Evaluación

Solución

Ejercicio 2

a)

i. Verdadero.

Un programa lógico es un conjunto de cláusulas definidas, eso implica que cada cláusula tiene un literal positivo.

Una cláusula es verdadera en una interpretación si un literal al menos es verdadero.

Basta con elegir una interpretación que haga verdaderos todos los literales positivos del programa para que esa interpretación sea un modelo.

ii. Falso.

Sea el objetivo definido $G = \neg p$

p falso es un modelo para G

iii. Falso.

Existen respuestas correctas que no son computadas, ya que el intérprete da la respuesta más general y cualquier instancia de esta será una respuesta correcta pero no computada.

b)

i. Dada una fórmula de la lógica proposicional, determinar si existe o no una asignación de valores de verdad a las variables que haga la fórmula verdadera, y, en caso afirmativo, devolver esta asignación.

ii. Sí, ya que α implica lógicamente β si $\alpha \wedge \neg\beta$ es insatisfacible, o sea cualquier algoritmo para SAT contesta que no hay modelo.

iii. Sí, ya que una fórmula α es válida si $\neg\alpha$ es insatisfacible.

Ejercicio 3

```
a) %resMaximo(+G,+N)
resMaximo(G,N) :-
    resMaximo(G,0,N,_).
```

```
%resMaximo(+G,+Hini,+N,-Hfin) <- resolver el objetivo G, la altura inicial
es Hini, la altura máxima es N, y %tiene como salida en Hfin la altura del
árbol para resolver G
```

```
resMaximo(true,H,N,H) :- !,H =< N.
resMaximo((A,Gs),H,N,H3) :-
    resMaximo(A,H,N,H2),
    resMaximo(Gs,H2,N,H3).
```

```
resMaximo(G,H,N,H2) :-
    H < N, H1 is H + 1,
    clause(G,Body),
    resMaximo(Body,H1,N,H2).
```

b)

```
s --> w(Y), s1, w(Y).
s1 --> [a], s1, [a].
s1 --> [b], s1, [b].
s1 --> [].
w([a|Y]) --> [a], w(Y).
w([b|Y]) --> [b], w(Y).
w([]) --> [].
```

Ejercicio 4

a)

Las soluciones son:

- A = [1,3,4] B = [2]
- A = [1,2,4] B = [3]
- A = [3,4] B = [1,2]
- A = [2,4] B = [1,3]

Como justificación, este es el 'trace' [otra opción es hacer el árbol SLD]:

```

← p([1,2,3,4],A,B)
(c.1) p([1,2,3,4],[1|_G1],_G2) ← q([2,3,4],_G1,_G2)
      (c.3) falla
      (c.4) q([2,3,4],[3|_G3],[2|_G4]) ← q([4],_G3,_G4)
            (c.3) q([4],[4],[ ]). solución: A=[1,3,4] B=[2]
            (c.4) falla
            (c.5) falla
      (c.5) q([2,3,4],[2|_G3],[3|_G4]) ← q([4],_G3,_G4)
            (c.3) q([4],[4],[ ]). solución: A=[1,2,4] B=[3]
            (c.4) falla
            (c.5) falla
(c.2) p([1,2,3,4],_G1,[1|_G2]) ← g([2,3,4],_G1,_G2)
      (c.3) falla
      (c.4) q([2,3,4],[3|_G3],[2|_G4]) ← q([4],_G3,_G4)
            (c.3) q([4],[4],[ ]). solución: A=[3,4] B=[1,2]
            (c.4) falla
            (c.5) falla
      (c.5) q([2,3,4],[2|_G3],[3|_G4]) ← q([4],_G3,_G4)
            (c.3) q([4],[4],[ ]). solución: A=[2,4] B=[1,3]
            (c.4) falla
            (c.5) falla
    
```

b)

- i. El cut ahí no tiene ningún efecto, porque cuando unifica con la cláusula 3 no puede unificar con las otras cláusulas, por lo que no está cortando otras posibles soluciones. Las soluciones son las mismas que en (a).
- ii. Siempre que unifique la cláusula 4, también unificará la 5. Por lo tanto el cut en la cláusula 4 hace que las ramas marcadas con (c.5) nunca se recorran. Así que las soluciones quedan:

- A = [1,3,4] B = [2]
- A = [3,4] B = [1,2]

c) El cut rojo cambia la semántica del programa (se devuelven distintas soluciones) mientras que el verde no la cambia.

El cut (i) es verde si consideramos invocaciones con el primer argumento instanciado. Si el primer argumento no está instanciado es rojo.

El cut (ii) es rojo.

Ejercicio 5

```

%generarConjetura(-Conjetura)
generarConjetura(Conjetura) :-
    Conjetura = [_,_,_,_],
    selects(Conjetura, [1,2,3,4,5,6,7,8,9,0]).

selects([],_).
selects([X|Xs], L) :-
    select(X, L, L2),
    selects(Xs, L2).

%chequearConsistencia(+RespPrevias,+Conjetura)
chequearConsistencia([], _).
chequearConsistencia([(ViejaConjetura, B, R)|Consultas], Conjetura) :-
    buenasYRegularesCoinciden(ViejaConjetura, B, R, Conjetura),
    chequearConsistencia(Consultas, Conjetura).
    
```

```
buenasYRegularesCoinciden(ViejaConjetura, B, R, Conjetura) :-
    coincidenciasExactas(ViejaConjetura, Conjetura, N1),
    B = N1,
    coincidenciasEnComun(ViejaConjetura, Conjetura, N2),
    R is N2 - B.
```

```
coincidenciasExactas(Xs, Ys, N) :-
    coincidenciasExactas(Xs, Ys, 0, N).
```

```
coincidenciasExactas([], [], N, N).
coincidenciasExactas([X|Xs], [X|Ys], K, N) :-
    !,
    K1 is K+1,
    coincidenciasExactas(Xs, Ys, K1, N).
```

```
coincidenciasExactas([_|Xs], [_|Ys], K, N) :-
    coincidenciasExactas(Xs, Ys, K, N).
```

```
coincidenciasEnComun(Xs, Ys, N) :-
    coincidenciasEnComun(Xs, Ys, 0, N).
```

```
coincidenciasEnComun([], _, N, N).
coincidenciasEnComun([X|Xs], Ys, K, N) :-
    member(X, Ys),
    !,
    K1 is K+1,
    coincidenciasEnComun(Xs, Ys, K1, N).
```

```
coincidenciasEnComun([_|Xs], Ys, K, N) :-
    coincidenciasEnComun(Xs, Ys, K, N).
```

```
%mastermind(+RespPrevias,-Codigo)
mastermind(RespPrevias, Codigo) :-
    generarConjetura(Conjetura),
    chequearConsistencia(RespPrevias, Conjetura),
    consultarOponente(Conjetura, B, R),
    chequeoSol(Conjetura, B, R, RespPrevias, Codigo),
    !.
```

```
chequeoSol(Codigo, 4, 0, _, Codigo).
chequeoSol(Conjetura, B, R, RespPrevias, Codigo) :-
    mastermind([(Conjetura,B,R)|RespPrevias], Codigo).
```