Representación del conocimiento
Tarea 2

Angel García Báez
Alumno de la Maestría en Inteligencia Artificial

IIIA Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
Campus Sur, Calle Paseo Lote II, Sección 2a, No 112
Nuevo Xalapa, Xalapa, Ver., México 91097

ZS24019400@estudiantes.uv.mx

3 de abril de 2025

1. Lógica proposicional en base al reglamento de posgrados

Con base en el Reglamento General de Estudios de Posgrado de nuestra universidad, defina en lógica proposicional los requisitos que debe cumplir la persona para presentar el examen para obtener su grado académico (Artículo 65)[30/100]

Se hizo la consulta del Articulo 65 del Reglamento General de Estudios de Posgrado de la Universidad Veracruzana. Dicho articulo abarca los siguientes requisitos para poder presentar el examen de la obtención del diploma o grado académico que deben cumplir los estudiantes:

I Haber acreditado todas las experiencias educativas y actividades académicas que establezca el plan de estudios del programa educativo correspondiente;

II Presentar el certificado de estudio del posgrado cursado;

III Presentar los votos aprobatorios de los sinodales;

IV Presentar la carta responsiva anexa a este reglamento, firmada por el director del trabajo recepcional;

V Aval del Consejo Técnico u órgano equivalente, en caso de haber excedido el tiempo reglamentario para la presentación del examen;

VI No tener adeudo con la Universidad VeracruzaNA;

VII Pagar el arancel correspondiente; y

VIII Las demás que señale la normatividad universitaria.

En un primer intento por descomponer la normativa para pasarla a lenguaje de lógica proposicional, se hizo una lectura detenida de las condiciones para presentar el examen, dicho primer intento busca convertir los enunciados en variables lógicas como se muestra:

I Haber acreditado todas las experiencias educativas (a) y ($\wedge$) actividades académicas que establezca el plan de estudios del programa educativo correspondiente (b);

II Presentar el certificado de estudio del posgrado cursado (c);

III Presentar los votos aprobatorios de los sinodales (d);

IV Presentar la carta responsiva anexa a este reglamento, firmada por el director del trabajo recepcional; (e)

V Aval del Consejo Técnico (f) u ($\vee$) órgano equivalente (g), en caso de haber excedido el tiempo reglamentario para la presentación del examen (h);

VI No tener adeudo con la Universidad Veracruzana; (i)

VII Pagar el arancel correspondiente; y (j)

VIII Las demás que señale la normatividad universitaria. (k).

IX Si se cumplen con las condiciones requeridas, implica que se lleva a cabo la presentación del examen. (l).

Identificadas las posibles variables, se re escribe una versión simplificada antes de pasarla totalmente a lógica proposicional:

Si a se cumple y b se cumple junto con c, d y e. Ademas, si h se presenta, entonces se debe cumplir f o g. Junto con todo lo demás, es indispensable que se cumpla i, j, y k. Si todo se cumple adecuadamente, entonces se lleva a cabo el examen.

Pasándolas en forma de lógica proposicional:

$$((a \wedge b) \wedge c \wedge d \wedge e \wedge (h \implies (f \vee g)) \wedge i \wedge j \wedge k)$$

2. Implementación del algoritmo CNF en prolog

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]

Para la implementación del algoritmo CNF, fue necesario guiarse de el algoritmo en pseudocódigo de la diapositivas de Guerra-Hernández (2025).

El primer paso, fue definir la semántica y la función de los operadores de implicación, negación, conjunción y disyunción como se muestra:

```
%% Declaración de los operadores %%
:- op(700, xfy, ->). % -> para la implicación
:- op(500, xfy, ^).  % '^' para conjunción
:- op(600, xfy, v).  % 'v' para disyunción
:- op(400, fy, ~).   % 'neg' para negación
%% Semantica %%
v(X,Y):- X;Y.              % Disyunción
^(X,Y):- X,Y.              % Conjunción
~(X):- \+ X.               % Negación
->(X,Y):- (\X;Y).          % Implicación
```

Nota, en el código se utilizo el símbolo $\Longrightarrow$ en lugar de ->pero se representa así a lo largo del reporte con cuestiones de incompatibilidad con el símbolo ascii.

Dada la forma que tiene el algoritmo de funcionar, donde primero de sebe eliminar la implicación, manejar las negaciones y finalmente, aplicar el algoritmo CNF, el siguiente paso es mostrar la implementación de la eliminación de la implicación bajo una función recursiva de 3 casos compuestos y 2 casos base (1 cuando la literal es positiva y otro cuando sea negativa, se hizo así para evitar errores y omisiones con los operadores):

```
% Predicado para verificar si una expresión es un literal
es_literal(X) :- atom(X).
es_literal(~X) :- atom(X).

% Eliminación de la implicación (IMPL FREE)
implfree(Literal, Literal):- atom(Literal).
implfree(~X, ~FX):- implfree(X,FX). % Caso para la negación
% Implicación A -> B -> ¬A v B
implfree(A -> B, ~FA v FB):-
implfree(A, FA),
implfree(B, FB).
% Conjunción
implfree(A ^ B, FA ^ FB):-
implfree(A, FA),
implfree(B, FB).
% Disyunción
implfree(A v B, FA v FB):-
implfree(A, FA),
implfree(B, FB).
```

El siguiente paso a dar, es la implementación de la NNF para manejar las formas negativas que resultan de la eliminación de la implicación, para ello se definió la función NNF bajo una función recursiva de 5 casos compuestos y 2 casos base (el caso base donde la literal es positiva y el caso base donde la literal es negativa).

```
%%% NNF %%%
% Transformación a forma normal negativa (NNF)
nnf(X, X) :- atom(X).  % Literal no necesita transformación
nnf(~X, ~X) :- atom(X).  % Negación de un literal
nnf(~ ~X, NNF) :- nnf(X, NNF).  % Eliminar doble negación
% Conjunción
nnf(X ^ Y, NNF) :-
nnf(X, NNF_X),
nnf(Y, NNF_Y),
NNF = NNF_X ^ NNF_Y.
% Disyunción
nnf(X v Y, NNF) :-
nnf(X, NNF_X),
nnf(Y, NNF_Y),
NNF = NNF_X v NNF_Y.
% De Morgan: ~(X ^ Y) -> ~X v ~Y
nnf(~(X ^ Y), NNF) :-
nnf(~X, NNF_X),
nnf(~Y, NNF_Y),
NNF = NNF_X v NNF_Y.
% De Morgan: ~(X v Y) -> ~X ^ ~Y
nnf(~(X v Y), NNF) :-
nnf(~X, NNF_X),
nnf(~Y, NNF_Y),
NNF = NNF_X ^ NNF_Y.
```

El ultimo paso, una vez que se tiene la función para eliminar la implicación y manejar las negaciones del enunciado, el resto es hacer un predicado que permita verificar si la cosa ya es una CNF o si por el contrario, aun falta hacer procesado de la expresión para garantizar que ya sea una CNF. Para ello se define a continuación el CNF con su correspondiente función de distribución para propagarse recursivamente:

```
%%% CNF %%%
% Transformación a forma normal conjuntiva (CNF)
cnf(X, X) :- atom(X).  % Caso base: si es literal, ya está en CNF
cnf(~X, ~X) :- atom(X).  % Negación de un literal
% Caso 1: Conjunción de fórmulas
cnf(X ^ Y, C) :-
cnf(X, CX),
cnf(Y, CY),
C = CX ^ CY.
% Caso 2: Disyunción de fórmulas, distribución sobre la conjunción
cnf(X v Y, C) :-
cnf(X, CX),
cnf(Y, CY),
distribuye(CX, CY, C).
%%% DISTRB %%%
% Caso base: Si ninguno de los dos es una conjunción, simplemente devolvemos A v B
distribuye(A, B, A v B).
% Distribuye A v (B ^ C) -> (A v B) ^ (A v C)
% Caso 1: Si n1 es una conjunción (n1 = n11 ^ n12), entonces distribuimos sobre n2
distribuye(A ^ B, C, R) :-
distribuye(A, C, AC),
distribuye(B, C, BC),
R = AC ^ BC.
% Caso 2: Si n2 es una conjunción (n2 = n21 ^ n22), distribuimos sobre n1
distribuye(A, B ^ C, R) :-
distribuye(A, B, AB),
distribuye(A, C, AC),
R = AB ^ AC.
```

Con todo esto, tenemos un algoritmo CNF funcional que debe ser estructurado bajo el siguiente orden para pedirle las consultas:

```
?.- implfree(FORMULA,R),nnf(R,Q),cnf(Q,FINAL).
```

Finalmente, para probar que la cosa funciona, se pide que se haga la prueba con el ejercicio visto en clase que corresponde a la siguiente formula:

$$(\neg p \wedge q \implies p \wedge (r \implies q))$$

El resultado de ejecutar la consulta al programa es el siguiente:

```
?.- FORMULA =(~p^q-> p^(r-q)),implfree(FORMULA,Q),nnf(Q,P),cnf(P,R).
FORMULA =  (~p^q->p^(r->q)),
Q = ~ (~p^q)v p^(~r v q),
P =  (p v ~q)v p^(~r v q),
R =  ((p v ~q)v p)^((p v ~q)v~r v q)
```

Llega satisfactoriamente a lo mismo que esta en las diapositivas de clase.

Cabe mencionar que debido a la forma en que esta construido, el programa da pie a hacer una reconsideración y encuentra otra formula de la cual se duda sobre su correcto resultado.

A continuación se muestra por fines ilustrativos:

```
FORMULA =  (~p^q->p^(r->q)),
Q = ~ (~p^q)v p^(~r v q),
P = R, R =  (p v ~q)v p^(~r v q);
false.
```

3. Conversión de lógica proposicional a CNF

Convierta los requisitos del ejercicio uno a forma normal conjuntiva, usando su programa CNF. [20/100]

Se retoma la propuesta dada en el ejercicio uno de este reporte, donde se busca convertir los requisitos para presentar el examen de grado a su forma CNF.

La expresión que se propuso fue:

$$((a \wedge b) \wedge c \wedge d \wedge e \wedge (h \implies (f \vee g)) \wedge i \wedge j \wedge k)$$

A continuación se muestra la consulta y los resultados obtenidos en prolog:

```
?.- FORMULA = (a^b^c^d^e^(h -> (f v g))^i^j^k),
        implfree(FORMULA,Q),nnf(Q,P),
        cnf(P,R).
        FORMULA = a^b^c^d^e^(h->f v g)^i^j^k,
        Q = P,
        P = R,
        R = a^b^c^d^e^(~h v f v g)^i^j^k .
```

Lo más que hace en esta expresión, dada su estructura es hacer la eliminación de la implicación, de ahí en fuera, el algoritmo considera que ya esta en forma cnf.

4. Conversión de logica proposicional a CNF

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

Primero, se hace uso del algoritmo CNF implementado y descrito previamente para ejecutar la query relacionada al ejercicio 3 de la tarea 1.

Mediante la implementación de CNF, el resultado obtenido para la consulta es el siguiente:

```
?.- FORMULA = (p -> q),implfree(FORMULA,Q),nnf(Q,P),cnf(P,R).
FORMULA = (p -> q),
P = Q,
Q = R,
R = ~p vq.
```

El resultado obtenido por el CNF es el esperado (en el fondo solo aplico la eliminación de la implicación, como ya estaba en forma conjuntiva, no fue necesario que se dispararan más cosas como el nnf y el cnf como tal).

A continuación se muestra el resultado de la consulta en el algoritmo SAT:

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

El algoritmo SAT llega a computar que P y Q deben de ser ambos verdaderos, y de esta forma se cumple que *pimpliesq* y que a su vez, $\neg p \wedge q$, dado que si vemos sus valores de tabla de verdad, son equivalentes. Para este caso particular, si P es verdadero (lo niega la negación de P) y Q es verdadero, por el OR, hace que sea verdadera toda la expresión.

5. Referencias

Referencias

Guerra-Hernández, A. (2025). Deducción natural. Notas de clase, Capítulo 3, Curso de Representación del Conocimiento, Universidad Veracruzana.