

1. Dadas las expresiones: `[True, []]` `True: []` `[True]: []` `[[True], []]`

- ☒ Exactamente una de las expresiones está mal tipada
- ☐ Exactamente dos de las expresiones están mal tipadas
- ☐ Las dos anteriores son falsas.

2. Sean las cuatro expresiones: `[True: []]` `[]: [True]` `[True]: []` `[[True], []]`

- ☐ Dos de ellas están mal tipadas
- ☒ Dos de ellas son sintácticamente equivalentes y una está mal tipada
- ☐ Las dos anteriores son falsas.

**Nota: en las preguntas siguientes, sobre sintaxis de listas, suponemos que los numerales 1, 2, ... tienen un tipo concreto, por ejemplo Int.**

3. Dadas las expresiones: `0: [1]` `0: [1]: [2]` `0: [1, 2]` `[0, 1]: [[2]]` `([]: [], 2)`

- ☒ Exactamente una de las expresiones está mal tipada
- ☐ Exactamente dos de las expresiones están mal tipadas
- ☐ Las dos anteriores son falsas.

4. Dadas las expresiones: `[]: [1]` `[1: [2]]: []` `[1: [2]]: [[]]` `[1, 1]: (2: [])` `(1: []): []`

- ☐ Exactamente tres de las expresiones están mal tipadas
- ☒ Exactamente dos de las expresiones están mal tipadas
- ☐ Las dos anteriores son falsas.

5. Dadas las expresiones: `0: [1]` `0: [1]: [2]` `0: [1, 2]` `[0, 1]: [2]` `(1: []): []`

- ☐ Exactamente tres de las expresiones están mal tipadas
- ☒ Exactamente dos de las expresiones están mal tipadas
- ☐ Las dos anteriores son falsas.

6. Dadas las expresiones: `[1]: []` `[[]]: []` `[]: []` `(1: 2): []` `1: (2: [])`

- ☒ Exactamente una de ellas está mal tipada
- ☐ Exactamente dos de ellas están mal tipadas
- ☐ Las dos anteriores son falsas.

7. Dadas las expresiones: `[0]: [1]` `[]: [[]]: []` `[0]: [[]]: []` `[0]: [[1, 2]]` `([[]]: [], [1])`

- ☐ Exactamente una de las expresiones está mal tipada
- ☒ Exactamente dos de las expresiones están mal tipadas
- ☐ Exactamente tres de las expresiones están mal tipadas

8. Dadas las expresiones: `[1]: []` `[1]: [[]]: [2]` `[1]: [[]]` `0: 1: 2` `(0: [1], 2)`

- ☐ Exactamente una de las expresiones está mal tipada
- ☐ Exactamente dos de las expresiones están mal tipadas
- ☒ Las dos anteriores son falsas.

9. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a `[[[0], [], [2, 2]]]`?

- ☒ `((0: []): [[]], 2: 2: []) : []`
- ☐ `[0]: []: [2, 2]: []: []`
- ☐ Ninguna de las anteriores, porque de hecho la expresión está mal tipada

9/9

9/9

10. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a  $[[1, []], [3, 4]]$ ?
- ☐  $((1: []): [ [], 3: 4: []]): []$
  - ☐  $[1]: []: [3, 4]: []: []$
  - ☒ Ninguna de las anteriores, porque de hecho la expresión está mal tipada
11. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a  $[[3, 4], []]$ ?
- ☐  $3: 4: ([ [], []])$
  - ☒  $(3: 4: []): []: []$
  - ☐ Ninguna de las anteriores, porque de hecho la expresión está mal tipada
12. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a  $(1: []): (1: 2: []): []$ ?
- ☒  $[[1], [1, 2]]$
  - ☐  $[[1], [1, 2], []]$
  - ☐ Ninguna de las anteriores, porque de hecho la expresión está mal tipada
13. ¿Cuántas de las siguientes expresiones son sintácticamente equivalentes a  $[[1, 2], []]$ ?
- $1: 2: []$      $1: 2: [ [] ]$      $[1, 2]: [ [] ]$      $[1: [2], []]$
- ☒ Exactamente dos
  - ☐ Exactamente tres
  - ☐ Exactamente cuatro
14. Considérense las expresiones
- $[[1], [2]]$      $[1]: [[2]]: []$      $[1]: [ [2] ]$      $[1]: [2, []]$      $[1, 2]: []$
- ¿Cuál de las siguientes afirmaciones es cierta?
- ☐ La primera, la tercera y al menos otra más son sintácticamente equivalentes entre sí
  - ☐ La segunda, la cuarta y al menos otra más son sintácticamente equivalentes entre sí
  - ☒ Las dos anteriores son falsas.
15. Considérense las expresiones
- $[[1, 2]]$      $([1]: [[2]]) : []$      $(1: [2]) : []$      $[1, 2]: []$      $[1]: [2]: []$
- ¿Cuál de las siguientes afirmaciones es cierta?
- ☒ La primera, la tercera y al menos otra más son sintácticamente equivalentes entre sí
  - ☐ La segunda, la cuarta y al menos otra más son sintácticamente equivalentes entre sí
  - ☐ Las dos anteriores son falsas.
16. ¿Cuántas de las siguientes expresiones son sintácticamente equivalentes a  $[[1, 2], [1]]$ ?
- $1: 2: [], 1: []$      $[1: [2], [1]]$      $[1, 2]: [1]: []$      $(1: 2: []): [[1]]$
- ☐ Exactamente dos
  - ☐ Exactamente tres
  - ☒ Exactamente cuatro
17. Considérense las expresiones (solo difieren en los paréntesis):
- $e_1 = (f ((x y) y)) (f 0)$   
 $e_2 = f (x y y) (f 0)$   
 $e_3 = f (x y y) f 0$
- ☐  $e_1 \equiv e_2 \equiv e_3$
  - ☐  $e_1 \not\equiv e_2 \equiv e_3$
  - ☒  $e_1 \equiv e_2 \not\equiv e_3$

8/8

8/8

18. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ (f \ x \ (y^2)) \ y \\ e_2 &= f \ ((f \ x) \ ((^ y \ 2)) \ y \\ e_3 &= f \ (f \ x \ (y^{\wedge} \ 2)) \ y \end{aligned}$$

- ☐  $e_1 \equiv e_2 \equiv e_3$   
☒  $e_1 \equiv e_2 \not\equiv e_3$   
☐  $e_1 \not\equiv e_2 \not\equiv e_3$

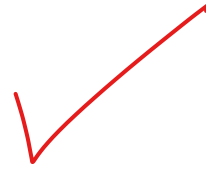


19. Considérense las expresiones (solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ (z \ (y \ x)) \ ((z \ 0) \ x) \\ e_2 &= (f \ (z \ (y \ x))) \ (z \ 0 \ x) \\ e_3 &= f \ z \ (y \ x) \ (z \ 0 \ x) \end{aligned}$$

Entonces:

- ☐  $e_1 \equiv e_2 \equiv e_3$   
☒  $e_1 \equiv e_2 \not\equiv e_3$   
☐  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$



20. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ x \ (g \ x, y/2) \\ e_2 &= f \ x \ (g \ x) \ (y/2) \\ e_3 &= (f \ x) \ (g \ x, (/y) \ 2) \end{aligned}$$

- ☐  $e_1 \equiv e_2 \equiv e_3$   
☒  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$   
☐  $e_1 \equiv e_3 \not\equiv e_2$



21. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ x \ (g \ (x+1) \ y) \\ e_2 &= (f \ x) \ (g \ (((+) \ x) \ 1) \ y) \\ e_3 &= (f \ x) \ (g \ (x+1)) \ y \end{aligned}$$

- ☒  $e_1 \equiv e_2 \not\equiv e_3$   
☐  $e_1 \not\equiv e_2 \equiv e_3$   
☐  $e_1 \equiv e_2 \equiv e_3$



22. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ x \ (g \ x, y+1) \\ e_2 &= f \ x \ (g \ x) \ (y+1) \\ e_3 &= (f \ x) \ (g \ x, (+) \ y \ 1) \end{aligned}$$

- ☐  $e_1 \equiv e_2 \equiv e_3$   
☐  $e_1 \not\equiv e_2 \not\equiv e_3$   
☒  $e_1 \equiv e_3 \not\equiv e_2$



23. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ x \ g \ (x+1) \ y \\ e_2 &= (f \ x) \ (g \ (x+1) \ y) \\ e_3 &= (f \ x) \ g \ ((+) \ x \ 1) \ y \end{aligned}$$

- ☐  $e_1 \equiv e_2 \not\equiv e_3$   
☒  $e_1 \equiv e_3 \not\equiv e_2$   
☐ Las dos anteriores son falsas.



24. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned} e_1 &= f \ x \ 1 \ (x + y) \\ e_2 &= (f \ x \ 1) \ (x + y) \\ e_3 &= f \ x \ 1 \ ((+) \ x \ y) \end{aligned}$$

- ☐  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$

7/7

7/7

- ☐  $e_1 \equiv e_3 \not\equiv e_2$   
☒  $e_1 \equiv e_2 \equiv e_3$

25. Suponiendo la declaración `infixr 9 !`, considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned}
 e_1 &= ((! \ g) \ f) \ ! \ ((h \ !) \ i) \ ! \ j \\
 e_2 &= ((!) \ (f \ ! \ g)) \ ((h \ ! \ i) \ ! \ j) \\
 e_3 &= (!) \ ((!) \ f \ g) \ ((!) \ ((!) \ h \ i) \ j)
 \end{aligned}$$

- ☒  $e_1 \equiv e_2 \equiv e_3$   
☐  $e_1 \equiv e_2 \not\equiv e_3$   
☐  $e_1 \not\equiv e_2 \equiv e_3$

26. Considérense las expresiones:

$$\begin{aligned}
 e_1 &= f \ x \ (y-1):z \\
 e_2 &= ((:) \ f) \ x \ ((-) \ y \ 1) \ z \\
 e_3 &= (: \ z) \ ((f \ x) \ ((-) \ y \ 1))
 \end{aligned}$$

- ☐  $e_1 \equiv e_2 \equiv e_3$   
☐  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$   
☒  $e_1 \equiv e_3 \not\equiv e_2$

27. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned}
 e_1 &= f \ ((g \ x) \ (x \ y)) \ x \\
 e_2 &= f \ (g \ x) \ (x \ y) \ x \\
 e_3 &= (f \ (g \ x \ (x \ y))) \ x
 \end{aligned}$$

- ☐  $e_1 \equiv e_2 \not\equiv e_3$   
☒  $e_1 \equiv e_3 \not\equiv e_2$   
☐  $e_1 \equiv e_2 \equiv e_3$

28. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned}
 e_1 &= f \ x \ y + z \ 4 \\
 e_2 &= f \ x \ ((+) \ y \ (z \ 4)) \\
 e_3 &= (+) \ ((f \ x) \ y) \ (z \ 4)
 \end{aligned}$$

- ☐  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$   
☐  $e_1 \equiv e_2 \equiv e_3$   
☒  $e_1 \equiv e_3 \not\equiv e_2$

29. Considérense las expresiones (que solo difieren en los paréntesis):

$$\begin{aligned}
 e_1 &= f \ x + g \ z \ 4 \\
 e_2 &= f \ x \ ((+) \ g \ (z \ 4)) \\
 e_3 &= (+) \ (f \ x \ (g \ z \ 4))
 \end{aligned}$$

- ☒  $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$   
☐  $e_1 \equiv e_2 \equiv e_3$   
☐  $e_1 \equiv e_3 \not\equiv e_2$

30. Considérense las expresiones de tipo (solo difieren en los paréntesis):

$$\begin{aligned}
 \tau_1 &= (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow (a \rightarrow a) \\
 \tau_2 &= (a \rightarrow a) \rightarrow (a \rightarrow a \rightarrow a \rightarrow a) \\
 \tau_3 &= (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a \rightarrow a
 \end{aligned}$$

- ☐  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$   
☒  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$   
☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$

31. Considérense las expresiones de tipo (solo difieren en los paréntesis):

4/6

$$\tau_1 = ((b \rightarrow a) \rightarrow a) \rightarrow ((a \rightarrow b) \rightarrow (b \rightarrow b))$$

$$\tau_2 = (b \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow b) \rightarrow b \rightarrow b$$

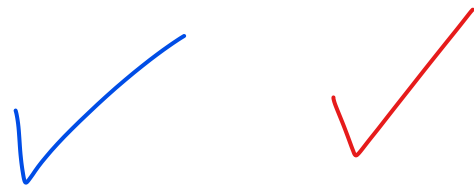
$$\tau_3 = (b \rightarrow a \rightarrow a) \rightarrow (a \rightarrow b \rightarrow b \rightarrow b)$$

Entonces:

☐  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$

☒  $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$



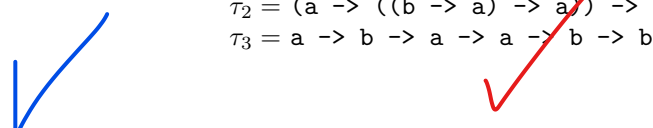
32. Considérense las expresiones de tipo (que solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= (a \rightarrow (b \rightarrow a) \rightarrow a) \rightarrow b \rightarrow b \\ \tau_2 &= (a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow (b \rightarrow b) \\ \tau_3 &= a \rightarrow b \rightarrow a \rightarrow a \rightarrow b \rightarrow b \end{aligned}$$

☒  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☐  $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$

☐  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$



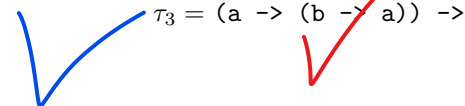
33. Considérense las expresiones de tipo (solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= (a \rightarrow b \rightarrow a) \rightarrow (a \rightarrow b \rightarrow b) \\ \tau_2 &= (a \rightarrow b \rightarrow a) \rightarrow a \rightarrow (b \rightarrow b) \\ \tau_3 &= (a \rightarrow (b \rightarrow a)) \rightarrow a \rightarrow b \rightarrow b \end{aligned}$$

☐  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☐  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$

☒  $\tau_1 \equiv \tau_2 \equiv \tau_3$



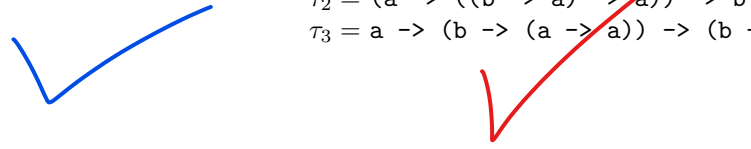
34. Considérense las expresiones de tipo (solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= a \rightarrow (b \rightarrow a \rightarrow a) \rightarrow b \rightarrow b \\ \tau_2 &= (a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow b \rightarrow b \\ \tau_3 &= a \rightarrow (b \rightarrow (a \rightarrow a)) \rightarrow (b \rightarrow b) \end{aligned}$$

☐  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☒  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$



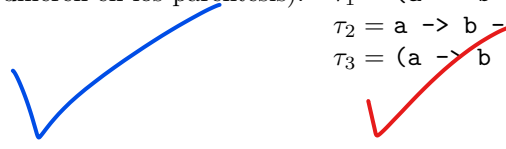
35. Considérense las expresiones de tipo (solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= (a \rightarrow b \rightarrow a \rightarrow a) \rightarrow b \rightarrow b \\ \tau_2 &= a \rightarrow b \rightarrow a \rightarrow a \rightarrow b \rightarrow b \\ \tau_3 &= (a \rightarrow b \rightarrow (a \rightarrow a)) \rightarrow (b \rightarrow b) \end{aligned}$$

☐  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☒  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$



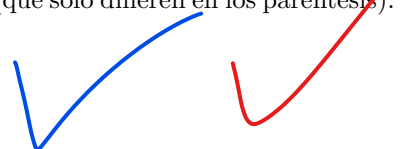
36. Considérense las expresiones de tipo (que solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= a \rightarrow a \rightarrow (a \rightarrow a) \rightarrow (b \rightarrow b) \\ \tau_2 &= a \rightarrow (a \rightarrow ((a \rightarrow a) \rightarrow b \rightarrow b)) \\ \tau_3 &= (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow b \rightarrow b \end{aligned}$$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$

☐  $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$

☒  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$



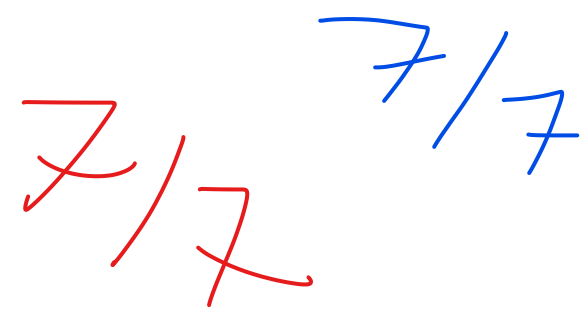
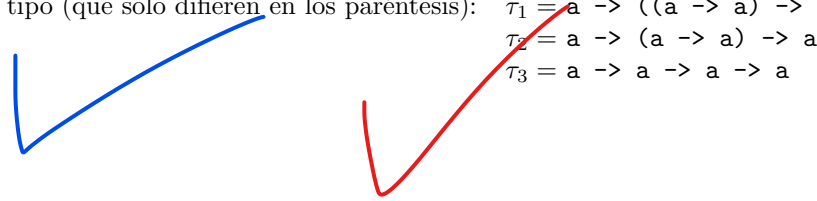
37. Considérense las expresiones de tipo (que solo difieren en los paréntesis):

$$\begin{aligned} \tau_1 &= a \rightarrow ((a \rightarrow a) \rightarrow a) \\ \tau_2 &= a \rightarrow (a \rightarrow a) \rightarrow a \\ \tau_3 &= a \rightarrow a \rightarrow a \rightarrow a \end{aligned}$$

☒  $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

☐  $\tau_1 \equiv \tau_3 \not\equiv \tau_2$

☐  $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$



38. Considérense las expresiones de tipo (que solo difieren en los paréntesis):  $\tau_1 = (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow (a \rightarrow a)$

$\tau_2 = a \rightarrow a \rightarrow (a \rightarrow a) \rightarrow (a \rightarrow a)$

$\tau_3 = (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a \rightarrow a$

☐  $\tau_1 \neq \tau_2 \neq \tau_3 \neq \tau_1$

☒  $\tau_1 \equiv \tau_3 \neq \tau_2$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$

39. Considérense las expresiones de tipo (solo difieren en los paréntesis):  $\tau_1 = (b \rightarrow a \rightarrow a) \rightarrow (a \rightarrow b) \rightarrow b$

$\tau_2 = (b \rightarrow (a \rightarrow a)) \rightarrow ((a \rightarrow b) \rightarrow b)$

$\tau_3 = (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow b \rightarrow b$

☐  $\tau_1 \equiv \tau_2 \equiv \tau_3$

☐  $\tau_1 \equiv \tau_3 \neq \tau_2$

☒  $\tau_1 \equiv \tau_2 \neq \tau_3$

40. Considérense las expresiones de tipo (solo difieren en los paréntesis):  $\tau_1 = a \rightarrow b \rightarrow (c \rightarrow d)$

$\tau_2 = a \rightarrow (b \rightarrow c \rightarrow d)$

$\tau_3 = a \rightarrow b \rightarrow c \rightarrow d$

☒  $\tau_1 \equiv \tau_2 \equiv \tau_3$

☐  $\tau_1 \equiv \tau_3 \neq \tau_2$

☐  $\tau_1 \neq \tau_2 \neq \tau_3$

41. Considérense las expresiones de tipo (que solo difieren en los paréntesis):  $\tau_1 = (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow (a \rightarrow a)$

$\tau_2 = (a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a \rightarrow a$

$\tau_3 = (a \rightarrow a) \rightarrow ((a \rightarrow a) \rightarrow (a \rightarrow a))$

☐  $\tau_1 \neq \tau_2 \neq \tau_3 \neq \tau_1$

☐  $\tau_1 \equiv \tau_3 \neq \tau_2$

☒  $\tau_1 \equiv \tau_2 \equiv \tau_3$

42. Considérense las expresiones de tipo (que solo difieren en los paréntesis):  $\tau_1 = (a \rightarrow a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow (a \rightarrow a)$

$\tau_2 = (a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a) \rightarrow a \rightarrow a$

$\tau_3 = (a \rightarrow a \rightarrow a) \rightarrow ((a \rightarrow a) \rightarrow a \rightarrow a)$

☐  $\tau_1 \neq \tau_2 \neq \tau_3 \neq \tau_1$

☐  $\tau_1 \equiv \tau_3 \neq \tau_2$

☒  $\tau_1 \equiv \tau_2 \equiv \tau_3$

43. Considérense las expresiones siguientes:

$e_1 \equiv (\text{let } x=5 \text{ in } x+x) + 3$

$e_2 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*y$

$e_3 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*y*x$

$e_4 \equiv \text{let } y=x+x \text{ in let } x=2 \text{ in } y*y*x$

$e_5 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y$

$e_6 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y*x$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

☐ Exactamente tres de ellas

☒ Exactamente dos de ellas

☐ Todas están correctamente formadas

44. Considérense las expresiones siguientes:

$e_1 \equiv (\text{let } x=5 \text{ in } x+x) + 5$

$e_2 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*x$

$e_3 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } x$

$e_4 \equiv \text{let } x=y \text{ in let } x=2 \text{ in } y*y*x$

$e_5 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y$

$e_6 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y*x$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

- ☐ Exactamente una de ellas
- ☒ Exactamente dos de ellas
- ☐ Exactamente tres de ellas

45. Considérense las expresiones siguientes:

$e_1 \equiv (\text{let } x=5 \text{ in } x+x) + x$        $e_2 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*y*x$   
 $e_3 \equiv \text{let } y=x+x \text{ in let } x=2 \text{ in } y*y*x$        $e_4 \equiv \text{let } \{y=x+x; x=2\} \text{ in } y*y*x$   
 $e_5 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y*x$        $e_6 \equiv \text{let } y=(\text{let } x=2 \text{ in } 3) \text{ in } y*y$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

- ☐ Exactamente dos de ellas
- ☒ Exactamente tres de ellas
- ☐ Exactamente cuatro de ellas

46. Considérense las expresiones siguientes:

$e_1 \equiv (\text{let } x=5 \text{ in } x+x) + 5$        $e_2 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*x$   
 $e_3 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } x$        $e_4 \equiv \text{let } x=y \text{ in let } x=2 \text{ in } y*y*x$   
 $e_5 \equiv \text{let } y=(\text{let } x=x \text{ in } x+x) \text{ in } y*y$        $e_6 \equiv \text{let } y=(\text{let } x=2 \text{ in } x+x) \text{ in } y*y*x$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

- ☐ Exactamente una de ellas
- ☒ Exactamente dos de ellas
- ☐ Exactamente tres de ellas

47. Considérense las expresiones siguientes:

$e_1 \equiv (\text{let } x=5 \text{ in } x+x) + (\text{let } x=3 \text{ in } 2*x)$        $e_2 \equiv \text{let } y=x+x \text{ in let } x=2 \text{ in } y*y*x$   
 $e_3 \equiv \text{let } x=2 \text{ in let } y=x+x \text{ in } y*y*x$        $e_4 \equiv \text{let } \{y=x+x; x=2\} \text{ in } y*y*x$   
 $e_5 \equiv [i \mid i < -[1..j], j < -[0..100], \text{mod } j \ 3 == 0]$        $e_6 \equiv [i \mid j < -[0..100], i < -[1..j], \text{mod } j \ 3 == 0]$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

- ☐ Exactamente una de ellas
- ☒ Exactamente dos de ellas
- ☐ Exactamente tres de ellas

48. Considérense las expresiones siguientes:

$e_1 \equiv \text{let } x=1:x \text{ in head } x$        $e_2 \equiv (\backslash x \rightarrow (\backslash y \rightarrow x+y)) \ x$   
 $e_3 \equiv \text{let } x=[1,2,3] \text{ in let } y=x!!2 \text{ in } y*\text{last } x$        $e_4 \equiv \text{let } \{y=2*x; x=5\} \text{ in } y*y*x$   
 $e_5 \equiv [i+j \mid i < -[1..j], j < -[0..100], \text{mod } j \ i == 0]$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

- ☒ Exactamente dos de ellas
- ☐ Exactamente en tres de ellas
- ☐ Exactamente en cuatro de ellas

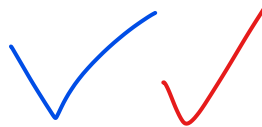
49. Considérense las expresiones siguientes:

$e_1 \equiv \backslash x \rightarrow ((\backslash y \rightarrow x) \ x)$        $e_2 \equiv \backslash x \rightarrow ((\backslash y \rightarrow x+y) \ y)$   
 $e_3 \equiv \text{let } y=[1,2,3] \text{ in let } x=y!!1 \text{ in } x*\text{head } y$        $e_4 \equiv \text{let } \{y=2*x; x=5\} \text{ in } y*y*x$   
 $e_5 \equiv [i+j \mid i < -[1..100], j < -[0..i], \text{mod } j \ i == 0]$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

4/6  
6/6

- ☒ Exactamente una de ellas
- ☐ Tres o más de ellas
- ☐ Las dos anteriores son falsas.

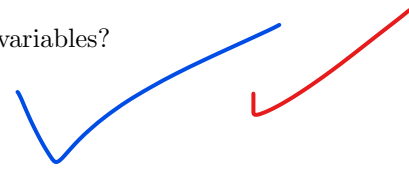


50. Considérense las expresiones siguientes:

$e_1 \equiv \lambda x \rightarrow ((\lambda x y \rightarrow x+y) x y)$        $e_2 \equiv \lambda x \rightarrow ((\lambda x y \rightarrow x+y) x x)$   
 $e_3 \equiv \text{let } y = (\text{let } x = 1 \text{ in } x+x) \text{ in } x+y$        $e_4 \equiv \text{let } y = (\text{let } x = 1 \text{ in } x+x) \text{ in } y+y$   
 $e_5 \equiv [j \mid i \leftarrow [1..100], j \leftarrow [0..i]]$

¿Cuántas de ellas son sintácticamente erróneas por problemas de ámbito de variables?

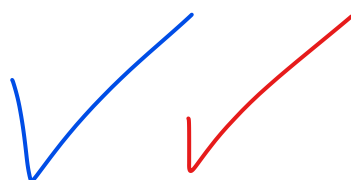
- ☐ Exactamente una de ellas
- ☒ Exactamente dos de ellas
- ☐ Las dos anteriores son falsas.



51. En el siguiente fragmento de código

```
data T a = A | (Int, T a)
f x (y:xs) = y
```

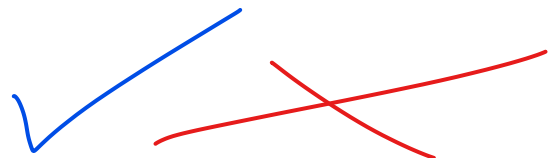
- ☒ La definición de T contiene algún error sintáctico, pero la de f no.
- ☐ La definición de f contiene algún error sintáctico, pero la de T no.
- ☐ Las dos anteriores son falsas.



52. En el siguiente fragmento de código

```
data T a = A | (Int, T a)
f x (x:xs) = True
f x (y:xs) = f x xs
```

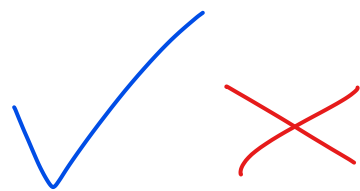
- ☐ La definición de T contiene algún error, pero la de f no.
- ☐ La definición de f contiene algún error, pero la de T no.
- ☒ Las dos anteriores son falsas.



53. En el siguiente fragmento de código

```
data T a = A | B (Int, T a)
f x (x:xs) = xs
```

- ☐ La definición del tipo T contiene algún error sintáctico, pero la de f no.
- ☒ La definición de la función f contiene algún error sintáctico, pero la de T no.
- ☐ Las dos anteriores son falsas.



54. Supongamos que  $1 :: \text{Int}$ ,  $(+) :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ , y considérese la función f definida por las dos reglas siguientes:

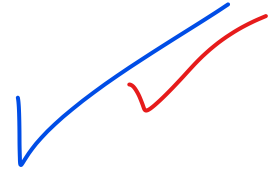
```
f True x y = (x,y)
f False y x = (y,x+1)
```

- ☒ El tipo que se infiere para f es  $\text{Bool} \rightarrow a \rightarrow \text{Int} \rightarrow (a, \text{Int})$
- ☐ El tipo que se infiere para f es  $\text{Bool} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow (\text{Int}, \text{Int})$
- ☐ f está mal tipada.



55. Considérese la función definida por  $f x y = y x x$ . El tipo de f es:

- ☒  $a \rightarrow (a \rightarrow a \rightarrow b) \rightarrow b$
- ☐  $a \rightarrow (a \rightarrow a \rightarrow a) \rightarrow a$
- ☐ No está bien tipada



Handwritten scores: 4/6 (red) and 6/6 (blue).



56. Sea  $f$  definida por  $f\ g\ x = x\ (g\ \text{True})\ g$ . El tipo de  $f$  es:

- ☒  $\forall a, b. (\text{Bool} \rightarrow a) \rightarrow (a \rightarrow (\text{Bool} \rightarrow a) \rightarrow b) \rightarrow b$
- ☐  $\forall a. (\text{Bool} \rightarrow a) \rightarrow (a \rightarrow (\text{Bool} \rightarrow a) \rightarrow a) \rightarrow a$
- ☐ Está mal tipada

57. Sea  $f$  definida por  $f\ g\ x = x\ g\ g$ . El tipo de  $f$  es:

- ☒  $\forall a, b. a \rightarrow (a \rightarrow a \rightarrow b) \rightarrow b$
- ☐  $\forall a, b. (a \rightarrow a \rightarrow b) \rightarrow a \rightarrow b$
- ☐ Está mal tipada

58. Sea  $f$  definida por  $f\ g\ x = x\ g\ g$ . El tipo de  $f$  es:

- ☐ Está mal tipada
- ☐  $\forall a. (a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a$
- ☒  $\forall a, b. a \rightarrow (a \rightarrow a \rightarrow b) \rightarrow b$

59. Considérese la función definida por  $f\ x\ y = x\ (y\ x)$ . El tipo de  $f$  es:

- ☒  $(a \rightarrow b) \rightarrow ((a \rightarrow b) \rightarrow a) \rightarrow b$
- ☐  $(a \rightarrow b \rightarrow a) \rightarrow (b \rightarrow a) \rightarrow a$
- ☐ No está bien tipada

60. Considérese la función definida por  $f\ x\ y = y\ (x\ x)$ . El tipo de  $f$  es:

- ☐  $(a \rightarrow a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a$
- ☐  $(a \rightarrow b \rightarrow a) \rightarrow (b \rightarrow a) \rightarrow a$
- ☒ No está bien tipada

61. Considérese la función definida por  $f\ x\ y = x\ x\ y$ . El tipo de  $f$  es:

- ☐  $(a \rightarrow a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a$
- ☐  $(a \rightarrow b \rightarrow a) \rightarrow (b \rightarrow a) \rightarrow a$
- ☒ No está bien tipada

62. Considérese la función definida por  $f\ x\ y = x\ (y\ y)$ . El tipo de  $f$  es:

- ☐  $(a \rightarrow b) \rightarrow (a \rightarrow a) \rightarrow b$
- ☐  $(a \rightarrow a) \rightarrow (a \rightarrow a) \rightarrow a$
- ☒ No está bien tipada

63. Considérese la función definida por  $f\ g = g\ (f\ g)$ . El tipo de  $f$  es:

- ☐  $a \rightarrow a \rightarrow a$
- ☒  $(a \rightarrow a) \rightarrow a$
- ☐ No está bien tipada

64. Considérese la función definida por  $f\ g = g\ (g\ f)$ . El tipo de  $f$  es:

- ☐  $a \rightarrow a \rightarrow a$
- ☐  $(a \rightarrow a) \rightarrow a$
- ☒ No está bien tipada

65. Sea  $f$  definida por  $f\ g\ x = x\ (x\ g)$ . El tipo de  $f$  es:

- ☐  $\forall a. a \rightarrow (a \rightarrow b) \rightarrow b$
- ☒  $\forall a. a \rightarrow (a \rightarrow a) \rightarrow a$
- ☐ Está mal tipada

10/10 10/10

66. Sea  $f$  definida por  $f\ x\ g = x\ x\ g$ . El tipo de  $f$  es:

- ☐  $\forall a.a \rightarrow (a \rightarrow a \rightarrow a) \rightarrow a$
- ☐  $\forall a,b.a \rightarrow (b \rightarrow a \rightarrow b) \rightarrow b$
- ☒ Está mal tipada

67. Sea  $f$  definida por  $f\ x\ y\ z = x\ (y\ z)$ . El tipo de  $f$  es:

- ☐  $(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow a \rightarrow c$
- ☒  $(a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$
- ☐  $f$  está mal tipada

68. Sea  $f$  definida por  $f\ x\ g = x\ (x\ (g\ \text{True}))$ . El tipo de  $f$  es:

- ☐  $\forall a,b.(a \rightarrow b) \rightarrow (\text{Bool} \rightarrow a) \rightarrow b$
- ☒  $\forall a.(a \rightarrow a) \rightarrow (\text{Bool} \rightarrow a) \rightarrow a$
- ☐ Ninguno,  $f$  está mal tipada

69. Sea  $f$  definida por  $f\ x\ y = (x\ y).(x\ y)$ . El tipo de  $f$  es:

- ☐  $(a \rightarrow a) \rightarrow (a \rightarrow a)$
- ☐  $(a \rightarrow b) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow b)$
- ☒  $(a \rightarrow b \rightarrow b) \rightarrow a \rightarrow b \rightarrow b$

70. Sea  $f$  definida por  $f\ x\ y = x\ (x\ y)$ . El tipo de  $f$  es:

- ☒  $(a \rightarrow a) \rightarrow (a \rightarrow a)$
- ☐  $(a \rightarrow b) \rightarrow a \rightarrow b$
- ☐  $a \rightarrow b \rightarrow a$

71. Considérese la función definida por  $f\ g = g\ (f\ f)$ . El tipo de  $f$  es:

- ☐  $a \rightarrow a \rightarrow a$
- ☐  $(a \rightarrow a) \rightarrow a$
- ☒ No está bien tipada

72. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función  $f$  definida por

$f\ x\ y\ z = \text{if } x \leq y \text{ then } z + 1 \text{ else } z$  será:

- ☐  $f :: \text{Num } a \Rightarrow a \rightarrow a \rightarrow a \rightarrow a$
- ☒  $f :: (\text{Ord } a, \text{Num } b) \Rightarrow a \rightarrow a \rightarrow b \rightarrow b$
- ☐  $f :: (\text{Ord } a, \text{Num } a) \Rightarrow a \rightarrow a \rightarrow a \rightarrow a$

73. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función  $f$  definida por

$f\ x\ y\ z = \text{if } x \leq y\ z \text{ then } z + 2 \text{ else } z$  será:

- ☐  $f :: \text{Num } a \Rightarrow a \rightarrow a \rightarrow a \rightarrow a$
- ☐  $f :: (\text{Ord } a, \text{Num } b) \Rightarrow a \rightarrow b \rightarrow b \rightarrow a$
- ☒  $f :: (\text{Ord } a, \text{Num } b) \Rightarrow a \rightarrow (b \rightarrow a) \rightarrow b \rightarrow b$

74. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función  $f$  definida por

$f\ x\ y\ z = \text{if } x \text{ then } y \leq z \text{ else } x$  será:

- ☐  $f :: (\text{Ord } a, \text{Bool } a) \Rightarrow a \rightarrow a \rightarrow a \rightarrow a$
- ☒  $f :: \text{Ord } a \Rightarrow \text{Bool} \rightarrow a \rightarrow a \rightarrow \text{Bool}$
- ☐ Esa definición dará un error de tipos

9/9 9/9

75. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y z = if x <= y then z+1 else x
```
- será:
- ☒ `f :: (Num a, Ord a) => a -> a -> a -> a`
  - ☐ `f :: (Ord a, Num b) => a -> a -> b -> b`
  - ☐ `f :: (Ord a, Num b) => a -> a -> b -> a`
76. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y z = if x <= y then z else not x
```
- será:
- ☐ `f :: (Ord a, Bool a) => a -> a -> a -> a`
  - ☐ `f :: Ord a => Bool -> a -> a -> Bool`
  - ☒ `Bool -> Bool -> Bool -> Bool`
77. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y z = z (x <= y+1)
```
- será:
- ☒ `f :: (Num a, Ord a) => a -> a -> (Bool -> b) -> b`
  - ☐ `f :: (Ord a, Num b, Ord b) => a -> b -> (Bool -> c) -> c`
  - ☐ Dará un error de tipos
78. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y z = if x <= y then z + x else z
```
- será:
- ☐ `f :: Num a => a -> a -> a -> a`
  - ☐ `f :: (Ord a, Num b) => a -> a -> b -> b`
  - ☒ `f :: (Ord a, Num a) => a -> a -> a -> a`
79. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y = if x <= 0 then y + 1 else y
```
- será:
- ☐ `f :: Num a => a -> a -> a`
  - ☒ `f :: (Num a, Ord a, Num b) => a -> b -> b`
  - ☐ `f :: (Ord a, Num a) => a -> a -> a`
80. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y = if x == y+1 then y else y+1
```
- será:
- ☐ `f :: Eq (Num a) => a -> a -> Num a`
  - ☒ `f :: (Eq a, Num a) => a -> a -> a`
  - ☐ `f :: (Eq a, Num b) => a -> b -> b`
81. El tipo que inferirá Haskell, teniendo en cuenta clases de tipos, para una función `f` definida por
- ```
f x y z = if not x then z <= y else x
```
- será:
- ☐ `f :: Ord Bool => Bool -> Bool -> Bool -> Bool`
  - ☐ `f :: Bool -> Bool -> Bool -> Bool`
  - ☒ `f :: Ord a => Bool -> a -> a -> Bool`
82. ¿Cuál de los siguientes tipos para `f` hacen que la expresión `(curry f 0).(||, True)` esté bien tipada?
- ☒ `f :: (Int, Bool) -> Int`
  - ☐ `f :: Int -> Bool -> Int`
  - ☐ Esa expresión está mal tipada, sea cual sea el tipo de `f`
83. Sea `f` de tipo  $\tau \rightarrow \tau$ , y `unaLista` de tipo  $[\tau]$ . El tipo de la expresión `map (take 2) (map (iterate f) unaLista)` es:

8/8

8/8

- ☐  $[\tau]$
- ☒  $[[\tau]]$
- ☐ Esa expresión está mal tipada

84. Sea  $f$  de tipo  $\tau \rightarrow \tau$ , y `unaLista` de tipo  $[\tau]$ . El tipo de la expresión `map (iterate f) (map (take 2) unaLista)` es:

- ☐  $[\tau]$
- ☒  $[[\tau]]$ , si es que  $\tau$  es de la forma  $[\tau']$
- ☐ Esa expresión está en cualquier caso mal tipada

85. ¿Cuál de los siguientes tipos para  $f$  hace que la expresión `(|| True).(uncurry f)` esté bien tipada?

- ☐  $f :: (\text{Int}, \text{Int}) \rightarrow \text{Bool}$
- ☒  $f :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}$
- ☐ Esa expresión está mal tipada, sea cual sea el tipo de  $f$

86. ¿Cuál de los siguientes tipos para la expresión  $e$  hace que la expresión `zipWith filter [(> 0), (< 0)] e` esté bien tipada?

- ☐  $[\text{Int}]$
- ☒  $[[\text{Int}]]$
- ☐  $[(\text{Int}, \text{Int})]$

87. ¿Cuál de los siguientes tipos para la expresión  $e$  hace que la expresión `zipWith filter e [[1..4], [-2..3]]` esté bien tipada?

- ☐  $\text{Int} \rightarrow \text{Bool}$
- ☒  $[\text{Int} \rightarrow \text{Bool}]$
- ☐ Las dos anteriores son falsas.

88. ¿Cuál de los siguientes tipos para la expresión  $e$  hace que la expresión `takeWhile e (zip (iterate not True) [0..10])` esté bien tipada?

- ☐  $\text{Int} \rightarrow \text{Int}$
- ☐  $\text{Int} \rightarrow \text{Bool}$
- ☒  $(\text{Bool}, \text{Int}) \rightarrow \text{Bool}$

89. ¿Cuál de los siguientes tipos para la expresión  $e$  hace que la expresión `zipWith e (iterate not True) (iterate (+ 1) 0)` esté bien tipada?

- ☐  $[\text{Bool}] \rightarrow [\text{Int}] \rightarrow [\text{Bool}]$
- ☐  $[\text{Bool}] \rightarrow [\text{Int}] \rightarrow [(\text{Bool}, \text{Int})]$
- ☒  $\text{Bool} \rightarrow \text{Int} \rightarrow \text{Char}$

90. ¿Cuál de los siguientes tipos para la expresión  $e$  hace que la expresión `(head.e) (zip (iterate not True) [1..5])` esté bien tipada?

- ☒  $[(\text{Bool}, \text{Int})] \rightarrow [\text{Int}]$
- ☐  $(\text{Bool}, \text{Int}) \rightarrow \text{Int}$
- ☐  $(\text{Bool}, \text{Int}) \rightarrow [\text{Int}]$

91. ¿Cuántas de las siguientes definiciones de tipos (independientes unas de otras) son correctas?

```
data Tip = A | C Int Tip | (Int, Int, Tip)
data Tap = A | C Int Tap | D Int Int Tap
data Top = A | C a Top | D a b Top
```

- ☐ Las tres
- ☐ Ninguna de las tres
- ☒ Una de las tres

92. ¿Cuántas de las siguientes definiciones de tipos (independientes unas de otras) son correctas?

data Tip = A | C Int Tip | C (Int, Int, Tip)  
 data Tap = A | C Int Tap | D (Int, Int, Tap)  
 data Top a = A | C a | D a a

- ☐ Una de las tres
- ☒ Dos de las tres
- ☐ Las tres

93. En lo que sigue,  $\leq$  indica el orden estándar (el obtenido por deriving Ord) para tipos con constructoras de datos. Considérense las afirmaciones siguientes: (i)  $[] \leq [True] \leq [False, True]$  (ii)  $[] \leq [False, True] \leq [True, False]$ . Entonces, teniendo en cuenta que para los booleanos  $False \leq True$ , se tiene:

- ☐ (i) es cierta pero (ii) no
- ☒ (ii) es cierta pero (i) no
- ☐ Las dos son falsas

94. Considérese la definición del tipo data T = A | B | C T T deriving (Eq, Ord). ¿Cuál de las siguientes afirmaciones es cierta?

- ☒ A <= B && B <= C A A se evalúa a True
- ☐ A <= B && B <= C A A se evalúa a False
- ☐ C loop loop == C loop loop se evalúa a True, donde loop está definido por loop = loop

95. Considérese la definición del tipo data T = A | B | C T T deriving (Eq, Ord) y la función mal = head []. ¿Cuál de las siguientes afirmaciones es cierta?

- ☐ C mal A <= C mal B se evalúa a True
- ☐ C A mal == C B mal se evalúa a True
- ☒ A <= C mal mal && B <= C mal mal se evalúa a True

96. Considérese la definición del tipo data T = A | B | C T T deriving (Eq, Ord) y la función loop = loop. ¿Cuál de las siguientes afirmaciones es cierta?

- ☒ A <= B && B <= C loop loop se evalúa a True
- ☐ A <= B && B <= C loop loop se evalúa a False
- ☐ C loop loop == C loop loop se evalúa a True

97. Considérese la definición del tipo data T = A | B | C T T deriving (Eq, Ord) y la función loop = loop. ¿Cuál de las siguientes afirmaciones es falsa?

- ☒ loop <= B && B <= C loop loop se evalúa a True
- ☐ A <= C loop loop && B <= C loop loop se evalúa a True
- ☐ C A loop == C B loop se evalúa a False

98. Considérense la definición del tipo data T = A | B | C T T deriving (Eq, Ord) y la función mal = head []. ¿Cuál de las siguientes afirmaciones es cierta?

- ☐ C mal A <= C mal B se evalúa a True
- ☒ A <= C mal mal && B <= C mal mal se evalúa a True
- ☐ C A mal == C B mal se evalúa a True

2/8

7/8

99. Considérese la definición del tipo `data T = A | B | C T T deriving (Eq,Ord)` y la función `mal = head []` y considérense las siguientes afirmaciones:

- (i) `C mal A <= C mal B` se evalúa a `True`
- (ii) `C A mal == C B mal` se evalúa a `True`
- (iii) `A <= C mal mal && B <= C mal mal` se evalúa a `True`

- ☒ Exactamente una es cierta
- ☐ Exactamente dos son ciertas
- ☐ Las dos anteriores son falsas.

100. En lo que sigue,  $\leq$  indica el orden estándar (el obtenido por `deriving Ord`) para tipos con constructoras de datos. Considérense las afirmaciones siguientes: (i) `[] ≤ [0] ≤ [0,1]` (ii) `[] ≤ [0,1] ≤ [1,0]`. Entonces:

- ☐ (i) es cierta pero (ii) no
- ☐ (ii) es cierta pero (i) no
- ☒ Las dos son ciertas

101. Considérense las declaraciones de clase e instancia

|                                                                                  |                                                             |                                                       |
|----------------------------------------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------|
| <pre>class C a where   f, g :: a -&gt; Int   f x = g x + 1   g x = f x - 1</pre> | <pre>instance C Bool where   g True = 0   g False = 1</pre> | <pre>instance C Int where   f x = x   g x = 2*x</pre> |
|----------------------------------------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------|

¿Qué afirmación es correcta?

- ☒ `f False` se evalúa a 2 y `g 1` se evalúa a 2
- ☐ La evaluación de una de las expresiones del caso anterior da un error
- ☐ Las dos anteriores son falsas.

102. Considérese la declaración de clase

```
class C a where
  f, g :: a -> Int
  f x = g x + 1
  g x = f x - 1
```

¿Qué afirmación es correcta?

- ☐ Esa declaración es errónea, porque `f` y `g` no terminarán nunca
- ☒ `{f}` es un conjunto minimal suficiente de métodos de `C`
- ☐ `{f,g}` es un conjunto minimal suficiente de métodos de `C`

103. Considérese la declaración de clase `class C a where f, g :: a -> Int`  
`f x = g x + 1`

¿Qué afirmación es correcta?

- ☐ Esa declaración es errónea, porque `g` no está definida
- ☒ `{g}` es un conjunto minimal suficiente de métodos de `C`
- ☐ `{f}` es un conjunto minimal suficiente de métodos de `C`

104. Considérese la declaración de clase

```
class C a where
  f, g, h :: a -> Int
  f x = g x + 1
  g x = f x - 1
```

¿Qué afirmación es correcta?

- ☐ El sistema dará un error con esta definición
- ☐ Al declarar una instancia de `C` no es obligado definir `h` ni redefinir `f`, `g`
- ☐ Al declarar una instancia de `C` es obligado definir `h` y razonable redefinir `f` o bien `g`

Te salta  
un warning

$s/s^+?$   $4/5 + ?$

105. Considérense la declaraciones de clase e instancia

|                                                                                      |                                                   |                                           |
|--------------------------------------------------------------------------------------|---------------------------------------------------|-------------------------------------------|
| <pre>class C a where   f, g :: Int -&gt; a   f x = g (x + 1)   g x = f (x - 1)</pre> | <pre>instance C Bool where   g x = (x == 0)</pre> | <pre>instance C Int where   f x = x</pre> |
|--------------------------------------------------------------------------------------|---------------------------------------------------|-------------------------------------------|

¿Qué afirmación es correcta?

- ☐ f 0 && g 0 se evalúa a False y f 1 se evalúa a 1
- ☒ La evaluación de exactamente una de las expresiones del caso anterior da un error
- ☐ Las dos anteriores son falsas.

106. Considérense la declaraciones de clase e instancia

|                                                                          |                                                             |                                                       |
|--------------------------------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------|
| <pre>class C a where   f, g :: Int -&gt; a   f x = g 0   g x = f 1</pre> | <pre>instance C Bool where   g 0 = True   g _ = False</pre> | <pre>instance C Int where   f x = x   g x = 2*x</pre> |
|--------------------------------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------|

¿Qué afirmación es falsa?

- ☐ Al intentar evaluar f 0 resulta un error de ambigüedad de tipos
- ☐ not (f 0) se evalúa a False
- ☒ f 0 == f 0 se evalúa a True

107. Considérese la función f definida como f xs = foldr g [] xs where f x y = y++[x]. Entonces:

- ☒ f xs computa la inversa de xs
- ☐ f xs computa la propia lista xs
- ☐ f está mal tipada

108. Considérense las funciones

|                                                                   |                                                                    |
|-------------------------------------------------------------------|--------------------------------------------------------------------|
| <pre>f xs = foldr g [] xs where   g x y = x:filter (/= x) y</pre> | <pre>f' xs = foldl g [] xs where   g y x = x:filter (/= x) y</pre> |
|-------------------------------------------------------------------|--------------------------------------------------------------------|

(y ojo al orden de argumentos en g). Entonces:

- ☐ f xs y f' xs coinciden, para cualquier lista finita xs.
- ☒ Los elementos de f xs y f' xs coinciden, quizás en otro orden, para cualquier lista finita xs.
- ☐ Una de las dos está mal tipada.

109. Considérese la función f definida como f xs = foldl g [] xs where g y x = y++[x]. Entonces:

- ☐ f xs computa la inversa de xs
- ☒ f xs computa la propia lista xs
- ☐ f está mal tipada

110. La evaluación de foldl (\e x -> x:x:e) [] [1,2,3] produce como resultado

- ☐ [1,1,2,2,3,3]
- ☒ [3,3,2,2,1,1]
- ☐ [3,2,1,3,2,1]

111. La evaluación de foldr (\x e -> x:[1..length e]) [0] [1,2,3] produce como resultado

- ☐ [1,2,3,0]
- ☐ [3,1,2,3]
- ☒ [1,1,2,3]

112. La evaluación de foldr (\x y -> x y) 1 [\x -> x\*x, \x -> x-1, (+ 3)] produce como resultado

- ☒ 9
- ☐ [1,0,4]
- ☐ Una lista de funciones

5/8

6/8

113. La evaluación de `foldl (\x y -> y-x ) 0 [1,2,3]` produce como resultado  
☐ -4  
☐ -6  
☒ 2
114. La evaluación de `foldr (\x y -> x-y ) 0 [1,2,3]` produce como resultado  
☐ -4  
☐ -6  
☒ 2
115. La evaluación de `foldr (\x y -> x/y ) 1 [8,4,2]` produce como resultado  
☒ 4  
☐ 2  
☐ 1
116. La evaluación de `foldl (\x y -> y-x ) 1 [1,2,3]` produce como resultado  
☐ -1  
☒ 1  
☐ 0
117. La evaluación de `foldl (\x y -> x-(last y)) 10 [[1,2],[3,4],[5]]` produce como resultado  
☐ 5  
☐ [8,6,5]  
☒ -1  
☒ -1
118. Sea  $\oplus :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$  una función semánticamente asociativa,  $n$  el valor de `foldl  $\oplus$  0 [1,2,3]` y  $m$  el valor de `foldr  $\oplus$  0 [1,2,3]`. Entonces  
☐ Es seguro que  $n = m$   
☐ Es seguro que  $n \neq m$   
☒ Las dos anteriores son falsas.
119. Sea  $n$  el valor de `foldl (\x y -> y) 0 [1,2,3]` y  $m$  el valor de `foldr (\x y -> y) 0 [1,2,3]`. Entonces  
☒  $n > m$   
☐  $n < m$   
☐  $n = m$
120. La evaluación de `foldr (\x y -> y+(x!!0)) 2 [[1,2],[3,4],[5,6]]` produce como resultado  
☒ 11  
☐ [2,3,4,5,6,7]  
☐ Nada porque la expresión está mal tipada
121. La evaluación de `foldr (\x y -> y+(x!!1)) 2 [[1,2],[3,4],[5,6]]` produce como resultado  
☐ [3,4,5,6,7,8]  
☒ 14  
☐ Nada porque la expresión está mal tipada

9/9 9/9



122. Sea  $f :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$  una función conmutativa, y considérense las igualdades siguientes:

(i)  $\text{foldr } f \ 0 \ [3] = \text{foldl } f \ 0 \ [3]$

(ii)  $\text{foldr } f \ 0 \ [3,5] = \text{foldl } f \ 0 \ [3,5]$ . Entonces:

- ☐ Tanto (i) como (ii) son con seguridad ciertas.
- ☒ Sólo (i) es con seguridad cierta.
- ☐ Las dos anteriores son falsas.

124. Sea  $l$  una lista finita de enteros positivos. Entonces la evaluación de la expresión

$\text{foldr } (\backslash x \ y \rightarrow \text{if } x > 2 \text{ then } x \text{ else } y) \ 5 \ l$

- ☐ Produce el valor 3, con independencia de  $l$
- ☒ Produce el valor 5, con independencia de  $l$
- ☒ Las dos anteriores son falsas.

*Pura  $l = [1] \rightarrow 5$   
 $l = [3] \rightarrow 3$*

125. Sea  $l$  una lista finita de enteros positivos. Entonces la evaluación de la expresión

$\text{foldl } (\backslash x \ y \rightarrow \text{if } x > 2 \text{ then } x \text{ else } y) \ 5 \ l$

- ☐ Produce el valor 3, con independencia de  $l$
- ☒ Produce el valor 5, con independencia de  $l$
- ☐ Las dos anteriores son falsas.

126. La evaluación de la expresión

$\text{foldr } (\backslash x \ y \rightarrow \text{not } x \ || \ y) \ \text{False} \ [\text{False}, \text{True}, \text{undefined}]$  da como resultado

- ☒ True
- ☐ False
- ☐ Un error en tiempo de ejecución

127. La evaluación de la expresión

$\text{foldl } (\backslash x \ y \rightarrow \text{not } x \ || \ y) \ \text{False} \ [\text{False}, \text{True}, \text{undefined}]$  da como resultado

- ☐ True
- ☐ False
- ☒ Un error en tiempo de ejecución

128. La evaluación de la expresión

$\text{foldr } (\backslash x \ y \rightarrow (\text{not } y) \ \&\& \ x) \ \text{undefined} \ [\text{True}, \text{True}]$  da como resultado

- ☐ True
- ☐ False
- ☒ Un error en tiempo de ejecución

129. La evaluación de la expresión

$\text{foldl } (\backslash x \ y \rightarrow (\text{not } y) \ \&\& \ x) \ \text{undefined} \ [\text{True}, \text{True}]$  da como resultado

- ☐ True
- ☒ False
- ☐ Un error en tiempo de ejecución

130. La reducción de la expresión  $(\backslash x \ y \rightarrow (\backslash z \rightarrow y \ (z+2)) \ (y \ x)) \ 3 \ (\backslash x \rightarrow x+1)$  producirá el resultado

- ☐ 8
- ☒ 7
- ☐ 6

*8/8 8/8*

131. La reducción de la expresión  $(\lambda x y \rightarrow x (x y)) (\lambda x \rightarrow x + 3) 4$  producirá el resultado  
☐ 7  
☒ 10  
☐ Las dos anteriores son falsas.
132. La reducción de la expresión  $(\lambda x y \rightarrow x (x y)) (\lambda x \rightarrow x + y) 4$  producirá el resultado  
☐ 8  
☐ 12  
☒ Las dos anteriores son falsas.
133. La reducción de la expresión  $(\lambda x y \rightarrow (\lambda u \rightarrow x (u+1)) (x y)) (\lambda x \rightarrow x+2) 3$  producirá el resultado  
☒ 8  
☐ 7  
☐ 6
134. La reducción de la expresión  $(\lambda x y \rightarrow x (y x)) (\lambda x \rightarrow x+1) (\lambda x \rightarrow x 1)$  producirá el resultado  
☐ 1  
☐ 2  
☒ 3
135. La reducción de la expresión  $(\lambda x y \rightarrow (\lambda u \rightarrow x (u+1)) (x y)) (\lambda x \rightarrow x+1) 3$  producirá el resultado  
☐ 5  
☒ 6  
☐ 7
136. La reducción de la expresión  $(\lambda x y \rightarrow y (y x)) ((\lambda x \rightarrow x+1) 0) (\lambda x \rightarrow x+1)$  producirá el resultado  
☐ 1  
☐ 2  
☒ 3
137. La reducción de la expresión  $(\lambda x y \rightarrow x + (\lambda x \rightarrow y+x) y) ((\lambda x \rightarrow x+1) 5) 2$  producirá el resultado  
☒ 10  
☐ 14  
☐ Depende del orden en que se efectúe
138. La reducción de la expresión  $(\lambda x y \rightarrow x (y x)) (\lambda x \rightarrow x + 2) (\lambda x \rightarrow x 3)$  producirá el resultado  
☐ 5  
☐ 3  
☒ 7
139. La reducción de la expresión  $(\lambda x y z \rightarrow x z y) (\lambda x y \rightarrow x-y) ((\lambda x \rightarrow x+1) 2) 1$  producirá el resultado  
☐ 1  
☐ 2  
☒ -2
140. La reducción de la expresión  $(\lambda x y \rightarrow y (y x)) 1 (\lambda x \rightarrow x+2)$  producirá el resultado  
☐ 3  
☒ 5  
☐ 7

10/10

10/10

141. La reducción de la expresión  $(\lambda x y z \rightarrow x z (y z)) (\lambda x y \rightarrow y+1) (\lambda x \rightarrow x+1) 1$  producirá el resultado
- ☐ 1
- ☐ 2
- ☒ 3
142. Considérense las funciones:
- $$f \ x \ y = [2*i | i \leftarrow [1..x], i > y]$$
- $$f' \ x \ y = \text{filter } (> y) (\text{map } (2 *) [1..x])$$
- $$f'' \ x \ y = \text{map } (2 *) (\text{filter } (> y) [1..x])$$
- ☐ f, f' y f'' computan lo mismo
- ☐ f y f' computan lo mismo, pero f'' no
- ☒ f y f'' computan lo mismo, pero f' no
143. La evaluación de `map (zipWith (-) [3,2,1]) [[1,2,3],[4,5,6]] !! 1 !! 0` produce como resultado:
- ☐ [-1]
- ☒ -1
- ☐ Un error, porque esa expresión está mal tipada
144. La evaluación de `[j | j <- [i-1..i+1], i <- [1..5], i > 1]` produce como resultado
- ☐ 2
- ☐ 3
- ☒ Un error
145. La evaluación de `length [i | i <- [1..5], j <- [1..i], i+j < 5]` produce como resultado
- ☐ 3
- ☒ 4
- ☐ 5
146. La evaluación de `length [i+j | i <- [1..5], i > 2, j <- [3..i]]` produce como resultado
- ☐ 0
- ☒ 6
- ☐ Da un error en tiempo de ejecución
147. La evaluación de `length [i+j | i <- [1..5], i > 2, j <- [i-1,i]]` produce como resultado
- ☐ 9
- ☒ 6
- ☐ Da un error en tiempo de ejecución
148. La evaluación de `[i+j | i <- [1..5], i > 2, j <- [i-1,i]] !! 2` produce como resultado
- ☐ 9
- ☒ 7
- ☐ Da un error en tiempo de ejecución
149. La evaluación de `length [i+j | i <- [1..5], i > 2, j <- [i-1,i], j < i]` produce como resultado
- ☐ 6
- ☒ 3
- ☐ 0

10  
10

9/10

150. La evaluación de `take 2 [take j [i..2*i] | i <- [1..10], i > 2, j <- [i-1..i+1]]` produce el resultado
- ☐ [3,4]
  - ☒ [[3,4],[3,4,5]]
  - ☐ No produce ningún valor, porque la expresión está mal tipada
151. La evaluación de `length [take j [1..i] | i <- [1..5], i > 2, j <- [i-1,i]]` produce como resultado
- ☐ 3
  - ☒ 6
  - ☐ No produce ningún valor, porque la expresión está mal tipada
152. La evaluación de `map (zip [1..4]) [2,5..12]` produce el resultado
- ☐ [(1,11),(2,11),(3,11),(4,11)]
  - ☐ [(4,11)]
  - ☒ No produce ningún valor, porque la expresión está mal tipada
153. La evaluación de `map length [take 2 ys | x <- [1..3], ys <- iterate (+ 3) x]` produce como resultado
- ☐ [2,2,2]
  - ☐ [2]
  - ☒ Un error, porque la expresión está mal tipada
154. La evaluación de `(head.tail) (map ((take 3).(iterate (+ 2))) [1,3..10])` produce como resultado
- ☐ [4,6,8,11,14]
  - ☐ [1,4,7]
  - ☒ [3,5,7]
155. La evaluación de `take 3 (zipWith (+) [x+y|x<-[1..4],y<-[x,x+2,x+4]] (iterate (*2) 1))` produce como resultado
- ☐ [3,5,7]
  - ☒ [3,6,10]
  - ☐ []
156. La evaluación de `(head.tail) (map ((take 2).(iterate (+ 3))) [1,3..10])` produce como resultado
- ☐ [4,6,8,11,14]
  - ☐ [1,4]
  - ☒ [3,6]
157. La evaluación de `[j|i <- [1..5], i > 1, j <- [i-1..i+1]] !! i` produce como resultado
- ☐ 2
  - ☐ 3
  - ☒ Un error, porque hay una variable fuera de ámbito
158. La evaluación de `map (!! 2) (map (iterate (\x -> 2*x)) [0..3])` produce el resultado
- ☐ 2
  - ☒ [0,4,8,12]
  - ☐ No produce ningún valor, porque la expresión está mal tipada

9/9

9/9

159. La evaluación de `map fst [(zip [0..i] [1..j]) | i | i <- [1..3], j <- [1..(i+1)], j > i]` produce el resultado
- ☐ (0,1)
  - ☒ [1,2,3]
  - ☐ No produce ningún valor, porque la expresión está mal tipada
160. La evaluación de `((!! 2).head) (map (iterate (+2)) [2,0,4])` produce como resultado
- ☒ 6
  - ☐ [2,4,6]
  - ☐ Las dos anteriores son falsas.
161. La evaluación de `(head.(!! 1)) (map (zip [0..3]) [[1..4],[2..5]])` produce como resultado
- ☐ 1
  - ☐ (1,2)
  - ☒ (0,2)
162. ¿Cuál de las siguientes definiciones es equivalente a `f n m = [x*y | x <- [1..n], y <- [x..m]]`?
- ☒ `f n m = concat (map f [1..n]) where f x = map (\y -> x*y) [x..m]`
  - ☐ `f n m = concat (map f [x..m]) where f y = map (\y -> x*y) [1..n]`
  - ☐ `f n m = zipWith (*) [1..n] [x..m]`
163. ¿Cuál de las siguientes definiciones es equivalente a `f n m = [x*n | x <- [1..n], x > m]`?
- ☒ `f n m = map (\x -> x*n) (filter (> m) [1..n])`
  - ☐ `f n m = concat (map (\x -> x*n) (filter (> m) [1..n]))`
  - ☐ `f n m = filter (> m) (map (\x -> x*n) [1..n])`
164. Sea `f` una función definida previamente, y considérense las definiciones
- ```

g x y = let z = f x y in (z,x*y,z+1)
g' x y = (f x y,x*y,f x y + 1)
g'' x y = h x y (f x y)
h x y z = (z,x*y,z+1)

```
- ¿Qué afirmación es correcta?
- ☒ `g`, `g'` y `g''` computan los mismos valores, `g` y `g''` tienen una eficiencia similar, pero `g'` es menos eficiente
  - ☐ `g`, `g'` y `g''` computan los mismos valores, `g` y `g''` tienen una eficiencia similar, pero `g'` es más eficiente
  - ☐ No es verdad que `g`, `g'` y `g''` computen los mismos valores
165. Sea `h :: Int -> Int` una función costosa de calcular, y sean `f`, `f'`, `f''` definidas por
- ```

f x y = (x+u,y+u)      f' x y = (x+(h x) , y+(h x))      f'' x y = g x y (h x)
      where u = h x      g x y z = (x+z,y+z)

```
- ☐ Tanto `f'` como `f''` son extensionalmente equivalentes a `f` y comparables en eficiencia con ella.
  - ☒ Tanto `f'` como `f''` son extensionalmente equivalentes a `f`, pero `f'` es menos eficiente.
  - ☐ Las dos anteriores son falsas.
166. ¿Cuál de las siguientes afirmaciones es cierta?
- ☒ Las listas intensionales pueden eliminarse usando `map`, `filter`, `concat`, y recíprocamente
  - ☐ Las listas intensionales pueden eliminarse usando `map`, `filter`, `concat`, pero el recíproco no es cierto
  - ☐ Las dos anteriores son falsas.

167. Sea  $l = [(x,y) | x \leftarrow [0,2,3], b, y \leftarrow [1..x]]$ , donde  $b$  es una cierta expresión booleana, y sea  $n = \text{length } l$ .

¿Cuál de la siguientes situaciones **no** puede darse?

- ☐  $n = 0$   
☐  $n = 5$   
☒  $n = 4$

168. Sea  $l = [(x,y) | x \leftarrow [2,4], y \leftarrow [1..x], b]$ , donde  $b$  es una cierta expresión booleana, y sea  $n = \text{length } l$ .

¿Cuántas de la siguientes situaciones:  $n = 1$        $n = 5$        $n = 7$  pueden darse?

- ☐ Solo una de ellas  
☒ Solo dos de ellas  
☐ Las tres son posibles

169. Considérense las siguientes expresiones:

```
head (reverse [1..10^30])  
last (reverse [1..10^30])  
last (takeWhile (< 1000) ([1..10^30] ++ [1..10^30]))
```

¿Cuántas de ellas nos llevará toda la vida evaluarlas?

- ☐ Exactamente una de ellas  
☒ Exactamente dos de ellas  
☐ Las tres

170. Considérense las siguientes expresiones:

```
take 30 (reverse [1..10^30])  
reverse (take 30 [1..10^30])  
last (takeWhile (< 1000) (iterate (+ 2) 1))
```

¿Cuántas de ellas nos llevará toda la vida evaluarlas?

- ☒ Exactamente una de ellas  
☐ Exactamente dos de ellas  
☐ Las tres

171. Considérense las siguientes expresiones:

```
head (reverse (reverse [1..10^30]))  
reverse (reverse (take 30 [1..10^30]))  
take 100 (iterate (+ 2) 1 ++ [1..100]))
```

¿Cuántas de ellas nos llevará toda la vida evaluarlas?

- ☒ Exactamente una de ellas  
☐ Exactamente dos de ellas  
☐ Las tres

172. Considérense las siguientes definiciones de funciones:

$f \ x \ y \ z = x + y - z$        $g \ x \ y = f \ x \ x \ y$        $g' \ x = f \ x \ x$        $g'' = f \ x$

Entonces:

- ☐  $g$ ,  $g'$  y  $g''$  son equivalentes  
☒  $g$  y  $g'$  son equivalentes, pero la definición de  $g''$  contiene un error  
☐ Las dos anteriores son falsas.

173. Considérense las siguientes definiciones de funciones:

$f \ x \ y \ z = x + y - z$        $g \ x \ y = f \ x \ y \ y$        $g' \ x = f \ x$        $g'' \ x = f \ x \ y$

S/E

Entonces:

- ☐  $g, g'$  y  $g''$  son equivalentes
- ☐  $g$  y  $g'$  son equivalentes, pero la definición de  $g''$  contiene un error
- ☒ Las dos anteriores son falsas.

174. Considérense las siguientes definiciones de funciones:

- $f\ x\ y\ z = x + y - z$        $g\ x\ y\ z = f\ (x+1)\ y\ z$        $g'\ x\ y = f\ (x+1)\ y$        $g''\ x = f\ (x+1)$
- ☒  $g, g'$  y  $g''$  son equivalentes
  - ☐  $g$  y  $g'$  son equivalentes entre sí, pero no equivalentes a  $g''$
  - ☐ La definición de  $g''$  contiene un error

175. Sean las definiciones  $f\ g\ h\ x\ y = g\ (h\ x\ (g\ y))$      $f'\ g\ h\ x = g.\ (h\ x).\ g$      $f''\ g\ h = g.\ (\lambda x \rightarrow h\ x).\ g$

- ☐  $f, f'$  y  $f''$  son extensionalmente equivalentes
- ☒  $f$  y  $f'$  son extensionalmente equivalentes, pero  $f''$  no
- ☐  $f$  y  $f''$  son extensionalmente equivalentes, pero  $f'$  no

176. Sean las definiciones  $f\ g\ h\ x = g\ (h\ (g\ x))$      $f'\ g\ h\ x = g.h.\ (\lambda x \rightarrow g\ x)$      $f''\ g\ h = g.h.g$

- ☐  $f, f'$  y  $f''$  son extensionalmente equivalentes
- ☐  $f$  y  $f'$  son extensionalmente equivalentes, pero  $f''$  no
- ☒  $f$  y  $f''$  son extensionalmente equivalentes, pero  $f'$  no

177. Considérense las siguientes definiciones de funciones:

- $f1\ x\ y\ z = x .\ y$        $f2\ x\ y\ z = (x .\ y)\ z$        $f3\ x\ y\ z = x\ (y\ z)$        $f4\ x\ y = x .\ y$
- ☐  $f1, f2, f3, f4$  computan la misma función
  - ☐  $f1, f2, f3$  computan la misma función, pero  $f4$  no
  - ☒  $f2, f3, f4$  computan la misma función, pero  $f1$  no

178. Considérense las igualdades

$(take\ m).(take\ n) = take\ (\min\ n\ m)$      $(drop\ m).(drop\ n) = drop\ (n+m)$      $(take\ m).(drop\ n) = (drop\ n).(take\ (m+n))$

donde  $m, n$  son  $\geq 0$ . Se tiene:

- ☒ Las tres son correctas
- ☐ Solo dos son correctas
- ☐ Solo una es correcta

179. ¿Cuántas de las siguientes igualdades son correctas?

$map\ id = id$      $map\ f\ (xs\ ++\ ys) = (map\ f\ xs)\ ++\ (map\ f\ ys)$      $map\ (f.g)\ xs = (map\ f\ .\ map\ g)\ xs$

*Nota: en la primera de ellas, se entiende que las funciones de ambos lados se aplican solo a listas*

- ☒ Las tres
- ☐ Solo dos
- ☐ Solo una

180. Sea un programa funcional con dos funciones  $f, g :: Int \rightarrow Int$  y la función  $h = h$ . ¿Cuál de las siguientes situaciones es posible?

- ☐  $\llbracket g\ (f\ h) \rrbracket = 2$  y  $\llbracket g\ (f\ 0) \rrbracket = \perp$
- ☒  $\llbracket g\ (f\ h) \rrbracket = \perp$  y  $\llbracket g\ (f\ 0) \rrbracket = 2$
- ☐  $\llbracket g\ (f\ h) \rrbracket = 0$  y  $\llbracket g\ (f\ 0) \rrbracket = 2$

8/8

181. Sea un programa funcional con dos funciones  $f, g :: \text{Int} \rightarrow \text{Int}$  y la función  $h = h$ . ¿Cuál de las siguientes situaciones no es posible?

- ☒  $\llbracket g(f\ h) \rrbracket = 2$  y  $\llbracket g(f\ 0) \rrbracket = \perp$
- ☐  $\llbracket g(f\ h) \rrbracket = \perp$  y  $\llbracket g(f\ 0) \rrbracket = 2$
- ☐  $\llbracket g(f\ h) \rrbracket = \perp$  y  $\llbracket g(f\ 0) \rrbracket = \perp$

182. Sea un programa funcional con dos funciones  $f, g :: \text{Int} \rightarrow \text{Int}$  y la función  $\text{loop} = \text{loop}$ . ¿Cuál de las siguientes situaciones es posible?

- ☐  $\llbracket f(g\ \text{loop}) \rrbracket = 1$  pero  $\llbracket f(g\ 0) \rrbracket = \perp$
- ☒  $\llbracket f(g\ \text{loop}) \rrbracket = \perp$  pero  $\llbracket f(g\ 0) \rrbracket = 1$
- ☐  $\llbracket f(g\ \text{loop}) \rrbracket = 0$  pero  $\llbracket f(g\ 0) \rrbracket = 1$

183. De una función  $f$  solo sabemos que su tipo es  $\forall a. [a] \rightarrow \text{Int}$ . ¿Cuál de las siguientes situaciones es posible?

- ☐  $\llbracket f\ [1,2] \rrbracket = 2$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 1$
- ☒  $\llbracket f\ [1,2] \rrbracket = 1$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 1$
- ☐  $\llbracket f\ [1,2] \rrbracket = 1$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 2$

184. De una función  $f$  solo sabemos que su tipo es  $\forall a. \text{Eq } a \Rightarrow [a] \rightarrow \text{Int}$ . ¿Cuántas de las siguientes situaciones son posibles?

- $\llbracket f\ [1,2] \rrbracket = 2$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 1$
- $\llbracket f\ [1,2] \rrbracket = 1$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 1$
- $\llbracket f\ [1,2] \rrbracket = 1$  y  $\llbracket f\ [\text{True}, \text{True}] \rrbracket = 2$
- ☒ Las tres
- ☐ Solo dos
- ☐ Solo una

185. La función  $f$  definida por las siguientes ecuaciones:

|     |                |                |                  |
|-----|----------------|----------------|------------------|
| $f$ | $x$            | $\text{False}$ | $= \text{True}$  |
| $f$ | $\text{False}$ | $y$            | $= \text{True}$  |
| $f$ | $\text{True}$  | $\text{True}$  | $= \text{False}$ |

- ☒ Es estricta en el segundo argumento pero no en el primero
- ☐ No es estricta en ninguno de sus argumentos
- ☐ Es estricta en sus dos argumentos

186. Sean las funciones

```
f x y = if x > 0 then 1 else y
g x = if x > 0 then 1 else 0
h x y = (g. (f x)) y
```

- ☒ La función  $h$  es estricta en el primer argumento pero no en el segundo
- ☐ La función  $h$  no es estricta en ninguno de sus argumentos
- ☐ La función  $h$  es estricta en sus dos argumentos

187. Considérese el programa

```
f y 0 = g y
f 0 x = h x (x*x)
g x = if x > 0 then 1 else 0
h x y = if x > y then 1 else 0
```

- ☐ La función  $f$  es estricta en el segundo argumento pero no en el primero

6/7



- ☐ La función  $f$  no es estricta en ninguno de sus argumentos
- ☒ La función  $f$  es estricta en sus dos argumentos

188. Considérese el programa

```
f 0 y = g y
f x 0 = h x (x*x)
g x = if x > 0 then 1 else 0
h x y = if x > y then 1 else 0
```

- ☐ La función  $f$  es estricta en el primer argumento pero no en el segundo
- ☐ La función  $f$  no es estricta en ninguno de sus argumentos
- ☒ La función  $f$  es estricta en sus dos argumentos

189. Considérese el programa

```
f 0 y = g y
f x y = h y
g x = if x > 0 then 1 else 0
h x = 0
```

- ☐ La función  $f$  no es estricta en ninguno de sus argumentos
- ☐ La función  $f$  es estricta en sus dos argumentos
- ☒ Las dos anteriores son falsas.

190. La función  $f$  definida por las siguientes ecuaciones:

```
f False y = True
f x False = True
f True True = False
```

- ☐ No es estricta en ninguno de sus argumentos
- ☒ Es estricta en el primer argumento pero no en el segundo
- ☐ Las dos anteriores son falsas.

191. Sea la función  $f$  definida por las siguientes ecuaciones:

```
f x False = True
f False y = True
f True True = False
```

y considérense las siguientes afirmaciones: (a)  $\llbracket f e \perp \rrbracket = \perp$ , para toda expresión  $e$   
 (b)  $\llbracket f \perp e \rrbracket = \perp$ , para toda expresión  $e$

- ☐ (a) y (b) son ciertas
- ☒ (a) es cierta y (b) es falsa
- ☐ Las dos anteriores son falsas.

192. Considérese el programa

```
f 0 y = g y      g x = if x > 0 then 1 else 0
f x y = h y      h x = 0
mal = head []
```

- ☐ Existen  $e, e'$  tales que ni la evaluación de  $(f e \text{ mal})$  ni la de  $(f \text{ mal } e')$  dan error
- ☐ Para todo  $e, e'$   $f e \text{ mal}$  y  $f \text{ mal } e'$  dan error
- ☒ Las dos anteriores son falsas.

S/S

193. Sea  $f$  definida por las siguientes ecuaciones:

|     |                |                |                  |                                          |
|-----|----------------|----------------|------------------|------------------------------------------|
| $f$ | $x$            | $\text{False}$ | $= \text{True}$  | $y$ sea $\text{mal} = \text{head } []$ . |
| $f$ | $\text{False}$ | $y$            | $= \text{True}$  |                                          |
| $f$ | $-$            | $-$            | $= \text{False}$ |                                          |

¿Cuál de las siguientes afirmaciones es cierta?

- ☐  $f \text{ mal } e$  da error de ejecución, para cualquier expresión  $e$
- ☒  $f e \text{ mal}$  da error de ejecución, para cualquier expresión  $e$
- ☐  $f \text{ mal True}$  se evalúa a  $\text{False}$

194. Sea  $f$  definida por las siguientes ecuaciones:

|     |                |                |                  |                                          |
|-----|----------------|----------------|------------------|------------------------------------------|
| $f$ | $x$            | $\text{False}$ | $= \text{True}$  | $y$ sea $\text{mal} = \text{head } []$ . |
| $f$ | $\text{False}$ | $y$            | $= \text{True}$  |                                          |
| $f$ | $\text{True}$  | $\text{True}$  | $= \text{False}$ |                                          |

¿Cuál de las siguientes afirmaciones es cierta?

- ☐ La evaluación de  $f \text{ mal } e$  da error, para cualquier expresión  $e$
- ☒ La evaluación de  $f e \text{ mal}$  da error, para cualquier expresión  $e$
- ☐  $f \text{ mal True}$  se evalúa a  $\text{False}$

195. Sea  $f$  definida por las siguientes ecuaciones:

|     |                |                |                  |
|-----|----------------|----------------|------------------|
| $f$ | $\text{False}$ | $y$            | $= y$            |
| $f$ | $x$            | $\text{False}$ | $= \text{True}$  |
| $f$ | $\text{True}$  | $\text{True}$  | $= \text{False}$ |

y sea  $\text{mal} = \text{head } []$ . ¿Cuál de las siguientes afirmaciones es falsa?

- ☐ La evaluación de  $f \text{ mal } e$  da error, para cualquier expresión  $e$
- ☐ La evaluación de  $f e \text{ mal}$  da error, para cualquier expresión  $e$  cuya evaluación termine
- ☒ Las dos anteriores son falsas.

196. Considérense las definiciones:

|             |     |                    |               |   |
|-------------|-----|--------------------|---------------|---|
| $f \ x \ y$ | $ $ | $x == 0$           | $= 1$         | . |
|             | $ $ | $x < y$            | $= g \ (x-y)$ |   |
|             | $ $ | $\text{otherwise}$ | $= 2$         |   |
| $g \ x$     | $ $ | $x >= 0$           | $= x$         |   |

¿Qué afirmación es correcta?

- ☐  $f \ x \ y$  tiene un valor definido para valores cualesquiera de  $x$  e  $y$
- ☒  $f \ x \ y$  tiene un valor definido  $\Leftrightarrow x = 0 \vee x \geq y$
- ☐ La definición es errónea y será rechazada por el sistema

197. Considérese el programa

```
f x 0 = g x
f x y = h x
g x = if x > 0 then 1 else 0
h x = 0
```

- ☐ La función  $f$  no es estricta en ninguno de sus argumentos
- ☐ La función  $f$  es estricta en sus dos argumentos
- ☒ Las dos anteriores son falsas.

198. Sea  $f$  definida por las siguientes ecuaciones:

|     |     |                |                  |                                         |
|-----|-----|----------------|------------------|-----------------------------------------|
| $f$ | $x$ | $\text{False}$ | $= \text{False}$ | ¿Cuál de las siguientes afirmaciones es |
| $f$ | $y$ | $\text{True}$  | $= y$            | cierta?                                 |

- ☐ La función es estricta en sus dos argumentos

5/6

- ☒ La función es estricta en el segundo pero no en el primer argumento
- ☐ Las dos anteriores son falsas.

199. Sea  $f$  definida por las siguientes ecuaciones:  $f \ x \ \text{False} = x$       ¿Cuál de las siguientes afirmaciones es cierta?  
 $f \ y \ \text{True} = \text{not } y$

- ☒ La función es estricta en sus dos argumentos
- ☐ La función es estricta en el segundo pero no en el primer argumento
- ☐ Las dos anteriores son falsas.

200. Sea  $f$  definida por las siguientes ecuaciones:  $f \ x \ \text{False} = \text{True}$       ¿Cuál de las siguientes afirmaciones es cierta?  
 $f \ \text{True} \ \text{True} = \text{False}$

- ☐ La función es estricta en sus dos argumentos
- ☒ La función es estricta en el segundo pero no en el primer argumento
- ☐ Las dos anteriores son falsas.

201. Dada la definición del tipo  $\text{data } T = A \mid B \mid C \text{ T } T$ , considérese su dominio  $\mathcal{D}_T$ , y el orden de aproximación sobre él  $\sqsubseteq$ .  
 ¿Cuál de las siguientes afirmaciones es cierta?

- ☐  $A \sqsubseteq B \sqsubseteq C \ A \ B$
- ☐  $\mathcal{D}_T$  es finito y  $\perp$  es el elemento mínimo para  $\sqsubseteq$
- ☒ Las dos anteriores son falsas.

202. Dada la definición del tipo  $\text{data } T = A \mid B \mid C \text{ T } T$ , considérese su dominio  $\mathcal{D}_T$ , y el orden de aproximación sobre él  $\sqsubseteq$ .  
 ¿Cuál de las siguientes afirmaciones es cierta?

- ☐  $A$  y  $B$  son los únicos maximales para  $\sqsubseteq$
- ☒  $A$  y  $B$  son maximales para  $\sqsubseteq$  pero no son los únicos
- ☐ Las dos anteriores son falsas.

203. Considérese la definición del tipo  $\text{data } T = A \mid B \mid C \text{ T } T$ , y sea  $\sqsubseteq$  el orden de información o de aproximación sobre los valores de  $T$ . ¿Cuál de las siguientes afirmaciones es cierta?

- ☐  $\perp \sqsubseteq A \sqsubseteq B \sqsubseteq C \ \perp \ \perp$
- ☐  $A$  y  $B$  son los únicos valores maximales para  $\sqsubseteq$
- ☒  $\perp \sqsubseteq C \ \perp \ \perp \sqsubseteq C \ \perp \ B \sqsubseteq C \ A \ B$

204. Considérese la definición del tipo  $\text{data } T = A \mid B \mid C \text{ T } T$ , y sea  $\sqsubseteq$  el orden de información o de aproximación sobre los valores de  $T$ . ¿Cuál de las siguientes afirmaciones es cierta?

- ☐  $\perp \sqsubseteq A \sqsubseteq B \sqsubseteq C \ \perp \ \perp$
- ☒  $A$  y  $B$  son valores maximales para  $\sqsubseteq$
- ☐  $\perp \sqsubseteq C \ A \ \perp \sqsubseteq C \ \perp \ B \sqsubseteq C \ A \ B$

205. Considérese la definición del tipo  $\text{data } T = A \mid B \mid C \text{ T } T$ , y sea  $\sqsubseteq$  el orden de información o de aproximación sobre los valores de  $T$ . ¿Cuál de las siguientes afirmaciones es falsa?

- ☐  $\perp \sqsubseteq C \ \perp \ \perp \sqsubseteq C \ A \ A$
- ☐  $C \ A \ A$  es un valor maximal para  $\sqsubseteq$
- ☒  $\perp \sqsubseteq C \ \perp \ B \sqsubseteq C \ B \ \perp \sqsubseteq C \ A \ B$

2/2

206. En lo que sigue,  $\sqsubseteq$  indica el orden de aproximación entre valores semánticos. ¿Cuál de las siguientes afirmaciones es cierta?
- ☐  $\perp \sqsubseteq [0] \sqsubseteq [0, \perp]$
  - ☐  $\perp \sqsubseteq [\perp, \perp] \sqsubseteq [0, \perp]$
  - ☒  $\perp \sqsubseteq [\perp, \perp] \sqsubseteq [0, \perp]$
207. En lo que sigue,  $\sqsubseteq$  indica el orden de aproximación entre valores semánticos. ¿Cuál de las siguientes afirmaciones es cierta?
- ☐  $[] \sqsubseteq [\perp] \sqsubseteq [\perp, \perp]$
  - ☐  $[\perp] \sqsubseteq [\perp, \perp] \sqsubseteq [0, \perp]$
  - ☒ Las dos anteriores son falsas.
208. En lo que sigue,  $\sqsubseteq$  indica el orden de aproximación entre valores semánticos del tipo  $[\text{Int}]$ . ¿Cuál de las siguientes afirmaciones es falsa?
- ☐  $\perp \sqsubseteq [\perp] \sqsubseteq [0]$
  - ☒  $[\perp] \sqsubseteq [\perp, \perp] \sqsubseteq [0, \perp]$
  - ☐  $\perp \sqsubseteq [\perp, \perp] \sqsubseteq [0, \perp]$
- 
209. El valor de la expresión `let x=1:3:x in take 3 x` es:
- ☒ `[1,3,1]`
  - ☐  $\perp$ , porque la evaluación no termina
  - ☐ Hay un error sintáctico porque `x` no puede aparecer en el lado derecho de `let x= ...`
210. El valor de la expresión `let x=1:3:x in head (x ++ x)` es:
- ☒ `1`
  - ☐  $\perp$ , porque la evaluación no termina
  - ☐ Hay un error sintáctico porque `x` no puede aparecer en el lado derecho de `let x= ...`
211. El valor de la expresión `let x= reverse (x++[1]) in head x` es:
- ☐ `1`
  - ☒ La evaluación no termina
  - ☐ La expresión es incorrecta, porque `x` no puede aparecer a la derecha de `let x = ...`
212. El valor de la expresión `let x=x++x in head (2:x)` es:
- ☒ `2`
  - ☐  $\perp$ , porque la evaluación no termina
  - ☐ La expresión está mal construida o mal tipada
213. El valor de la expresión `let x=x++[1] in last x` es:
- ☐ `1`
  - ☒  $\perp$ , porque la evaluación no termina
  - ☐ La expresión está mal construida o mal tipada
214. Considere la evaluación de las expresiones `let x=2:filter (/=2) x in head x`  
`let x=2:filter (/=2) x in head (tail x)`
- ☐ Ninguna de las dos termina
  - ☒ Una de las dos termina
  - ☐ Las dos terminan

215. El valor de la expresión `let x=x++[1] in x!!1` es:  
☐ 1  
☒  $\perp$ , porque la evaluación no termina  
☐ La expresión está mal construida o mal tipada
216. El valor de la expresión `let x=[1]++x in head (tail x)` es:  
☒ 1  
☐ La evaluación no termina  
☐ La expresión está mal construida o mal tipada
217. El valor de la expresión `let x=1:y in head x` es:  
☐ 1  
☐  $\perp$ , porque la evaluación no termina  
☒ La expresión está mal construida o mal tipada
218. La evaluación de la expresión `let x= 1:map (+ 2) x in take 3 x` produce como resultado:  
☒ [1,3,5]  
☐ Una lista que empieza por 1 y luego da un error  
☐ Un error sintáctico o de tipos
219. La evaluación de la expresión `let x= 1:3:map (+ 1) x in last (take 3 x)` produce como resultado:  
☒ 2  
☐ Un error en ejecución  
☐ Un error sintáctico o de tipos
220. La evaluación de la expresión `let x= 1:map (+ 1) x in length (take 4 x)` produce como resultado:  
☒ 4  
☐ Un cómputo no terminante  
☐ Un error
221. Dadas las expresiones: `length [1..]` y `let x=length [1..] in fst (1,x)`  
☒ La evaluación de la segunda termina pero la de la primera no  
☐ La evaluación de ninguna de las dos termina  
☐ Las dos anteriores son falsas.
222. Considérense las definiciones  
`f x = 1 + f (x+1)`  
`g x = if x >= 1 then 1 else 0`  
`h x = 3`  
y las expresiones  $e \equiv g (f 1)$  y  $e' \equiv h (g (f 1))$ . Entonces:  
☐ Ni la evaluación de  $e$  ni la de  $e'$  terminan, tanto al usar evaluación impaciente como perezosa.  
☐ Ni la evaluación de  $e$  ni la de  $e'$  terminan al usar evaluación impaciente, pero sí lo hacen (ambas) al usar evaluación perezosa.  
☒ Las dos anteriores son falsas.
223. Sea  $e$  una expresión. ¿Cuál de las siguientes situaciones es posible?  
☐ Al evaluar  $e$  por evaluación impaciente se obtiene el valor 3, y por evaluación perezosa el valor 2  
☐ Al evaluar  $e$  por evaluación impaciente resulta el valor 3, y por evaluación perezosa el cómputo no termina  
☒ Al evaluar  $e$  por evaluación perezosa resulta el valor 3, y por evaluación impaciente el cómputo no termina

224. Sea  $e$  una expresión. ¿Cuántas de las siguientes situaciones son posibles al evaluar  $e$ ?

- Por evaluación impaciente se obtiene el valor 3, y por evaluación perezosa el valor 2.
  - Se obtiene el valor 3 tanto por evaluación impaciente como por evaluación perezosa.
  - Por evaluación perezosa el cómputo no termina, y por evaluación impaciente el cómputo termina.
  - Por evaluación perezosa el cómputo termina, y por evaluación impaciente el cómputo no termina.
  - El cómputo no termina ni por evaluación impaciente ni por evaluación perezosa.
- ☐ Exactamente dos
- ☒ Exactamente tres
- ☐ Las dos anteriores son falsas.

225. Sean las funciones  $f$  y  $g$  definidas por  $f\ x = f\ x$  y por  $g\ f\ x = \text{if } x == 0 \text{ then } 0 \text{ else } f\ (x-1)$ .

Entonces, al evaluar la expresión  $g\ f\ 1$ :

- ☐ Con evaluación perezosa se obtiene el valor 0, y con evaluación impaciente el cómputo no termina.
- ☐ Tanto con evaluación perezosa como con evaluación impaciente se obtiene el valor 0.
- ☒ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

226. Sean las funciones  $f$  y  $g$  definidas por  $f\ g = g\ (f\ g)$  y por  $g\ f\ x = \text{if } x == 0 \text{ then } 0 \text{ else } f\ (x-1)$ .

Entonces, al evaluar la expresión  $f\ g\ 1$ :

- ☒ Con evaluación perezosa se obtiene el valor 0, y con evaluación impaciente el cómputo no termina.
- ☐ Tanto con evaluación perezosa como con evaluación impaciente se obtiene el valor 0.
- ☐ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

227. Sean las funciones  $f$  y  $g$  definidas por  $f\ g = g\ (f\ g)$  y por  $g\ f\ x = \text{if } x == [] \text{ then } 0 \text{ else } 1 + f\ (\text{tail } x)$ .

Entonces, al evaluar la expresión  $f\ g\ [2]$ :

- ☐ Con evaluación perezosa se obtiene el valor 2, y con evaluación impaciente el cómputo no termina.
- ☒ Con evaluación perezosa se obtiene el valor 1.
- ☐ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

228. ¿Cuáles de las dos siguientes expresiones representa correctamente una acción de  $I/O$ ?

```
do x <- getChar
  x
```

```
do x <- getChar
  return x
```

- ☐ Las dos
- ☐ Solo la primera
- ☒ Las dos anteriores son falsas.

229. ¿Cuál de las siguientes expresiones denota correctamente la acción de leer un carácter y escribirlo dos veces?

```
let x=getChar in [x,x]
```

```
do x <- getChar
  return x
  return x
```

☒

```
do x <- getChar
  putStr [x,x]
```

230. Considérense las definiciones:

```
f :: IO ()
f = do x <- getChar
      print x
```

```
g = do x <- f
      print x
```

Entonces, la evaluación de `g` tiene por efecto:

- ☐ Leer un carácter y escribirlo dos veces
- ☐ `g` no llegará a evaluarse pues hay errores de tipo en ese código
- ☒ Las dos anteriores son falsas.

