Jose Revete
19-10040

Tarea 3 (Prolog)

1. **Se desea que implemente en Prolog una calculadora para números de Church.**

R:

```
% a) suma/3
suma(zero, Y, Y).
suma(next(X), Y, next(Z)) :- suma(X, Y, Z).

% b) resta/3
resta(X, zero, X).
resta(next(X), next(Y), Z) :- resta(X, Y, Z).

% c) producto/3
producto(zero, _, zero).
producto(next(X), Y, Z) :- producto(X, Y, W), suma(W, Y, Z).
```

2. **Considere hechos de la forma arco(A, B) en ProLog, representando una conexión dirigida desde un nodo A hasta un nodo B.**

R:

```
% Definición de arcos
arco(a, b).
arco(a, c).
arco(b, d).
arco(c, d).
arco(c, e).

% a) Predicado hermano
hermano(A, B) :-
    arco(C, A),
    arco(C, B),
    A \= B.

% b) Predicado alcanzable
alcanzable(A, B) :-
    alcanzable(A, B, []).

alcanzable(A, B, _) :-
    arco(A, B).

alcanzable(A, B, Visitados) :-
    arco(A, C),
    C \= B,
    \+ member(C, Visitados),
    alcanzable(C, B, [C|Visitados]).

% c) Predicado lca
lca(A, B, C) :-
    alcanzable(C, A),
    alcanzable(C, B),
    \+ (arco(C, D), alcanzable(D, A), alcanzable(D, B)).

% d) Predicado tree
tree(A) :-
    findall(Node, alcanzable(A, Node), Nodes),
    \+ ciclo(A, []),
    forall(member(Node, Nodes), unico_camino(A, Node)).
```

```
% Predicado auxiliar para verificar si hay un ciclo en el grafo
ciclo(Node, Visitados) :-
    arco(Node, Vecino),
    member(Vecino, Visitados).
ciclo(Node, Visitados) :-
    arco(Node, Vecino),
    \+ member(Vecino, Visitados),
    ciclo(Vecino, [Node|Visitados]).

% Predicado auxiliar para verificar si hay un camino unico entre A y B
unico_camino(A, B) :-
    findall(Camino, camino(A, B, [], Camino), Caminos),
    length(Caminos, 1).

% Predicado auxiliar para encontrar un camino entre A y B
camino(A, B, Visitados, [A, B]) :-
    arco(A, B),
    \+ member(A, Visitados).
camino(A, B, Visitados, [A|Camino]) :-
    arco(A, C),
    \+ member(A, Visitados),
    camino(C, B, [A|Visitados], Camino).
```

INVESTIGACIÓN:

1. **Decimos que una fórmula lógica es ground si no hay variables libres que ocurren en dicha fórmula.**
    a. **Considere el dominio de valores {a, b, c}. ¿Cuántas posibles fórmulas ground (en el dominio sugerido) se pueden construir a partir de la fórmula p(X) ∧ q(Y)?**

R: Serian **9 formulas**: p({a,b,c}) y q({a, b, c}), habrian 3 posibilidades por cada una, por lo tanto 3x3= **9**

    b. **Aumente ahora el dominio, con dos funcionales f (de adicidad 1) y g (de adicidad 2). ¿Cúantas posibles formulas ground (en el dominio sugerido) se pueden construir ahora a partir de la formula p(X) ∧ q(Y)?**

R: Al ser X y Y variables, encontramos que las funciones f y g definen nuevos valores para el dominio de valores, los cuales a su vez pueden ser usados nuevamente por X y Y, lo que produce infinita cantidad de fórmulas ground.

    c. **Plantee el conjunto de posibles valores que pueden reemplazar bien sea a X o a Y en la fórmula de la pregunta anterior (Nota: Puede utilizar las operaciones básicas de conjuntos, así como definiciones inductivas).**

R: Definiéndolo inductivamente:

Base: {a,b,c}

Paso inductivo:

Si $t \in T$, entonces $f(t) \in T$.

Si $t \in T$ entonces $g(t, f(c)) \in T$.

Por lo tanto:

$T = \{ a,b,c \} \cup \{ f(t) \mid t \in T \} \cup \{ g(t, f(c) \mid t \in T \}$.

d. Diga un modelo para el programa (la conjunción de todas las fórmulas). Un modelo es un conjunto de predicados (hechos) ground que hacen ciertas todas las reglas.

R:

- “q(a,a)”, “q(b,a)”, “q(c, a)”, etc.
- “q(g(f(f(b)), a), f(f(b)))”, “q(g(f(f(b)), b), c)”, “q(g(c, s), f(f(b)))”, “q(g(c, p),c)”
- r(f(f(b)))
- r(c)
- p(f(_))

e. Considerando el mismo programa, diga cuales predicados ground son ciertos sin ejecutar el programa (solo hechos).

R:

- r(f(f(b)))
- r(c)

f. Considerando el mismo programa, diga cuales predicados ground son ciertos ejecutando a lo sumo una sola vez las reglas.

R:

- q(g(X, Y), Z)
- q(X, a)

g. Plantee una ecuación general recursiva H_k, que dado el conjunto de predicados ground que son ciertos ejecutando a lo sumo k − 1 veces las reglas (en otras palabras, H_k−1), calcule dicho conjunto ejecutando las reglas a lo sumo k veces.

R: Para plantear la ecuacion general, el conjunto H_k se obtiene a partir de H _k−1 agregando los nuevos hechos que se pueden derivar aplicando las reglas sobre H _k−1, dado que si tenemos H_0 esto podría tener reglas in la necesitas de ejecutarse, para H_1, puede tener reglas que basta con ejecutar una única vez mas las de H_0, y así sucesivamente, por lo tanto:

H_k = H_k−1 ∪ {reglas nuevas derivadas de las reglas aplicadas a H_k−1}

h. Sea k* el valor más bajo de k tal que H_k = H_k+1 (mínimo punto fijo de H). Diga si el modelo que encontró en la parte (d) corresponde a un subconjunto de H_k*. ¿Su respuesta será igual para cualquier otro modelo del programa?

R:

Sí, el modelo encontrado en la parte (d) es un subconjunto de H_k*. Tomando en cuenta que para p(f(X)), k es 2, y el valor más bajo es si H_k = H_k+1, y se cumple para k=2, por lo tanto también encontramos que esta respuesta será igual para cualquier otro modelo del programa.

i. Diga el nivel correspondiente a los predicados p, q y r

R:

- El predicado r(c) no depende de otros predicados; es un hecho base. Por lo tanto **r está a nivel 0**.
- q(X,Y) depende de r(X) (negado) y de sí mismo q(Y,a) (positivo). Entonces al ser de nivel 0, q debe ser, al menos, de nivel 1 para cumplir la regla de negación. Por lo que **el nivel de q es 1**.

- p(X) depende de q(X,Y) (positivo), q(X,X) (negado) y de r(Y) (negado). Sabiendo que q es de nivel 1, entonces p debe ser, al menos, de nivel 2 para cumplir la negación sobre q(X,X). Por lo tanto **p está en nivel 2**.

j. Calcule H_k* para el programa anterior: Tomando en cuenta primero los predicados de nivel 0 (de haberlos), luego los predicados de nivel 1 (de haberlos) y así en adelante.

R:

- H_0* = { r(c) }
- H_1* = { q(c,a) } (esto se debe a que la primera comprobación que hace es not(r(X)), q(Y, a), entonces not(r(c)) seria y al entrar en q(X,a) es true. Pero si es distinto a q(c,a), entonces not(r(X)) daria true, y estaría en bucle comprobando en q(Y,a)).
- H_2* = { p(c) }

k. Si existen ciclos de predicados negados en un programa, los niveles antes mencionados no están bien definidos. Discuta cómo afecta esto el cálculo de H_k* .

R: Cuando hay ciclos de predicados que dependen de negaciones entre sí, no es posible calcular H_k* de manera clara porque se crea una especie de "bucle". Cada predicado necesita saber el valor del otro para definirse, y esto genera situaciones donde no se puede determinar la certeza de dicha definición.