PLP - Segundo Recuperatorio - 1do cuatrimestre de 2024

#Orden	Libreta	Apellido y Nombre	Ej1	Ej2	Ej3	Nota Final
117.	64/22	Kerbs Octavio	B-	B-	B-	A

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido y número de orden en todas las hojas, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolverlo.

Ejercicio 1 - Resolución

a) Representar en forma clausal las siguientes fórmulas de lógica de primer orden:

1. $\exists C.\forall X.(X \in a \Rightarrow X \leq C)$
   Existe una cota superior para el conjunto a.
2. $\forall X.\forall Y.(X \leq Y \Rightarrow n(Y) \leq n(X))$
   Si $X \leq Y$, entonces $-Y \leq -X$.
3. $\forall X.(X \in a \Leftrightarrow n(X) \in b)$
   X pertenece al conjunto a si y solo si -X pertenece al conjunto b.

b) Utilizando resolución, determinar si existe una cota inferior para el conjunto b:
$\exists C.\forall X.(n(X) \in b \Rightarrow C \leq n(X))$.

c) La resolución utilizada en el punto anterior, ¿fue SLD? Justificar.

Ejercicio 2 - Programación Lógica

Implementar los predicados respetando en cada caso la instanciación pedida. Los generadores deben cubrir todas las instancias válidas de aquello que generan sin repetir dos veces la misma. No usar cut (!) ni predicados de alto orden como `setof`, con la única excepción de `not`.

a) Implementar el predicado `generarCapicúas(-L)`, que genera todas las listas capicúas de números naturales (sin incluir al cero). Por ejemplo:

```
?- generarCapicuas(L).
L = [1] ;
L = [2] ;
...
L = [1,3,1] ;
...
L = [2,3,3,2];
...
```

b) El predicado anterior ¿es reversible? Justificar.

c) Definir el predicado `tokenizar(+D,+F,-T)`, que es verdadero cuando `D` representa un diccionario de palabras conocidas (lista de listas de letras), `F` es una lista de letras sin espacios, y `T` es la lista de palabras en la que se puede partir la frase. El predicado debe generar todas las posibles maneras de dividir la frase en palabras conocidas. Por ejemplo:

```
?- tokenizar([[a],[a,b],[b,a],[c]], [a,b,a,c,a,b,a], T).
T = [[a],[b,a],[c],[a],[b,a]];
T = [[a],[b,a],[c],[a,b],[a]];
T = [[a,b],[a],[c],[a],[b,a]];
T = [[a,b],[a],[c],[a,b],[a]];
false.
```

d) Definir el predicado `mayorCantPalabras(+D,+F,-T)` que es verdadero cuando `T` es una tokenización de la frase `F` con el diccionario `D` que tiene la mayor cantidad de palabras. En este inciso no está permitido utilizar estructuras auxiliares.

Ejercicio 3 - Objetos y Deducción Natural

a) Considere las siguientes clases:

```
Object subclass: #A

    m1
    ^[self eval].

    eval
    ^self value.

    value
    ^0.
```

```
A subclass: #B

    m2
    ^[super eval].

    m3
    ^self.

    value
    ^1.

    eval
    ^2.
```

Para cada una de las siguientes expresiones, hacer una tabla donde se indique, en orden, cada mensaje que se envía, qué objeto lo recibe, en qué clase está el método respectivo, y cuál es el resultado final de cada colaboración:

I) `B new m1 value`

II) `B new m3 m2 value`

b) Implementar un *closure* que tome como parámetro una colección y devuelva los elementos de la colección que saben responder al mensaje `#ptff`.
Sugerencia: pueden utilizar el mensaje `#respondsTo:`, que toma un mensaje y devuelve si el objeto sabe responderlo.

c) Demostrar en deducción natural que vale la siguiente fórmula, sin utilizar principios de razonamiento clásicos:

$$\exists X.P(X) \vee \forall Y.Q(Y) \implies \exists X.(Q(X) \vee P(X))$$

(H1) ①

B-

I) $\exists C. \forall x. (x \in a \Rightarrow x \leqslant C)$

$\equiv \exists C. \forall x. (\neg(x \in a) \vee (x \leqslant C))$.

① SKOLEM $\equiv \{\neg(x_1 \in a), (x_1 \leqslant C)\}$ ✓

II) $\forall x. \forall y. (X \leqslant Y \Rightarrow \cap(Y) \leqslant \cap(X))$.

$\equiv \forall x. \forall y. (\neg(X \leqslant Y) \vee (\cap(Y) \leqslant \cap(X)))$.

② $\equiv \{\neg(X_2 \leqslant Y_2), (\cap(Y_2) \leqslant \cap(X_2))\}$. ✓

III) $\forall x. (X \in a \leftrightarrow \cap(x) \in b)$.

$\equiv \forall x. ((X \in a) \rightarrow (\cap(X) \in b)) \wedge$
$((\cap(X) \in b) \rightarrow (X \in a)))$.

$\equiv \forall x. ((\neg(X \in a) \vee (\cap(X) \in b)) \wedge$
$(\neg(\cap(X) \in b) \vee (X \in a)))$.

③ $\equiv \{\{\neg(X_3 \in a), (\cap(X_3) \in b)\},$ ✓
$\{\neg(\cap(X_4) \in b)\}, \{X_4 \in a\}\}$. ✓

④

b) $\neg(\exists C. \forall x. (\cap(X) \in b \rightarrow C \leqslant \cap(X)))$.

$\equiv \forall C. \exists x. ((\cap(X) \in b) \wedge \neg(C \leqslant \cap(X)))$.

SKOLEM $\equiv \{\{\cap(f(C_5)) \in b\}, \{\neg(C_6 \leqslant \cap(f(C_6)))\}\}$

⑤ ✓ ⑥ ✓

⑦. $\{\neg(f(\cap(Y_2)) \leqslant Y_2)\}$ de 6 y 2. ✓

$\sigma_1 = \{C_6 \leftarrow \cap(Y_2), X_2 \leftarrow f(C_6)\}$.

⑧ $\{\neg(f(\cap(c)) \in a)\}$ de 7 y 1 ✓

$\sigma_2 = \{X_1 \leftarrow f(\cap(Y_2)), Y_2 \leftarrow c\}$.

(9) $\{\neg(n(\dagger(n(c)))) \in b\}$ de 8 y 4.

$\sigma_3 = \{X_4 \leftarrow \dagger(n(c))\}$.

(10) $\{\square\}$ de 9 y 5.

$\sigma_4 = \{X_5 \leftarrow n(c)\}$

-> La resolución es SLD porque son todas cláusulas de horn, tue lineal, comenzamos por una cláusula objetivo. (6).

Además existe resolución trivial en todas las ramas.

(2)

a) generar Capicúas(L) :- desde (1, N),
listaDeN(N, L),
capicúa(L).

- listaDeN(0, []).
listaDeN(N, L) :-
N > 0,
between(1, N, N2),
N3 is N - N2,
listaDeN(N3, L1),
append([N2], L1, L). ✓

- capicúa([]). % Asumo que la lista vacía es capicúa.

capicúa([_]).
capicúa(L) :-
append([X], Xs, L),
append(Ys, [X], Xs),
capicúa(Ys). ✓

Esta corrección está mal, capicúa está bien implementado (lo hablé con el corrector)

- desde(X, X).
desde(X, Y) :- N is X+1, desde(N, Y).

b) Con reversible supongo que se refiere a que si yo le doy un arreglo instanciado, me dev. true si es capicúa y false en caso contrario.

Supongamos que lo lista es L = [2]. (es capicúa). desde (1, N) instancio a N en 1. Luego se resuelve listoDeN. Se instancio N2 = 1, se instancio N3 = 0. Se instancio L1 = [], y por último el append consulto si [1|[]] = [2]. Lo cual es falso.

~~=> No es reversible. Porque L = [2] es capicúa pero generan Capicuas no encontró validación.~~

-> Estoy pensando que si se hackeo el "desde", cuando N=2, va a encontrar una instancia donde [2|[]] = L = [2] y es capicúa pero luego va a hacer consultas infinitas. Nos va a dar un true y se va a colgar.

-> Una posible solución sería definir otra regla para generan Capicuas donde chequee si L viene instanciado con noVan(L). Si viene instanciado solo chequeamos si es capicúa. Caso contrario, lo que definí yo (agregando Van(L)).

NO ESCRIBAS COSAS QUE NO TE PEDIMOS, SI ESTÁ BIEN NO SUMA, SI ESTÁ MAL SI RESTA.

c) [illegible]

- tokenizar(_, [], []). ✓

[illegible]

tokenizar(D, F, [P|Xs]) :-
    member(P, D),
    append(P, Ys, F),
    tokenizar(D, Ys, Xs). ✓

d). mayorContPalabras(D, F, T) :-
    [illegible],
    tokenizarLong(D, F, T, N),
    not((tokenizarLong(D, F, _, N2), N2 > N)).

- tokenizarLong(D, F, T, N) :-
    tokenizar(D, F, T),
    length(T, N). ✓

③. a)

B-

mensaje	receptor	Ubicación	resultado.
new	B	Object Clase	aB.
m1	aB	A	[self eval]
value	[self eval]	BlockClosure	2
eval	aB	B	2.
new	B	Object Clase	anotherB
m3	anotherB	B	anotherB.
m2	anotherB	B	[super eval]
value	[super eval]	BlockClosure	1
eval	anotenB	A	1
Value	anotherB	B	1

—> Resultado final

I). 2

I) 1.

b). [:xs |

^ (xs select: [:x | x respondsTo: #ptff])

].

(Este return te saca del metodo)

—> no está especificado el formato que tome ~~de ptff~~. Lo paso como dice el enunciado. "respondsTo:"!

$X \notin fv(\gamma, \exists x.(Q(x) \vee P(x)))$

$$\dfrac{}{\gamma, P(x) \vdash P(x)}\ ax$$

$$\dfrac{\gamma, P(x) \vdash P(x)}{\gamma, P(x) \vdash Q(x) \vee P(x)\ \{x := x\}}\ \vee i_2$$

$$\dfrac{\gamma, P(x) \vdash Q(x) \vee P(x)\ \{x := x\}}{\gamma, P(x) \vdash \exists x.(Q(x) \vee P(x))}\ \exists i$$

$$\dfrac{}{\gamma \vdash \exists x. P(x)}\ ax$$

$$\dfrac{\gamma \vdash \exists x. P(x) \qquad \gamma, P(x) \vdash \exists x.(Q(x) \vee P(x))}{\gamma : \Gamma, \exists x. P(x) \vdash \exists x.(Q(x) \vee P(x))}\ \exists e$$

$$\dfrac{}{\psi \vdash \forall y. Q(y)}\ ax$$

$$\dfrac{\psi \vdash \forall y. Q(y)}{\psi \vdash Q(y)\ \{y := y\}}\ \forall e$$

$$\dfrac{\psi \vdash Q(y)\ \{y := y\}}{\psi \vdash Q(y) \vee P(y)\ \{x := y\}}\ \vee i_1$$

$$\dfrac{\psi \vdash Q(y) \vee P(y)\ \{x := y\}}{\psi : \Gamma, \forall y. Q(y) \vdash \exists x.(Q(x) \vee P(x))}\ \exists i$$

$$\dfrac{}{\Gamma \vdash \exists x. P(x) \vee \forall y. Q(y)}\ ax$$

$$\dfrac{\Gamma \vdash \exists x. P(x) \vee \forall y. Q(y) \qquad \gamma : \Gamma, \exists x. P(x) \vdash \exists x.(Q(x) \vee P(x)) \qquad \psi : \Gamma, \forall y. Q(y) \vdash \exists x.(Q(x) \vee P(x))}{\Gamma : \exists x. P(x) \vee \forall y. Q(y) \vdash \exists x.(Q(x) \vee P(x))}\ \vee e$$

$$\dfrac{\Gamma : \exists x. P(x) \vee \forall y. Q(y) \vdash \exists x.(Q(x) \vee P(x))}{\emptyset \vdash \exists x. P(x) \vee \forall y. Q(y) \Rightarrow \exists x.(Q(x) \vee P(x))}\ \Rightarrow i$$