# PLP - Segundo Parcial - 1er cuatrimestre de 2025

| #Orden | Nro. Libreta | Apellido y Nombre | Ej1 | Ej2 | Ej3 | Nota Final |
|---|---|---|---|---|---|---|
| | | | | | | |

Este examen se aprueba obteniendo al menos dos ejercicios bien (`B`) y uno regular (`R`), y se promociona con al menos dos ejercicios muy bien (`MB`) y uno bien (`B`). Es posible obtener una aprobación condicional con un ejercicio muy bien (`MB`), uno bien (`B`) y uno insuficiente (`I`), pero habiendo entregado algo que contribuya a la solución del ejercicio. Las notas para cada ejercicio son: `-`, `I`, `R`, `B`, `MB`, `E`. 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.

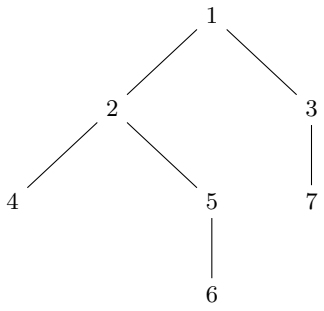
### Ejercicio 1 - Programación Lógica

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`. **No está permitido utilizar estructuras auxiliares**.

Trabajaremos con árboles binarios, usando `nil` y `bin(I, V, D)` para representarlos en Prolog. Como ejemplo, usaremos el siguiente hecho:

```
arbol(bin(bin(bin(nil,4,nil),2,bin(nil,5,bin(nil,6,nil))),1,bin(bin(nil,7,nil),3,nil))).
```

que representa el árbol:




**a.** Definir el predicado `rama(+A,-C)` que es verdadero cuando `C` es un camino desde la raíz del árbol `A` hasta alguna de sus hojas. Por ejemplo:

```
?- arbol(A), rama(A,C).
C = [1,2,4];
C = [1,2,5,6];
C = [1,3,7];
false.
```

**b.** ¿El predicado anterior es reversible en `C`? Justificar brevemente.

**c.** Definir el predicado `ramaMasLarga(+A,-C)` que es verdadero cuando `C` es la rama más larga de `A`. Por ejemplo:

```
?- arbol(A), ramaMasLarga(A,C).
C = [1,2,5,6];
false.
```

**d.** Definir el predicado `ramaUnicaDeLong(+A,+N,-C)` que es verdadero cuando `C` es la única rama de `A` que tiene longitud `N`. Por ejemplo:

```
?- arbol(A), ramaUnicaDeLong(A,4,C).
C = [1,2,5,6];
false.
?- arbol(A), ramaUnicaDeLong(A,3,C).
false.
```

**Ejercicio 2 - Resolución**

**a)** Representar en forma clausal las siguientes fórmulas de lógica de primer orden, que tratan acerca de términos cerrados del cálculo lambda:

i) $\forall T_1.\forall T_2.\forall M.\forall N.((\mathsf{Tipo}(M, T_1 \to T_2) \wedge \mathsf{Tipo}(N, T_1)) \implies \mathsf{Tipo}(\mathsf{app}(M, N), T_2))$
*Si el tipo de un término es $T_1 \to T_2$, y el tipo de otro término es $T_1$, entonces el tipo de la aplicación es $T_2$.*

ii) $\exists M.\mathsf{Tipo}(M, \alpha \to (\beta \to \gamma))$
*Existe un término de tipo $\alpha \to (\beta \to \gamma)$.*

iii) $\exists M.\mathsf{Tipo}(M, \alpha \to \beta)$
*Existe un término de tipo $\alpha \to \beta$.*

iv) $\exists M.\mathsf{Tipo}(M, \alpha)$
*Existe un término de tipo $\alpha$.*

*Ayuda: se puede pensar en el constructor de tipos $\to$ como una función binaria, por ejemplo el término $T_1 \to T_2$ se puede pensar como $\mathsf{flecha}(T_1, T_2)$. Además, $\alpha$, $\beta$ y $\gamma$ son constantes distintas entre sí.*

**b)** Utilizando resolución, determinar si la siguiente fórmula es consecuencia del conjunto anterior: $\exists M.\mathsf{Tipo}(M, \gamma)$ *(Existe un término de tipo $\gamma$).*

Indicar la sustitución utilizada en cada paso. Es importante tener un plan (escrito o en la cabeza).

**c)** ¿Fue SLD la resolución utilizada en el punto anterior? Justificar.

**Ejercicio 3 - Inferencia, Deducción Natural y POO**

**a)** Se extienden el Cálculo Lambda y algoritmo de inferencia para soportar listas de la siguiente manera:

$$\begin{array}{lcl} \tau & ::= & \ldots \mid [\tau] \\ M & ::= & \ldots \mid [\,]_\tau \mid M :: M \mid \mathsf{foldr}\, M\, \mathsf{base} \hookrightarrow M;\ \mathsf{rec}(h, r) \hookrightarrow M \end{array}$$

$\mathcal{I}(\Gamma \mid [\,]_\tau) = ([\tau] \mid \emptyset)$

$\mathcal{I}(\Gamma \mid M_1 :: M_2) = (\tau_2 \mid \{\tau_2 \stackrel{?}{=} [\tau_1]\} \cup E_1 \cup E_2)$
donde:

- $\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1)$
- $\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2)$

$\mathcal{I}(\Gamma \mid \mathsf{foldr}\, M_1\, \mathsf{base} \hookrightarrow M_2;\ \mathsf{rec}(h, r) \hookrightarrow M_3) = (\tau_2 \mid \{\tau_1 \stackrel{?}{=} [X_h], \tau_2 \stackrel{?}{=} \tau_3, \tau_3 \stackrel{?}{=} X_r\} \cup E_1 \cup E_2 \cup E_3)$
donde:

- $\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1)$
- $\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2)$
- $\mathcal{I}(\Gamma, h : X_h, r : X_r \mid M_3) = (\tau_3 \mid E_3)$
- $X_h$ y $X_r$ variables frescas.

Utilizando esta extensión del algoritmo, encontrar el juicio de tipado más general para el siguiente término o indicar por qué no es posible:

$$\mathsf{foldr}\ \ (\lambda x.\mathsf{succ}(x)) :: [\,]\ \ \mathsf{base} \hookrightarrow [\,];\ \ \mathsf{rec}(x, r) \hookrightarrow \mathsf{if}\ p\ x\ \mathsf{then}\ x :: r\ \mathsf{else}\ r$$

**b)** Demostrar el siguiente teorema usando deducción natural, sin utilizar principios clásicos:

$$(\exists X.P(X)) \Rightarrow (\forall Y.(P(Y) \Rightarrow Q(Y))) \Rightarrow \exists Z.Q(Z)$$

**c)** **i)** 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:

**1)** `B new m1 value`

**2)** `B new m3 m2 value`

**ii)** 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.