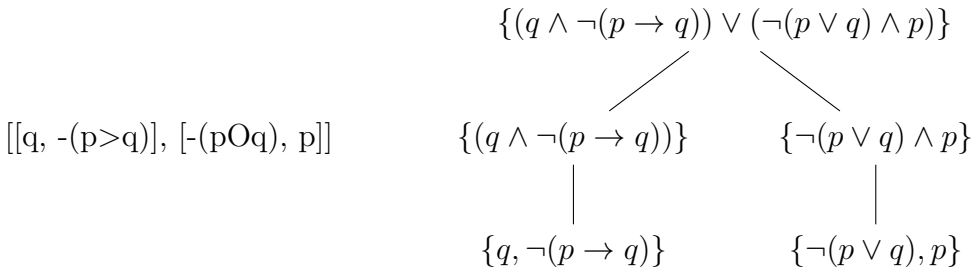
Antes de desarrollar este taller, le recomendamos tener a mano el algoritmo de construcción de tableaux (pág. 15 de las diapositivas correspondientes a la sesión 8 del curso).

En primer lugar, es importante considerar que el algoritmo de construcción de tableaux no necesita desarrollarse sobre un árbol, sino que puede desarrollarse sobre la lista de hojas del árbol. Si cada hoja se ve en sí misma como una lista de fórmulas, entonces la estructura de datos sobre la cual se desarrolla el algoritmo de construcción de tableaux es una lista de *listas* de fórmulas. Por ejemplo, la estructura de la izquierda corresponde al tableaux de la derecha (observe que en la estructura de la izquierda sólo se consideran las hojas del árbol de la derecha):



En el archivo `para_tableaux101.py` podrá encontrar las funciones que le permiten imprimir en pantalla una lista de fórmulas (`imprime_hoja()`) y una lista de *listas* de fórmulas (`imprime_tableau()`).

Una posible implementación en Python del algoritmo de construcción de tableaux puede desarrollarse de la siguiente manera. El primer objetivo es implementar los rombos de decisión que aparecen en el algoritmo de construcción de tableaux. La sugerencia es comenzar con el rombo correspondiente a la decisión "¿h contiene un par complementario?", donde h es una hoja que contiene sólo literales. Observe que, en su representación 'pythonesca', h es una lista de *literales*.

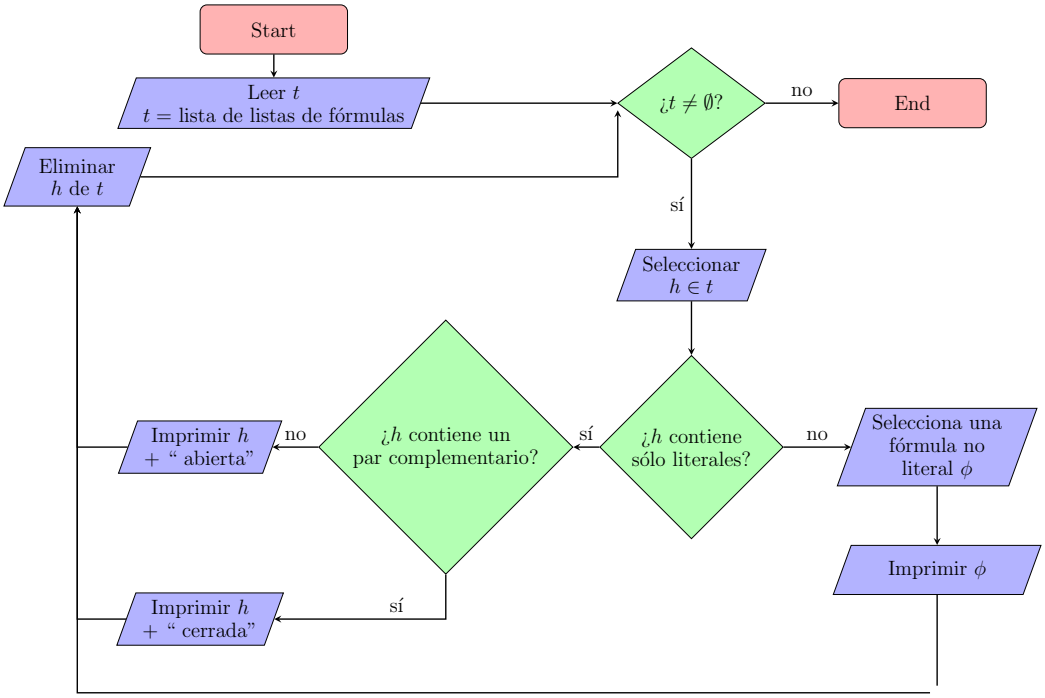
EJERCICIO 1: Implemente en Python una función que, dada una lista de literales h, verifique si en h hay por lo menos un par complementario (y finalice en el momento en que encuentre el primero). Verifique su función con las listas $[p, q]$, $[p, q, \neg p]$ y $[p, q, \neg p, \neg q]$. En la primera no hay un par complementario, en la segunda y la tercera sí. La ejecución sobre la última lista debe detenerse tan pronto verifique que está el par p, $\neg p$.

El siguiente paso consiste en implementar el proceso de decisión que alimenta el anterior rombo, es decir, nos estamos refiriendo al rombo correspondiente a la decisión "¿h contiene sólo literales?". En este caso, h es una hoja cualquiera, que contiene fórmulas. Observe que, en su representación 'pythonesca', h es una lista de *fórmulas*.

EJERCICIO 2: Implemente en Python una función que, dada una fórmula A, verifique si A es un literal. Verifique su función con las fórmulas p, $\neg p$, $\neg\neg p$, $\neg(p \wedge q)$. Las dos primeras son literales, las dos siguientes no.

EJERCICIO 3: Implemente en Python una función que, dada una lista de fórmulas h, verifique si en h hay alguna fórmula que no es un literal (y finalice en el momento en que encuentre la primera). Verifique su función con las listas $[p, q]$, $[\neg p, q]$, $[p, \neg\neg q]$, $[\neg\neg p, \neg(p \wedge q)]$. En las dos primeras todas las fórmulas son literales, en las dos siguientes hay fórmulas que no son literales. La ejecución sobre la última lista debe detenerse tan pronto verifique que $\neg\neg p$ no es un literal.

EJERCICIO 4: Implemente en Python un código que desarrolle la siguiente parte del algoritmo de construcción de tableaux:



Verificar el funcionamiento del código con $[[p], [\neg\neg p, q], [q, \neg q], [\neg(p \wedge \neg q), q]]$. Debería aparecer algo como esto (tal vez en distinto orden):

[p] abierta

- -p

[q,-q] cerrada

-(pY-q)

Consideremos ahora el rombo correspondiente a la decisión "¿árbol tiene hojas sin marcar?". Implementar este rombo requiere que establezcamos la manera mediante la cual vamos a representar la marcación de una hoja. La sugerencia es la siguiente. Dada la lista de hojas `listaHojas`, marcar la hoja h corresponde a eliminar a h de `listaHojas`. De esta manera, el rombo correspondiente a la decisión "¿árbol tiene hojas sin marcar?" puede resolverse así: hay hojas sin marcar sii la lista `listaHojas` es no vacía.

Debemos justificar esta representación de la marcación de las hojas. La marcación de una hoja con $\odot$ puede consistir en lo siguiente. Debemos crear, antes de todo el proceso de construcción de tableaux, una lista vacía llamada `listaInterpsVerdaderas` con el propósito de que, cada vez que debamos marcar una hoja con $\odot$, lo que hacemos es removerla de `listaHojas` e incluirla en la lista `listaInterpsVerdaderas`. Más adelante veremos que las hojas marcadas con $\odot$ serán muy útiles, pues ellas dan lugar a una interpretación (parcial) de la fórmula que se encuentra en la raíz del tableaux. Por esta razón, es importante guardar estas hojas.

Ahora bien, marcar una hoja con $\times$ es simplemente removerla de `listaHojas`, de tal manera que el rombo correspondiente a la decisión "¿hay hojas con $\odot$?" puede resolverse así: hay hojas marcadas con $\odot$ sii la lista `listaInterpsVerdaderas` es no vacía.

> Ejercicio 5: Implemente en Python una función que, dada una lista h de *listas* de *literales*, devuelva una lista llamada listaInterpsVerdaderas con las listas en h que no tienen pares complementarios. Verifique su función con las listas $[]$, $[[p, q], [\neg p, p]]$, $[[p, q], [q, \neg q], [q]]$. La primera devuelve [] y las otras dos listas devuelven $[[p, q]]$ y $[[p, q], [q]]$, respectivamente.

Ahora la parte carnuda del algoritmo: clasificar una fórmula como α o β, y extender el tableaux de acuerdo a la regla respectiva. Aquí hay que hacer una serie de decisiones que comienza con un if, seguido de muchos elifs; una decisión para cada tipo de fórmula. Cada decisón podrá tener más ifs y elifs anidados. Por ejemplo, determinar si un `A = Tree(label, left, right)` es de categoría 1α requiere considerar el código de la izquiera; observe que aumentar el código para determinar si `A` es 3α no es muy difícil (ver código de la derecha):

```
Si A.label == '-':
    Si A.right.label == '-':
        retornar '1alfa'
```

```
Si A.label == '-':
    Si A.right.label == '-':
        retornar '1alfa'
    Si no, si A.right.label == 'O':
        retornar '3alfa'
```

EJERCICIO 6: Implemente en Python una función que, dada una fórmula que no es un literal A, la clasifique como 1α, 2α, 3α o 4α (pág. 16 de las diapositivas correspondientes a la sesión 8 del curso), o como 1β, 2β o 3β (pág. 18 de las diapositivas correspondientes a la sesión 8 del curso). Verifique su función con las fórmulas:

$\neg\neg\big(\neg(p \vee q) \wedge \neg(r \to s)\big)$	1α	$\neg(p \wedge (r \to s))$	1β
$\neg(p \vee q) \wedge \neg(r \to s)$	2α	$\neg(p \wedge q) \vee (r \to s)$	2β
$\neg(\neg(r \to s) \vee q)$	3α	$r \to (s \vee q)$	3β
$\neg(r \to \neg(p \vee q))$	4α		

EJERCICIO 7: Implemente en Python una función que, dada una lista de *listas* de fórmulas, t, mientras alguna lista h en t contenga fórmulas que no son literales, tome una fórmula que no es un literal en h, la clasifique y ejecute la regla apropiada. Verifique su función con las listas:

$[[p], [q]]$	0 iteraciones y devuelve $[[p], [q]]$
$[[p], [q \wedge p]]$	1 iteración y devuelve $[[p], [q, p]]$
$[[p, q], [\neg(p \vee (r \wedge s))]]$	2 iteraciones y devuelve $[[p, q], [\neg p, \neg r], [\neg p, \neg s]]$
$[[p], [\neg\big(\neg(r \to s) \vee q\big)]]$	3 iteraciones y devuelve $[[p], [\neg r, \neg q], [s, \neg q]]$

EJERCICIO 8: Haga más eficiente el código del ejercicio 7 integrando el código del ejercicio 5. De esta manera, cada vez que en t encuentre un h que sólo contiene literales, lo clasifica y luego lo elimina. Además, si h es abierta, la incluye en listaInterpsVerdaderas. Verifique su función con las listas:

$[[p, \neg p], [\neg(p \vee (r \wedge s))]]$	2 iteraciones y devuelve $[[\neg p, \neg r], [\neg p, \neg s]]$
$[[p], [\neg\big(\neg(r \to q) \vee q\big)]]$	3 iteraciones y devuelve $[[p], [\neg r, \neg q]]$

EJERCICIO 9: Implemente en Python el algoritmo de construcción de tableaux. Verifique su función con las fórmulas $(p \vee q) \wedge (\neg p \wedge \neg q)$, $\neg(p \wedge \neg p)$, $\neg\neg\big(\neg(p \vee q) \wedge \neg(r \to s)\big)$. El tableaux para la primera es cerrado, mientras que los de las dos últimas son abiertos.