

Además de las funciones auxiliares de cada módulo, también tienen algunas funciones ya implementadas en **Logic.Propositions**.

## 1 Resolución

El propósito de esta sección es implementar resolución para fórmulas proposicionales. Esto requiere de varias funciones auxiliares que tiene que implementar.

1 pt **clashes** :: **Clause** a -> **Clause** a -> **Maybe** a

Determinar si dos cláusulas tienen alguna literal complementaria, y de tenerla regresarla. En caso de tener varias, regresar la primera que aparezca en la primera cláusula.

*Hint:* usar lista por comprensión y la función **areComplement**.

2 pt **resolve** :: **Clause** a -> **Clause** a -> **Clause** a

Dadas dos cláusulas con literales complementarias, aplicar la regla de resolución para generar una nueva cláusula.

*Hint:* usar la función anterior para encontrar la literal, y definir una función auxiliar nueva para quitarla de una cláusula.

0.5 pt **isTrivial** :: **Clause** a -> **Bool**

Determinar si una cláusula es trivial. Es decir, si tiene literales complementarias.

0.5 pt **hasEmpty** :: **Clause** a -> **Bool**

Determinar si una lista de cláusulas tienen una cláusula vacía.

1 pt **clashList** :: [**Clause** a] -> [**Clashing** a]

Dada una lista de cláusulas, obtener los pares de cláusulas en conflicto. Es decir, las cláusulas que tiene literales complementarias.

*Hint:* pueden usar la función **isJust** para procesar el resultado de **clashes**.

1 pt **someClash** :: **ResolReg** a -> **Maybe** (**Clashing** a)

Dada una lista de cláusulas, buscar una pareja de cláusulas en conflicto que no haya sido resulta.

*Hint:* Basta con tomar la salida de **clashList**, quitar las cláusulas que ya han sido procesadas y tomar el primer elemento.

2 pt **applyRes** :: **ResolReg** a -> **Bool**

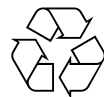
Aplica la regla de resolución sobre la lista de cláusulas hasta procesar todas o encontrar la cláusula vacía.

*Hint:* El caso base sería cuando **someClash** regrese **Nothing** o cuando **hasEmpty** regrese **True**. No olviden que la resolución de dos cláusulas solo se agrega a la lista de cláusulas si es no trivial.

0.5 pt **resolution** :: [**Clause** a] -> **Bool**

Dada una lista de cláusulas, determinar si es satisfacible usando el algoritmo de resolución. Irónicamente, a pesar de ser la función más importante del módulo, no es complicada.

*Hint:* basta con llamar a **applyRes** usando los valores iniciales adecuados. No olviden que no se puede resolver cláusulas triviales.



## 2 3CNF

El objetivo de este módulo es implementar la transformación de una fórmula en forma clausular a otra forma clausular equisatisficible, donde cada cláusula tiene tamaño exactamente tres usando las siguientes reglas

- Una literal  $l$

1. Añadir una literal complementaria  $p$  a  $l$

$$\{lp, l\bar{p}\}$$

2. Aún no son tres variables por cláusula. Añadir otra literal complementaria  $q$

$$\{lpq, lp\bar{q}, l\bar{p}q, l\bar{p}\bar{q}\}$$

- Dos literales  $lm$ . Añadir una nueva literal  $p$  complementaria

$$\{lmp, lm\bar{p}\}$$

- Más de tres literales  $l_1l_2l_3 \dots l_n$ . Añadir una literal complementaria  $p$

$$\{l_1l_2p, \bar{p}l_3 \dots l_n\}$$

Y recursivamente transformar  $\bar{p}l_3 \dots l_n$  en forma 3CNF.

Para implementar esto, tiene que hacer primero algunas funciones auxiliares.

1 pt `clauseMax :: [Clause] a -> a`

Obtener el valor máximo entre las literales de una lista de cláusulas. Para esto, pueden que los valores de las literales están ordenadas.

*Hint:* puede procesar primero individualmente cada cláusula y luego juntar los resultados. Pueden usar la función `getAtom` para obtener el valor de las literales y `maximum` para obtener el máximo de una lista.

2 pt `canonRule :: Clause a -> a -> [Clause] a`

Aplicar las reglas definidas arriba. Los casos base serían cuando la cláusula tiene cero, uno, dos o tres literales. El segundo parámetro que recibe la función se usa para generar variables nuevas usando `succ`.

*Hint:* no olviden que cada vez que generan una nueva variable el valor actualizado debe ser pasado a la llamada recursiva.

2 pt `to3CNFMax :: [Clause] a -> a -> [Clause] a`

Aplica la regla anterior a cada cláusula, llevando registro del máximo global para generar nuevas variables.

*Hint:* no olviden que cada vez que procesen una nueva cláusula el valor actualizado del máximo debe ser pasado a la llamada recursiva. Para actualizar ese valor puede usar `safeMax` para tener que realizar casos extra.

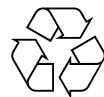
0.5 `to3CNF :: [Clause] a -> [Clause] a`

Transforma una lista de cláusulas a forma 3CNF.

*Hint:* basta con llamar a `to3CNFMax` con los valores iniciales adecuados.



¿Realmente necesitas imprimir esta hoja?



### 3 Transformación de Tseitin

Es un proceso alternativo de pasar una fórmula a CNF. Requiere hacer los siguientes pasos

1. Elegir una expresión  $l \circ l'$  de la fórmula  $A$
2. Reemplazarla por una nueva variable  $p$

$$A' = A_{p \leftarrow l \circ l'}$$

3. Añadir la asignación  $p \iff l \circ l'$  al resto de la fórmula

$$p \iff l \circ l' \wedge A'$$

4. Repetir el proceso con  $A'$  hasta que no tenga expresiones de esa forma. Es decir, cuando tenga a lo más una variable.
5. Pasar cada asignación  $p \iff q$  a CNF, usando las reglas normales para transformar a CNF.

Los pasos extra aseguran que cada expresión tendrá a lo más tres variables, lo cual genera cláusulas más cortas al pasar a forma clausular.

En este módulo deben implementar los pasos de esta transformación, excepto el último, ya que eso se puede hacer usando las funciones de la práctica anterior.

0.5 pt `isBase :: Formula a -> Bool`

Determinar si una fórmula puede o no ser reemplazada por una asignación. Esto es si contiene operadores binarios.

1 pt `findBin :: Formula a -> Maybe (Formula a)`

Dada una fórmula encontrar una expresión de la forma  $p \circ q$  donde  $p$  y  $q$  no contengan operadores binarios.

*Hint:* usar `isBase` para determinar cuando  $p$  y  $q$  no contengan operadores binarios.

2 pt `toTseitinMax :: Formula a -> a -> Formula a`

Aplicar los primeros pasos de la transformación de Tseitin usando la función anterior. Para generar nuevas variables se usa el mismo mecanismo que para pasar a 3CNF.

*Hint:* El caso base es cuando `findBin` no encuentre otra fórmula para reemplazar. Tienen disponibles `toSubst` y `applySubst` para aplicar la sustitución.

0.5 pt `toTseitin :: Formula a -> Formula a`

Aplica la transformación de Tseitin a una fórmula. Basta con llamar a `toTseitinMax` con los valores adecuados.

*Hint:* pueden usar `getAtoms` y `maximum`.



¿Realmente necesitas imprimir esta hoja?

