasta que una de ellas sea un modelo es decir
45 es_satisfacible(FORMULA,VARIABLES) :-
46 getEstructuras(VARIABLES,ESTRUCTURA), %obtiene las posibles estructuras
47 es_modelo(ESTRUCTURA,FORMULA).% evalua las estrcuturas para comprobar si alguna es modelo,
48 %si lo es entonces la formula es satisfacible
```

Desarrollo del código. Definición de operadores

```
1 :- op(1, fy, -). % negación
2 :- op(2, xfy, &). % conjunción
3 :- op(2, xfy, v). % disyunción
4 :- op(3, xfy, =>). % implicación
5 :- op(3, xfy, <=>). % Doble Implicación
6
```

Desarrollo del código. es_modelo

```
6
7  %si al evaluar la formula en M el resultado es 1, se dice que M es modelo de l
8  es_modelo(ESTRUCTURA,FORMULA) :-valor(FORMULA,ESTRUCTURA,V),
9      nl,
10     write(ESTRUCTURA),
11     write("  "),
12     write(V),
13     V = 1.
```

Desarrollo del código.Obtener las posibles estructuras

```
35 %obtención de las estructuras M. p.e para el conjunto de variables proposicionales
36 % ingresas un conjungo de variables p.e [p,q], devuelve las posibles estructuras:
37 % [(p,0),(q,0)] , [(p,0),(q,1)], [(p,1),(q,0)], [(p,1),(q,1)]
38 getEstructuras([],[]).
39 getEstructuras([VARIABLE|VARIABLES_COLA],[(VARIABLE,VALOR)|ML]) :-
40 valor_verdad(VALOR),
41 getEstructuras(VARIABLES_COLA,ML).
42
```

Desarrollo del código.Valores de verdad

```
3 %Los valores que pueden tomar las variables son 0 y 1, si ponemos valor_v
4 valor_verdad(0).
5 valor_verdad(1).
```

Desarrollo del código. Obtención del valor de una formula dada una estructura

```
4
5 % descompone la formula y evalua. p.e (p => q) v (q => p) se descompone en A = (p =>q) v B = (q => p).
6 % %Luego descompone A = (p) => B= (q). Con memberchk obtiene el valor de p y de q, finalmente
7 %evalua de adentro hacia afuera (por orden de operadores) las definiciones de las funciones, en este ca
8 valor(F, ESTRUCTURA, V) :-memberchk((F,V), ESTRUCTURA).
9 valor(-A, ESTRUCTURA, V) :-valor(A, ESTRUCTURA, VA),eval(-, VA, V).
0 valor(F, ESTRUCTURA, V) :-F =..[Op,A,B], valor(A, ESTRUCTURA, VA),valor(B, ESTRUCTURA, VB),
1 eval(Op, VA, VB, V).
```

Desarrollo del código. Evaluación de formulas

```
22
23 %eval contiene las definiciones de
24 eval(v, 0, 0, 0) :- !.%Definición de disyuncion (or)
25 eval(v, _, _, 1).
26 eval(&, 1, 1, 1) :- !.%Definición de conjuncion (and)
27 eval(&, _, _, 0).
28 eval(=>, 1, 0, 0) :- !.%definición de implicación (=>)
29 eval(=>, _, _, 1).
30 eval(<=>, X, X, 1) :- !.%definición de doble implicación (<=>)
31 eval(<=>, _, _, 0).
32 eval(-, 1, 0). %definición de negación (-)
33 eval(-, 0, 1).
34
```

Experimentos

Experimento de 1 variable

p & -p

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

No es satisfacible (insatisfacible).

Experimentos de baja dificultad

Experimento 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.

Experimento de media dificultad

Experimento 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.

Experimento de alta dificultad

((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

Experimento sin solución

Experimentode 0 variables

Proposición vacía.
No se obtiene solución.

Conclusiones

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

Referencias

- Lógica proposicional y Teoría de conjuntos, 2016. Recuperado el 28 de octubre de 2021 de: https://repository.eafit.edu.co/handle/10784/9774
- Lógica proposicional ¿Qué es?, 2019, Software DELSOL. Recuperado el 28 de octubre de 2021 de: https://www.sdelsol.com/glosario/logica-proposicional/