LÓGICA PARA COMPUTACIÓN
TRABAJO ENTREGABLE

Se aceptan entregas por Aulas hasta el 22/6/2016 a las 23:50

El objetivo del presente trabajo es implementar en Haskell un verificador de demostraciones. Éste permite garantizar que una demostración de validez de un razonamiento en Lógica de Primer Orden es correcta, utilizando el método de tableaux. En la primera sección se describen las características del verificador mientras que en la segunda sección se plantea su diseño e implementación. Finalmente, en la tercera parte, se describen los entregables.

Características del verificador. Recordemos que el método de tableaux para Lógica de Primer Orden, a diferencia de su versión en Lógica Proposicional, es *semidecidible*. Es capaz de detectar mecánicamente si el conjunto de fórmulas que se está tratando es inconsistente, es decir, si no tiene modelos. Sin embargo, para el caso contrario no ofrece un criterio mecánico, y es el usuario investigador quien debe detectar y hacer evidente la existencia de un modelo. En otras palabras, puede verse al método de tableaux como un auxiliar en la construcción de modelos de un conjunto de fórmulas, que entre sus servicios provee la detección de la imposibilidad de tal construcción. Como consecuencia de lo anterior, no es posible implementar un programa que, usando el método de tableaux, decida mecánicamente si un razonamiento en Lógica de Primer Orden es o no válido[1]. Nótese que como el método es capaz de detectar la inexistencia de modelos para un conjunto de fórmulas, también es capaz de detectar la validez de un razonamiento ya que un razonamiento es válido si y sólo si no existe modelo para el conjunto de fórmulas formado por las premisas del razonamiento y su conclusión negada.

En este trabajo vamos a implementar un programa que dado un razonamiento válido y una lista de reglas, verifica si la aplicación en orden de dichas reglas permite demostrar la validez del mismo. Notar que el problema de determinar si un árbol de tableaux se cierra por completo al aplicarle una sucesión de reglas es decidible.

Diseño e implementación. En primer lugar definimos tipos para representar los sublenguajes de términos y fórmulas de la Lógica de Primer Orden de la siguiente manera:

```
type Var = String
```

[1]Vale destacar que esta no es una falencia particular del método de tableaux, sino que no es posible escribir un algoritmo que decida si un razonamiento en esta Lógica es o no válido dado el carácter potencialmente infinito del dominio de individuos en cuestión.

```
type Simbolo = String

data Form = A Simbolo [Term] | Neg Form | Bc BinCon Form Form | All Var Form | Ex Var Form
    deriving (Eq, Show)

data Term = V Var | C Simbolo | F Simbolo [Term]
    deriving (Eq, Show)

data BinCon = And | Or | Impl
    deriving (Eq, Show)
```

Para representar un árbol de tableaux usaremos el tipo ArbolTableaux definido como una lista de listas de fórmulas:

```
type Rama = [Form]
type ArbolTableaux = [Rama]
```

Una rama del árbol es una lista de fórmulas a la cual es posible aplicarle reglas. Un árbol de tableaux está formado por una lista de ramas. Inicialmente el árbol contendrá una única rama y es posible generar nuevas ramas sólo mediante la aplicación de reglas disyuntivas. También tenemos un tipo de datos para representar las clases de reglas que se pueden aplicar sobre una fórmula, viz.: conjuntivas, disyuntivas, universales y existenciales.

```
data Regla = Conj | Disy | Exis Simbolo | Univer Simbolo
```

Notar que las reglas de carácter existencial y universal necesitan ser acompañadas de un símbolo que indica el parámetro en el que se instancian.

Finalmente introducimos el concepto de demostración mediante el tipo Demostracion:

```
type Demostracion = [(Regla, Int)]
```

Una demostración es una secuencia ordenada (i.e. una lista) de pares compuestos por la regla a aplicar y el índice (comenzando desde el cero y numerando ascendentemente de izquierda a derecha) de la fórmula sobre la que se aplica en la rama. Luego podemos implementar una función esDemostracionValida tal que dado un árbol de tableaux y una demostración, verifique si dicha demostración cierra por completo el árbol.

```
esDemostracionValida :: ArbolTableaux -> Demostracion -> Bool
```

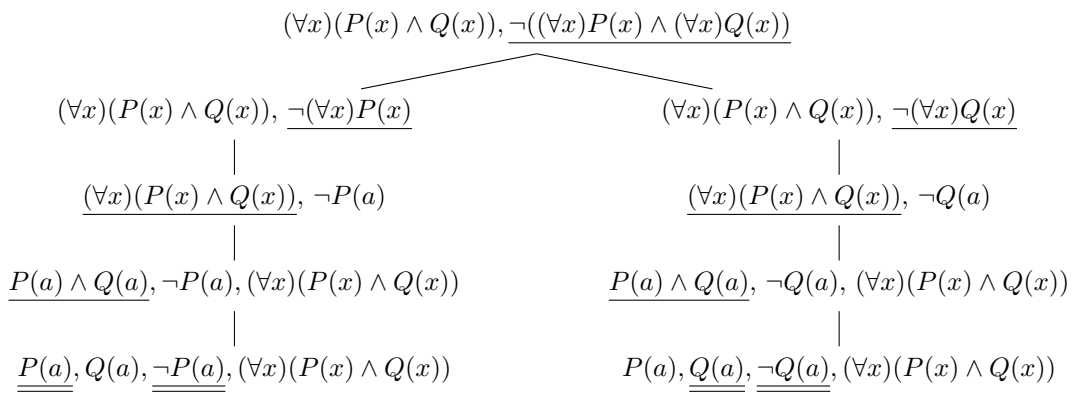
Esta función aplica en orden cada una de las reglas de la demostración y devuelve **True** si y sólo si todas las ramas del árbol se cierran. Para programarla, debemos primero definir ciertas consideraciones acerca de la aplicación de las reglas sobre el árbol:

(1) Las reglas se aplican siempre sobre la primera rama sin cerrar empezando desde la izquierda del árbol.
(2) Al cerrarse una rama, ésta se elimina de la lista de ramas del árbol.
(3) Al aplicarse una regla sobre una fórmula en una rama, se reemplaza ésta por el resultado de la aplicación de dicha regla en el orden definido en la tabla de reglas provista en el Anexo.

Como consecuencia de las consideraciones 1 y 2 tendremos que siempre aplicaremos las reglas de la demostración sobre el primer elemento de la lista de tipo ArbolTableaux, es decir sobre la primera rama no cerrada.

Si la demostración provista para el árbol es correcta entonces al consumir todas sus reglas no deberían quedar ramas abiertas, lo cual representamos en Haskell como un elemento de tipo ArbolTableaux sin ramas, i.e. una lista vacía.

Por ejemplo, considere el siguiente árbol de tableaux que demuestra la validez del razonamiento $(\forall x)(P(x) \wedge Q(x)) \vdash (\forall x)P(x) \wedge (\forall x)Q(x)$:



Esta demostración se codifica como una expresión d de tipo Demostracion de la siguiente manera:

```
d :: Demostracion
d = [(Disy, 1), (Exis "a", 1), (Univer "a", 0), (Conj, 0),
                (Exis "a", 1), (Univer "a", 0), (Conj, 0)]
```

?1. Observar el árbol provisto y verificar que las reglas de la lista de la expresión d se aplican en orden siempre sobre la primera rama sin cerrar comenzando desde la izquierda. Notar que el índice de cada regla indica la fórmula sobre la que se aplica la regla en la rama y que éste comienza a numerarse desde cero.

Al llamar a la función esDemostracionValida pasándole el estado inicial del árbol (una única rama con el conjunto de fórmulas formado por las premisas y la conclusión negada), ésta debería retornar True ya que la demostración es correcta, i.e. al efectuar las reglas en orden todas las ramas se cierran.

```
a :: ArbolTableaux
a = [[(All "x" (Bc And (A "P" [V "x"]) (A "Q" [V "x"]))),
      (Neg (Bc And (All "x" (A "P" [V "x"])) (All "x" (A "Q" [V "x"]))))]]
```

Con las definiciones anteriores debería cumplirse que esDemostracionValida a d = **True**.

Para implementar la función esDemostracionValida usamos algunas funciones auxiliares como efectuarDemostracion, aplicarRegla y verificarContradiccion .

?2. Estudiar la definición de las funciones antedichas en el template provisto junto a este trabajo y asegurarse de comprender en detalle su funcionamiento.

?3. Programar en Haskell las siguientes funciones que permiten aplicar las reglas sobre un árbol de tableaux. Para las funciones de las partes a, b, c, d y e en caso de no poderse aplicar la regla correspondiente, el programa deberá imprimir un error.

(a) appConjuntiva :: Form −> [Form]
Que recibe una fórmula a la cual se le puede aplicar una regla conjuntiva y devuelve con las fórmulas resultantes de aplicar dicha regla sobre la fórmula.

(b) appDisyuntiva :: Form −> ([Form], [Form])
Que recibe una fórmula a la cual se le puede aplicar una regla disyuntiva y devuelve un par ordenado con las fórmulas resultantes de aplicar dicha regla sobre la fórmula.

(c) appUniversal :: Simbolo −> Form −> [Form]
Que recibe el nombre de constante en el que se instanciará la regla, una fórmula y devuelve una lista con las fórmulas resultantes de aplicar la regla universal de tableaux.

Sugerencia: Implemente una función auxiliar que efectúe la sustitución de una variable por una constante en una fórmula.

(d) appExist :: Simbolo −> Form −> Form
Que recibe el nombre de constante en el que se instanciará la regla, una fórmula y devuelve la fórmula resultante de aplicar la regla existencial de tableaux.

(e) esNuevaConst :: Simbolo −> [Form] −> **Bool**
Que dado un nombre y una lista de fórmulas, determina si el nombre no aparece (en absoluto) en ninguna de las fórmulas de la lista.

(f) hayContradiccion ::[Form] −> **Bool**
Que determina si hay una contradicción en una rama, es decir si contiene un predicado y su negación aplicados a los mismos términos en el mismo orden.

?4. Verificar que la demostración propuesta anteriormente como ejemplo es efectivamente correcta.

?5. Indicar cuáles de los siguientes razonamientos sobre Lógica de Primer Orden son válidos. Para aquellos que los sean, codificar las expresiones de tipo ArbolTableaux y Demostracion tales que al pasárselas a la función esDemostracionValida ésta devuelva **True** (¡y verificarlo!). Para aquellos que no sean válidos, dejar las expresiones antedichas como **undefined**. Tenga en cuenta que junto con el código fuente deberá entregar un archivo PDF que contenga los árboles de demostración correspondientes a cada parte. En caso de ser inválido el razonamiento, dar una interpretación contraejemplo.

(a) $(\exists x)(P(x) \supset Q(x)) \vdash (\forall x)P(x) \supset (\exists x)Q(x)$
(b) $(\exists x)(P(x) \wedge Q(x)) \vdash (\exists x)P(x) \wedge (\exists x)Q(x)$
(c) $(\forall x)P(x) \supset (\forall x)Q(x), (\exists x)\neg Q(x) \vdash (\exists x)\neg P(x)$
(d) $(\exists x)(\forall y)(Q(x,y) \wedge Q(y,x) \supset P(x)), (\forall z)Q(z,c) \vdash (\forall z)Q(c,z) \supset P(c)$
(e) $(\forall x)(\forall y)(R(x,y) \supset \neg R(y,x)) \vdash (\forall x)\neg R(x,x)$
(f) $(\exists x)(\neg Q(x) \wedge P(x)), Q(b) \vdash (\exists x)\neg(P(x) \supset Q(x))$
(g) $(\forall x)(P(x) \vee Q(x)), Q(a) \vdash \neg(\forall x)P(x) \supset (\forall x)Q(x)$

Reglas de entrega. Para la obtención de los puntos del trabajo deberán cumplirse, sin excepción, las siguientes reglas:

- Debe realizarse en grupos de hasta dos estudiantes.
- Cada integrante del grupo debe subir un archivo Haskell (.hs) con el código fuente y un archivo PDF que contenga los árboles de tableaux correspondientes a la pregunta 5 a través de la plataforma de Aulas antes de las 23:55 del viernes 22/6/2016.
- **Advertencia sobre plagio:** La presentación por los estudiantes de cualquier parte del trabajo como original de la que no sean autores, supone un atentado a los derechos de propiedad intelectual, y será severamente penalizado por la universidad. Se ejecutará un software de verificación de autoría sobre todos los trabajos para detectar posibles plagios. En caso de sospecha de plagio el docente podrá solicitar a los involucrados que prueben su autoría.

Anexo. Reglas para el método de tableaux. Recuerde que su implementación, al aplicar las reglas, deberá respetar exactamente el mismo orden que se especifica a continuación:

Reglas Universales:

$$\begin{array}{c} (\forall x)(P(x)),\ \Gamma \\ \Big| \\ P(a),\ \Gamma,\ (\forall x)(P(x)) \end{array}$$

$$\begin{array}{c} \neg(\exists x)(P(x)),\ \Gamma \\ \Big| \\ \neg P(a),\ \Gamma,\ \neg(\exists x)(P(x)) \end{array}$$

Nota: luego de hacer uso de la regla universal, la fórmula que mantiene el cuantificador universal, deberá ser posicionada al final de todas las fórmulas de la rama.

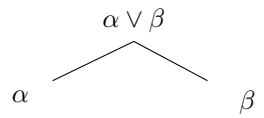
Reglas Existenciales:

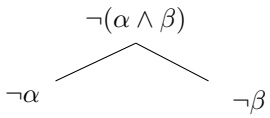
$$\begin{array}{c} (\exists x)(P(x)),\ \Gamma \\ \Big| \\ P(a),\ \Gamma \end{array}$$

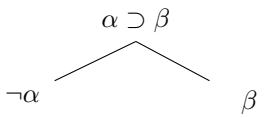
$$\begin{array}{c} \neg(\forall x)(P(x)),\ \Gamma \\ \Big| \\ \neg P(a),\ \Gamma \end{array}$$

Nota: Recordar que al instanciar la regla existencial se debe utilizar un nombre de constante nuevo (que no ocurra en la rama).

Reglas Disyuntivas:







Nota: el orden de las ramas generadas a partir de la regla de disyunción, deberá ser el especificado en el diagrama superior.

Reglas conjuntivas: