Representación del conocimiento
Tarea 2

Raúl Daniel García Ramón
rauld.garcia95@gmail.com
zs22000520@estudiantes.uv.mx

28 de marzo de 2023

Índice

1. COVID

A partir del lineamiento estandarizado para la vigilancia epidemiológica y por laboratorio de COVID-19 de la Secretaría de Salud (disponible en el repositorio Teams), defina en lógica proposicional los conceptos de caso sospechoso y caso confirmado de COVID-19 (pág. 12). Defina otro concepto del documento. [30/100]

Caso sospechoso:

Persona de cualquier edad que presente enfermedad respiratoria aguda leve o grave y que cuente con alguno de los siguientes antecedentes hasta 14 días del inicio de síntomas:

- Haber estado en contacto con un caso confirmado o bajo investigación a COVID-19, o
- Viaje o estancia a países con transmisión local comunitaria* de COVID-19. (*China, Hong Kong, Corea del Sur, Japón, Italia, Irán y Singapur)

Al definirlo en lógica proposicional queda de la siguiente manera:

$PersonaXEdad \land EnfermedadRespiratoria \land ((ContactoCasoConfirmado \lor ContactoCasoBajoInvestigacion) \lor Viaje) \implies CasoSospechoso$

Caso confirmado:

Persona que cumpla con la definición operacional de caso sospechoso y que cuente con diagnóstico confirmado por laboratorio emitido por el InDRE.

Al definirlo en lógica proposicional quedaría de la siguiente manera:

$CasoSospechoso \land DiagnosticoConfirmado \implies CasoConfirmado$

El médico de primer contacto después de identificar a un paciente con sospecha de COVID-19, deberá realizar el interrogatorio y atención del caso bajo medidas de precaución estándar y por gotas en un cubículo aislado,

bien ventilado y mantener la puerta cerrada.

Al definirlo en lógica proposicional queda de la siguiente forma:

$IdentificarPacienteConSospecha \implies RealizarInterrogatorio \wedge AtencionBajoEstandar \wedge PorGotasEnCubiculoAislado \wedge BienVentilado \wedge PuertaCerrada$

2. Implementación CNF

Implemente en Prolog el algoritmo CNF visto en clase, para convertir una fbf proposicional en su equivalente en forma normal conjuntiva. Pruebe su implementación con el ejemplo visto en clase. [30/100]

El código implementado es el siguiente:

```
% operadores proposicionales
:- op(700,xfy,v).
:- op(600,xfy,^).
:- op(500,fy,~).

% caso base si es una literal regresa la literal
impl_free(X,X):-
    atom(X).

% caso base si es una literal negada regresa la literal negada
impl_free(~X,~X):-
    atom(X).

% llama recursivamente a implfree para cada argumento del and
impl_free(X ^ Y,IMPLFREE):-
    impl_free(X,Izq),
    impl_free(Y,Der),
    IMPLFREE=((Izq) ^ (Der)).

% llama recursivamente a implfree para cada argumento del and
impl_free(X v Y,IMPLFREE):-
    impl_free(X,Izq),
    impl_free(Y,Der),
```

```
24      IMPLFREE=((Izq) v (Der)).
25
26  % cambia la implicación X => Y por ~X v Y
27  impl_free(X => Y,IMPLFREE):-
28      impl_free(X,Izq),
29      impl_free(Y,Der),
30      IMPLFREE=(~(Izq) v (Der)).
31
32
33  % NNF ----------------------------------------------------
34  % caso base si es una literal regresa la literal
35  nnf(X,X):-
36      atom(X).
37
38  % caso base si es una literal regresa la literal
39  nnf(~X,~X):-
40      atom(X).
41
42  % elimina doble negación
43  nnf(~(~X),X).
44
45  % llama a nnf para el lado izquierdo y derecho
46  % y el resultado lo une con ^
47  nnf(X ^ Y,NNF):-
48      nnf(X,Izq),
49      nnf(Y,Der),
50      NNF=((Izq)^(Der)).
51
52  % llama a nnf para el lado izquierdo y derecho y
53  % el resultado lo une con v
54  nnf(X v Y,NNF):-
55      nnf(X,Izq),
56      nnf(Y,Der),
57      NNF=((Izq) v (Der)).
58
59  % llama a nnf para el lado izquierdo negado
60  % y derecho negado y el resultado lo une con ^
61  nnf(~(X ^ Y),NNF):-
62      nnf(~X,Izq),
63      nnf(~Y,Der),
```

```
64      NNF=((Izq) v (Der)).
65
66  % llama a nnf para el lado izquierdo y derecho
67  % y el resultado lo une con v
68  nnf(~(X v Y),NNF):-
69      nnf(~X,Izq),
70      nnf(~Y,Der),
71      NNF=((Izq) ^ (Der)).
72
73  % CNF ------------------------------------------------
74  % si es literal regresa literal
75  cnf(X,X):-
76      atom(X).
77
78  % si es literal negada regresa literal negada
79  cnf(~X,~X):-
80      atom(X).
81
82  % si es and llama recursivamente a cnf para cada lado
83  % y regresa el and de ambos
84  cnf(X ^ Y,CNF):-
85      cnf(X,Izq),
86      cnf(Y,Der),
87      CNF=((Izq) ^ (Der)).
88
89  % si es or llama recursivamente a cnf para cada lado
90  % y llama a distr con ambos
91  cnf(X v Y,CNF):-
92      cnf(X,Izq),
93      cnf(Y,Der),
94      distr(Izq,Der,CNF),!.
95
96  % llama a impl_free para quitar las implicaciones
97  % llama a nnf con lo que regrese implfree
98  % para llevar las negaciones a las literales
99  cnf(X,CNF):-
100     impl_free(X,IMPLFREE),
101     nnf(IMPLFREE,NNF),
102     cnf(NNF,CNF),!.
103
```

```
% DISTR ------------------------------------------------------

% si X es and llama recursivamente con X1 y Y y con X2 y Y
% y regresa el and de lo que regrese la llamada recursiva
distr(X,Y,DISTR):-
    X= X1 ^ X2,
    distr(X1,Y,Izq),
    distr(X2,Y,Der),
    DISTR=(Izq ^ Der).

% si Y es and recursivamente con X y Y1 y con X y Y2
% y regresa el and de lo que regrese la llamada recursiva
distr(X,Y,DISTR):-
    Y= Y1 ^ Y2,
    distr(X,Y1,Izq),
    distr(X,Y2,Der),
    DISTR=(Izq ^ Der).

% si no hay and regresa el or de ambos
distr(X,Y,DISTR):-
    DISTR=(X v Y).
```

La función impl-free en el ejemplo visto en clase obtiene lo siguiente:

```
?- impl_free(~p ^ q => p ^ (r => q), IMPLFREE).
IMPLFREE = (~ (~p^q)v p^(~r v q))
```

Mientras que al probar el código desarrollado se obtiene lo siguiente:

```
?- impl_free(~p ^ q => p ^ (r => q), IMPLFREE).
IMPLFREE = (~ (~p^q)v p^(~r v q)).
```

Como se puede ver se obtiene el mismo resultado que en el ejemplo visto en clase. Ahora, al probar este resultado con el NNF en el ejemplo visto en clase, se obtiene lo siguiente:

```
?- impl_free(~p ^ q => p ^ (r => q), IMPLFREE), nnf(IMPLFREE,NNF).
IMPLFREE = (~ (~p^q)v p^(~r v q)),
NNF = ((p v ~q)v p^(~r v q))
```

Al probar el código implementado se obtiene lo siguiente:

```
?- impl_free(~p ^ q => p ^ (r => q), IMPLFREE), nnf(IMPLFREE,NNF).
IMPLFREE = (~ (~p^q)v p^(~r v q)),
NNF = ((p v ~q)v p^(~r v q))
```

Como se puede observar se obtiene el mismo resultado que en el ejemplo visto en clase. A continuación, al probar el CNF con el ejemplo visto en clase se obtiene lo siguiente:

```
?- cnf(~p ^ q => p ^ (r => q), CNF).
CNF = ((p v ~q)v p)^((p v ~q)v~r v q)
```

Probándolo en la implementación del código se obtiene:

```
?- cnf(~p ^ q => p ^ (r => q), CNF).
CNF = ((p v ~q)v p)^((p v ~q)v~r v q).
```

Se puede ver que el resultado es el mismo que el ejemplo visto en clase. Finalmente, al probar el último ejemplo visto en clase se debe obtener lo siguiente:

```
?- cnf(r => (s => (t ^ s => r)), CNF).
CNF = (~r v ~s v (~t v ~s)v r)
```

Al probarlo en el código implementado se obtiene lo siguiente:

```
?- cnf(r => (s => (t ^ s => r)), CNF).
CNF = (~r v ~s v (~t v ~s)v r).
```

Como se puede observar se obtiene el mismo resultado que en el ejemplo visto en clase, por lo que todos los ejemplos vistos en clase se implementaron de manera correcta.

3. Caso sospechoso CNF

Convierta la definición de caso sospechoso a forma normal conjuntiva, usando su programa CNF. [20/100]

Caso sospechoso:

$PersonaXEdad \wedge EnfermedadRespiratoria \wedge ((ContactoCasoConfirmado \vee ContactoCasoBajoInvestigacion) \vee Viaje) \implies CasoSospechoso$

Al convertirlo a forma normal conjuntiva con el programa queda de la siguiente manera:

```
cnf(((personaxedad ^ enfermedadrespiratoria) ^
((contactocasoconfirmado v contactocasobajoinvestigacion) v viaje)) =>
casosospechoso, CNF).
CNF = ((((~personaxedad v ~enfermedadrespiratoria)v~contactocasoconfirmado)v
casosospechoso)^(((~personaxedad v ~enfermedadrespiratoria)v
~contactocasobajoinvestigacion)v casosospechoso))^
(((~personaxedad v ~enfermedadrespiratoria)v~viaje)v casosospechoso).
```

4. CNF y SAT

Utilice los algoritmos CNF y SAT (disponible en el repositorio de Teams) para verificar que $p \implies q$ es equivalente a $\neg p \vee q$ (Ejercicio 3 de la tarea anterior). [20/100]

Al correr el algoritmo CNF con $p \implies q$ se obtiene lo siguiente:

```
?- cnf(p => q,CNF).
CNF = (~p v q).
```

Al correr en el algoritmo SAT el $\neg p \vee q$ obtenido con CNF se obtiene lo siguiente:

```
?- Clauses = [[false-P,true-Q]],sat(Clauses,[P,Q]).
Clauses = [[false-true, true-true]],
P = Q, Q = true ;
```

```
Clauses = [[false-false, true-true]],
P = false,
Q = true ;
Clauses = [[false-false, true-false]],
P = Q, Q = false.
```

De acuerdo a lo observado se puede notar que si es satisfacible, por lo que si son equivalentes.

Referencias

Bratko, I. (2012). *Prolog programming for Artificial Intelligence*. Pearson, fourth edition.

Clocksin, W. F. and Melish, C. S. (2003). *Programming in Prolog, using the ISO standard*. Springer-Verlag, Berlin-Heidelberg, Germany.

Labra, J. (1998). *Programación Práctica en Prolog*. Universidad de Oviedo.

Nilsson, U. and Maluszynski, J. (2000). *Logic, Programming and Prolog*. John Wiley & Sons Ltd, 2nd edition.

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kauffman Publishers.