Lógica de Primer Orden: Introducción y Sintaxis
Lógica Computacional 2016-2, Nota de clase 4

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca

29 de febrero de 2016

1. Introducción

La lógica ha sido estudiada desde los tiempos de Aristóteles. En sus inicios se dedicó a fundamentar de manera sólida el análisis del razonamiento humano, formalizando las "leyes del pensamiento", es decir, las nociones de argumento válido o inválido. Aristóteles aisló principios lógicos llamados *silogismos* los cuales fueron utilizados para analizar y verificar argumentos hasta la edad media por lógicos como Peter Abelard o Guillermo de Ockham.

En el siglo 19 George Boole y otros desarrollaron versiones algebráicas de la lógica debido a las limitaciones de la teoría del silogismo para manejar relaciones binarias, por ejemplo "x es hijo de y". Esto dio nacimiento a la lógica algebraica, formalismo que aun existe aunque ha perdido popularidad.

Al final del siglo 19 Gottlob Frege desarrollo la lógica cuantificada sentando así los cimientos de gran parte de la lógica moderna, aún cuando Bertrand Russell le mostró que su sistema era inconsistente. Al mismo tiempo Charles S. Pierce de manera independiente, propuso una lógica similar. La *paradoja de Russell* generó una crisis en los fundamentos de la matemática la cual se resolvío al inicio del siglo XX usando la lógica de primer orden, la cual funge hasta la actualidad como un fundamento de las matemáticas modernas. Debido a su gran poder, la lógica de primer orden también tiene grandes limitaciones que fueron descubiertas en los años 1930 por Kurt Gödel, Alan Turing y Alonzo Church, entre otros. Gödel dio un sistema deductivo completo y correcto para la lógica de primer orden, Alfred Tarski le dio la semántica formal que estudiaremos y que es la base para el estudio de las semánticas denotacionales de los actuales lenguajes de programación y Gerhard Gentzen y otros desarrollarón la teoría de la demostración. A partir de este punto la lógica de primer orden toma su forma actual.

Durante la siguiente parte del curso nos dedicaremos a estudiar los siguientes aspectos de la lógica de primer orden:

- Su sintaxis y semántica formal.
- El proceso de especificación formal del español a la lógica.
- Los métodos para establecer válidez de argumentos:

- Argumentación semántica (método util pero prácticamente imposible de implementar de manera eficaz).
- Tableaux semánticos (método fácilmente implementable)
- Resolución binaria (método fundamental de la programación lógica)
- Deducción natural (método completo y correcto, fundamento de los sistemas de tipos para lenguajes de programación).

1.1. Lógica proposicional: expresividad vs. decidibilidad

- La lógica proposicional no es lo suficientemente expresiva. Considérese por ejemplo el siguiente razonamiento:

  Algunas personas van al teatro. Todos los que van al teatro se divierten. De manera que algunas personas se divierten.

  La intuición dice que el argumento es correcto. Sin embargo la representación correspondiente en lógica proposicional es:

  $$p,\ q\ /\ \therefore\ r \qquad \text{¡Incorrecto!}$$

  Otros ejemplos de expresiones que no pueden ser formalizadas adecuadamente en la lógica proposicional son:

  - La lista está ordenada.
  - Cualquier empleado tiene un jefe.
  - Si los caballos son animales entonces las cabezas de caballos son cabezas de animales.

  Este problema de expresividad se soluciona al introducir la lógica de predicados.

- La lógica proposicional por otro lado es decidible, es decir, existen diversos algoritmos para responder a las preguntas

  $$\text{¿}\models\varphi\text{?, ¿ Existe }\mathcal{I}\text{ tal que }\mathcal{I}\models\varphi\text{?, ¿ Es }\Gamma\text{ satisfacible?, ¿ }\Gamma\models\varphi\text{ ?}$$

  Debido a esto, su estudio es de gran importancia en la teoría de complejidad computacional (problema SAT). Esta propiedad, de gran importancia desde el punto de vista computacional, se pierde en la lógica de predicados.

Antes de empezar el estudio formal discutimos con detalle un ejemplo de especificación no trivial en lógica de proposiciones. De aquí podremos observar una aplicación directa del problema SAT fuera del ámbito de la lógica. Esta clase de representaciones son típicas de los métodos de reducción de problemas en teoría de la complejidad.

2. Reducción del Sudoku al problema SAT

Consideramos aquí Sudokus que cumplen las siguientes dos propiedades[1]:

- Tienen una solución única
- Pueden resolverse con razonamiento únicamente, es decir, sin búsqueda.

La representación se sirve de 729 variables proposicionales denotadas s_{xyz} con $x, y, z \in \{1, \ldots, 9\}$, cuyo significado es que el número z está en la entrada xy. Debe pensarse el tablero de Sudoku como una matriz de manera que la posición xy denota a la celda en el renglón x y columna y. La especificación es la siguiente:

- Hay al menos un número en cada celda

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

  Esta fórmula nos dice que para cada renglón x y para cada columna y hay un número z en la celda xy.

- Cada número figura a los más una vez en cada renglón

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$$

  Esta fórmula nos dice que para cada columna y, para cada número z y para cada renglón x, excepto el último, si el número z está en el renglón x (en la celda xy) entonces no sucede que el número z está en alguno de los renglones posteriores a x (los denotados por i)

- Cada número figura a los más una vez en cada columna

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$$

  Esta fórmula nos dice que para cada renglón x, para cada número z y para cada columna y, excepto la última, si el número z está en la columna y (en la celda xy) entonces no sucede que el número z está en alguna de las columnas posteriores a y (las denotadas por i)

- Cada número figura a lo más una vez en cada cuadrante de 3×3:
  - Dentro de cada cuadrante, si un número z está en un renglón entonces no está en las celdas siguientes en el mismo renglón. Aquí i, j delimitan las celdas válidas del cuadrante y k se encarga de verificar las columnas siguientes a la de la celda en consideración. Todo dentro del mismo cuadrante.

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$$

[1]Esta sección se basa en el artículo *Sudoku as a SAT Problem* de Inês Lynce y Joël Ouaknine. 9th International Symposium on Artificial Intelligence and Mathematics, January 2006.

- Dentro de cada cuadrante, si un número z está en una columna entonces no está en los renglones superiores ni en la misma columna ni en las columnas siguientes. Aquí i, j delimitan las celdas válidas del cuadrante, k se encarga de verificar los renglones superiores y l verifica la columna actual y las columnas siguientes.

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{l=1}^{3}(\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$$

Las cláusulas anteriores conforman una codificación minima. Un bonito cálculo combinatorio lleva a concluir que a partir de las fórmulas anteriores la fórmula resultante de la transformación a FNC tendrá 8829 cláusulas de las cuales 81 cláusulas tienen 9 literales, y las restantes 8748 son binarias, es decir, tienen 2 literales. Adicionalmente se deben considerar las cláusulas unitarias correspondientes a las entradas del Sudoku que ya tienen un número asignado.

Al agregar las siguientes cláusulas modelamos la unicidad de un número en cada celda:

- Hay a lo más un numero en cada celda.

$$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{8}\bigwedge_{i=z+1}^{9}(\neg s_{xyz} \vee \neg s_{xyi})$$

  Esta fórmula nos dice que para cada renglón x, para cada columna y, para cada número no 9, si la celda xy tiene al numero z entonces no tiene a los números mayores que z

- Cada número figura al menos una vez en cada columna

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{x=1}^{9} s_{xyz}$$

  Está fórmula nos dice que para cada columna y y cada número z, existe un renglón x, tal que z está en la celda xy.

- Cada número figura al menos una vez en cada renglón

$$\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{y=1}^{9} s_{xyz}$$

  Está fórmula nos dice que para cada renglón x y cada número z, existe una columna y, tal que z está en la celda xy.

- Cada número figura al menos una vez en cada cuadrante de 3×3.

$$\bigvee_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}$$

  Está fórmula nos dice que hay un número z en cada uno de los cuadrantes, los cuales qse generan por las celdas $(3i + x)(3j + y)$.

La fórmula resultante de transformar todas las anteriores consta de 11988 cláusulas, 324 de ellas con 9 literales y las restantes 11664 binarias. Nuevamente debemos agregar cláusulas unitarias para las entradas previamente asignadas del Sudoku.

3. Lógica de predicados de manera informal

En la lógica proposicional debemos representar a ciertas frases del español como

Algunos programas usan ciclos anidados

como fórmulas proposicionales atómicas, es decir mediante una simple variable proposicional p. En la lógica de predicados descomponemos a este tipo de frases distinguiendo dos categorias: objetos y relaciones entre objetos. Ambas categorias construidas sobre un universo fijo el cual varía según el problema particular que se desee representar. Estudiemos con mas detalle estas categorias:

3.1. Objetos y universo de discurso

Algunas personas *van al teatro*
Todos los programas *en Java usan clases.*

- Universo o dominio de discurso: conjunto de objetos que se consideran en el razonamiento, pueden ser cosas, personas, datos, objetos abstractos, etc.

- En el caso de los enunciados anteriores en el universo podemos tener personas y programas en Java, y posiblemente el teatro y las clases.

3.2. Relaciones entre objetos

- Las propiedades o relaciones atribuibles a los objetos del dominio de discurso se representan con predicados.

- En el caso de los enunciados anteriores tenemos *ir al teatro, usar clases.*

3.3. Cuantificadores

Los cuantificadores especifican el número de individuos que cumplen con el predicado. Por ejemplo:

- *Todos* los estudiantes trabajan duro.

- *Algunos* estudiantes se duermen en clase.

- *La mayoría* de los maestros están locos

- *Ocho de cada diez* gatos lo prefieren.

- *Nadie* es más tonto que yo.

- *Al menos seis* estudiantes están despiertos.

- *Hay una infinidad* de números primos.

- *Hay más* PCs que Macs.

En lógica de primer orden sólo consideraremos dos cuantificadores: *todos* y *algunos*.

3.4. Proposiciones vs. Predicados

El uso de predicados en lugar de proposiciones podría parecer simplemente otra manera de escribir las cosas, por ejemplo, la proposición:

```
Chubaka recita poesía nordica
```

se representa con predicados como

```
Recita(Chubaka, poesía nordica)
```

La gran ventaja es que el predicado puede cambiar de argumentos, por ejemplo

```
Recita(Licantro, odas en sánscrito)
```

o en el caso general podemos usar variables para denotar individuos:

```
Recita(x,y)
```

Obsérvese que está última expresión no es una proposición.
En este sentido un mismo predicado está representando un número potencialmente infinito de proposiciones.

Reconozcamos ahora las componentes anteriores en un argumento particular:

Todos los matemáticos estudian algún teorema. Los teoremas son elementales o trascendentes. Giuseppe es matemático. Luego entonces, Giuseppe estudia algo elemental o trascendente.

- Universo: *matemáticos y teoremas*
- Predicados: *estudiar, elemental, trascendente.*
- Cuantificadores: *todos, los, algo.*
- Los individuos se representarán formalmente mediante dos categorias
    - Constantes: las cuales denotan individuos particulares, como Giuseppe en el ejemplo.
    - Variables: las cuales denotan individuos genéricos o indeterminados como los matemáticos o el teorema en el ejemplo. **Obsérvese que** las variables no figuran en el lenguaje natural, pero son necesarias para la formalización.

Formalicemos ahora todos los conceptos discutidos anteriormente.

4. Sintáxis de la lógica de primer orden

En contraposición al lenguaje de la lógica proposicional que está determinado de manera única, no es posible hablar de un solo lenguaje para la lógica de predicados. Dependiendo de la estructura semántica que tengamos en mente será necesario agregar símbolos particulares para denotar objetos y relaciones entre objetos. De esta manera el alfabeto consta de dos partes ajenas entre si, la parte común a todos los lenguajes determinada por los símbolos lógicos y auxiliares y la parte particular, llamada tipo de semejanza o signatura del lenguaje.

- La parte común a todos los lenguajes consta de:
  - Un conjunto infinito de variables $\mathsf{Var} = \{x_1, \ldots, x_n, \ldots\}$
  - Constantes lógicas: $\bot, \top$
  - Conectivos u operadores lógicos: $\neg, \wedge, \vee, \to, \leftrightarrow$
  - Cuantificadores: $\forall, \exists$.
  - Símbolos auxiliares: (,) y , (coma).
  - Si se agrega el símbolo de igualdad $=$, decimos que el lenguaje tiene igualdad.
- La signatura de un lenguaje en particular está dada por:
  - Un conjunto $\mathcal{P}$, posiblemente vacío, de símbolos o letras de predicado:

    $$P_1, \ldots, P_n, \ldots$$

    A cada símbolo se le asigna un índice[2] o número de argumentos m, el cual se hace explícito escribiendo $P_n^{(m)}$ lo cual significará que el símbolo P_n necesita de m argumentos.
  - Un conjunto $\mathcal{F}$, posiblemente vacío, de símbolos o letras de función:

    $$f_1, \ldots, f_n, \ldots$$

    Análogamente a los símbolos de predicado cada símbolo de función tiene un índice asignado, $f_n^{(m)}$ significará que el símbolo f_n necesita de m argumentos.
  - Un conjunto $\mathcal{C}$, posiblemente vacío, de símbolos de constante:

    $$c_1, \ldots, c_n, \ldots$$

    En algunos libros los símbolos de constante se consideran como parte del conjunto de símbolos de función, puesto que pueden verse como funciones de índice cero, es decir, funciones que no reciben argumentos.

Dado que un lenguaje de primer orden queda determinado de manera única por su signatura, abusaremos de la notación y escribiremos

$$\mathcal{L} = \mathcal{P} \cup \mathcal{F} \cup \mathcal{C}$$

para denotar al lenguaje dado por tal signatura.

[2]Este índice suele llamarse también "aridad", pero dado que tal palabra no existe en el diccionario de la lengua española, trataremos de evitar su uso.

4.1. Términos

Los términos del lenguaje son aquellas expresiones que representarán objetos en la semántica y su definición es:

- Los símbolos de constante $c_1, \ldots, c_n, \ldots$ son términos.
- Las variables $x_1, \ldots, x_n, \ldots$ son términos.
- Si $f^{(m)}$ es un símbolo de función y $t_1, \ldots, t_m$ son términos entonces $f(t_1, \ldots, t_m)$ es un término.
- Son todos.

Es decir los términos están dados por la siguiente gramática:

$$t ::= x \mid c \mid f(t_1, \ldots, t_m)$$

El conjunto de términos de un lenguaje dado se denota con $\mathsf{TERM}_{\mathcal{L}}$, o simplemente TERM si es claro cual es el lenguaje.

En algunas ocasiones se encuentra una definición de términos que no incluye constantes. Esta omisión no existe en realidad puesto que las constantes se consideran casos particulares de símbolos de función de índice cero. Es decir, un símbolo de función que no recibe argumentos. Con esto en mente presentamos la siguiente implementación.

4.1.1. Implementación

Emplearemos la siguiente representación

```
type Nombre = String

data Term = V Nombre | F Nombre [Term]
```

Veamos algunos ejemplos:

- c se implementa como `F ‘‘c’’ [ ]`.
- $f(x, y)$ se implementa como `F ‘‘f’’ [V ‘‘x’’, V ‘‘y’’]`
- $g(a)$ se implementa como `F ‘‘g’’ [F ‘‘a’’ []]`
- $h(f(x))$ se implementa como `F ‘‘h’’ [F ‘‘f’’ [V ‘‘x’’]]`
- $h(b, f(a), z)$ se implementa como `F ‘‘h’’ [F ‘‘b’’ [], F ‘‘f’’ [F ‘‘a’’ []], V ‘‘z’’]`

4.2. Fórmulas

El siguiente paso es determinar las expresiones o fórmulas atómicas, dadas por:

- Las constantes lógicas $\bot, \top$.
- Las expresiones de la forma: $P_1(t_1, \ldots, t_n)$ donde $t_1, \ldots, t_n$ son términos
- Las expresiones de la forma $t_1 = t_2$, si el lenguaje cuenta con igualdad

El conjunto de expresiones atómicas se denotará con $\mathsf{ATOM}_{\mathcal{L}}$.

El conjunto $\mathsf{FORM}_{\mathcal{L}}$ de expresiones compuestas aceptadas en un lenguaje $\mathcal{L}$, llamadas usualmente fórmulas, se define recursivamente como sigue:

- Si $\varphi \in \mathsf{ATOM}_{\mathcal{L}}$ entonces $\varphi \in \mathsf{FORM}_{\mathcal{L}}$. Es decir, toda fórmula atómica es una fórmula.
- Si $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ entonces $(\neg\varphi) \in \mathsf{FORM}_{\mathcal{L}}$
- Si $\varphi, \psi \in \mathsf{FORM}_{\mathcal{L}}$ entonces $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi) \in \mathsf{FORM}_{\mathcal{L}}$.
- Si $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ y $x \in \mathsf{Var}$ entonces $(\forall x\varphi), (\exists x\varphi) \in \mathsf{FORM}_{\mathcal{L}}$.
- Son todas.

La gramática para las fórmulas en forma de Backus-Naur es:

$$\varphi ::= at \mid (\neg\varphi) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \to \psi) \mid (\varphi \leftrightarrow \psi) \mid (\forall x\varphi) \mid (\exists x\varphi)$$

$$at ::= \bot \mid \top \mid P(t_1, \ldots, t_m) \mid t_1 = t_2$$

Cada lenguaje definido a partir de una signatura dada de la manera recién descrita será un *lenguaje de primer orden.*

Las siguientes observaciones son de suma importancia:

- Los términos y las fórmulas son dos categorías sintácticas ajenas, es decir **ningún** término es fórmula y **ninguna** fórmula es término.
- Los términos denotan exclusivamente individuos u objetos.
- Las fórmulas atómicas denotan proposiciones o propiedades acerca de los términos.
- **Sólo** los individuos u objetos son cuantificables y **únicamente** sobre ellos se puede "predicar". Esta característica justifica la denominación "primer orden".

4.3. Precedencia

Si bien los cuantificadores no son propiamente operadores entre fórmulas y por lo tanto no se puede hablar de una precedencia en el sentido usual, usamos la siguiente convención:

Los cuantificadores se aplican a la mínima expresión sintácticamente posible.

de manera que

$$\begin{array}{rcl} \forall x\varphi \to \psi & \text{es} & (\forall x\varphi) \to \psi \\ \exists y\varphi \wedge \forall w\psi \to \chi & \text{es} & (\exists y\varphi) \wedge (\forall w\psi) \to \chi \end{array}$$

4.4. Implementación

```
data Form = TrueF | FalseF | Pr Nombre [Term] | Eq Term Term |
            Neg Form | Conj Form Form | Disy Form Form | Imp Form Form | Equi Form Form |
            All Nombre Form | Ex Nombre Form
```

Algunos ejemplos son:

- $P(x, y)$ se implementa como `Pr ‘‘P’’ [V ‘‘x’’, V ‘‘y’’]`
- $Q(a, h(w))$ se implementa como `Pr ‘‘Q’’ [F ‘‘a’’ [], F ‘‘h’’ [V ‘‘w’’]]`
- $R(x) \to x = b$ se implementa como

```
Imp (Pr ‘‘R’’ [V ‘‘x’’]) (Eq (V ‘‘x’’) (F ‘‘b’’ []))
```

- $\neg R(z) \vee (S(a) \wedge f(x) = z)$ se implementa como

```
Disy (Neg (Pr ‘‘R’’ [V ‘‘z’’]))
     (Conj (Pr ‘‘S’’ [F ‘‘a’’ []])
           (Eq (F ‘‘f’’ [V ‘‘x’’]) (V ‘‘z’’))
     )
```

- $\forall x Q(x, g(x))$ se implementa como

```
All ‘‘x’’ (Pr ‘‘Q’’ [V ‘‘x’’,F ‘‘g’’ [V ‘‘x’’]])
```

- $\forall x \exists y P(x, y)$ se implementa como

```
All ‘‘x’’ (Ex ‘‘y’’ (Pr ‘‘P’’ [V ‘‘x’’,V ‘‘y’’]))
```

5. Inducción y Recursión

Dado que ahora tenemos dos categorías sintácticas, términos y fórmulas tenemos también principios de inducción correspondientes a cada categoría.

Definición 1 (Principio de Inducción Estructural para $\mathsf{TERM}_{\mathcal{L}}$) *Sea $\mathcal{P}$ una propiedad acerca de términos. Para demostrar que $\mathcal{P}$ es válida para todos los términos, basta seguir los siguientes pasos:*

- *Base de la inducción: mostrar que*
  - *Si $x \in \mathsf{Var}$ entonces $\mathcal{P}$ es válida para x.*
  - *Si $c \in \mathcal{C}$ entonces $\mathcal{P}$ es válida para c.*
- *Hipótesis de inducción: suponer $\mathcal{P}$ para cualesquiera $t_1, \ldots, t_n \in \mathsf{TERM}_{\mathcal{L}}$.*
- *Paso inductivo: usando la H.I. mostrar que*
  - *Si $f \in \mathcal{F}$ es un símbolo de función de índice n entonces $f(t_1, \ldots, t_n)$ cumple $\mathcal{P}$.*

Definición 2 (Principio de Inducción Estructural para $\mathsf{FORM}_{\mathcal{L}}$) *Sea $\mathcal{P}$ una propiedad acerca de fórmulas. Para probar que toda fórmula $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ tiene la propiedad $\mathcal{P}$ basta seguir los siguientes pasos::*

- *Caso base: mostrar que toda fórmula atómica tiene la propiedad $\mathcal{P}$.*
- *Hipótesis de inducción: suponer que φ y ψ cumplen $\mathcal{P}$.*
- *Paso inductivo: mostrar usando la H.I. que*
  - *$(\neg\varphi)$ también cumple $\mathcal{P}$.*
  - *$(\varphi \star \psi)$ tiene la propiedad $\mathcal{P}$, donde $\star \in \{\rightarrow, \wedge, \vee, \leftrightarrow\}$*
  - *$\forall x\varphi$ y $\exists x\varphi$ cumplen $\mathcal{P}$.*

Con respecto a la recursión podemos definir funciones sobre los términos y las fórmulas con los siguientes principios:

- **Definición Recursiva para Términos**.Para definir una función $h : \mathsf{TERM}_{\mathcal{L}} \rightarrow A$, basta definir h como sigue:
  - Definir $h(x)$ para $x \in \mathsf{Var}$.
  - Definir $h(c)$ para cada constante $c \in \mathcal{C}$.
  - Suponiendo que $h(t_1), \ldots, h(t_n)$ están definidas, definir $h(f(t_1, \ldots, t_n))$ para cada símbolo de función $f \in \mathcal{F}$ de índice n, utilizando $h(t_1), \ldots, h(t_n)$.
- **Definición Recursiva para Fórmulas**.Para definir una función $h : \mathsf{FORM}_{\mathcal{L}} \rightarrow A$, basta definir h como sigue:
  - Definir h para cada fórmula atómica, es decir, definir $h(\bot), h(\top)$, $h(P(t_1, \ldots, t_n))$ y $h(t_1 = t_2)$ si el lenguaje tiene igualdad.

- Suponiendo definidas $h(\varphi)$ y $h(\psi)$, definir a partir de éllas a $h(\neg\varphi)$, $h(\varphi \vee \psi)$, $h(\varphi \wedge \psi)$, $h(\varphi \to \psi)$, $h(\varphi \leftrightarrow \psi)$, $h(\forall x\varphi)$ y $h(\exists x\varphi)$.

Ejemplo 5.1 Definimos el conjunto de subtérminos de un término t recursivamente como sigue:

- $\mathsf{Subt}(x) = \{x\}$
- $\mathsf{Subt}(c) = \{c\}$
- $\mathsf{Subt}(f(t_1, \ldots, t_n)) = \{f(t_1, \ldots, t_n)\} \cup \mathsf{Subt}(t_1) \cup \ldots \cup \mathsf{Subt}(t_n)$

Ejemplo 5.2 Definimos recursivamente el número de conectivos y cuantificadores de una fórmula, conocido como el peso de una fórmula, como sigue:

- $\mathsf{peso}(\bot) = \mathsf{peso}(\top) = 0$.
- $\mathsf{peso}(P(t_1, \ldots, t_n)) = 0$.
- $\mathsf{peso}(\neg\varphi) = \mathsf{peso}(\varphi) + 1$.
- $\mathsf{peso}(\varphi \wedge \psi) = \mathsf{peso}(\varphi) + \mathsf{peso}(\psi) + 1$
- $\mathsf{peso}(\forall x\varphi) = \mathsf{peso}(\varphi) + 1$

5.0.1. Ejercicio

Definir recursivamente e implementar las siguientes funciones:

- `consT :: Term ->[Nombre]` tal que `ConsT t` devuelve la lista con todos los nombres de constante que figuran en t.
- `varT :: Term ->[Nombre]` tal que `VarT t` devuelve la lista con todos los nombres de variables que figuran en t.
- `funT :: Term ->[Nombre]` tal que `FunT t` devuelve la lista con todos los nombres de función que figuran en t.
- Análogamente para el caso de las fórmulas: `consF, varF,funF:: Form ->[Nombre]`
- `subT :: Term ->[Term]` tal que `subT t` devuelve la lista de subtérminos de t.
- `subF :: Form ->[Form]` tal que `subF A` devuelve la lista de subfórmulas de A.
- `cuantF :: Form ->[Formm]` tal que `cuantF A` devuelve la lista de cuantificaciones de A, es decir, la lista de subfórmulas de A que son cuantificaciones.

6. Ligado y alcance

En lógica de predicados los cuantificadores $\forall x$ y $\exists x$ son ejemplos del fenómeno de ligado que también surge en la mayoría de lenguajes de programación. La idea general es que ciertas presencias de variables son presencias *ligadas* o simplemente *ligas*, cada una de las cuales se asocia con una expresión llamada su *alcance*.

A continuación damos las definiciones para el caso de los cuantificadores.

Definición 3 *Dada una cuantificación $\forall x\varphi$ o $\exists x\varphi$, la presencia de x en $\forall x$ o $\exists x$ es la variable que liga el cuantificador correspondiente, mientras que la fórmula φ se llama el alcance, ámbito o radio de acción de este cuantificador*

Definición 4 *Una presencia de la variable x en la fórmula φ está ligada o acotada si es la variable que liga a un cuantificador de φ o si figura en el alcance de un cuantificador $\forall x$ o $\exists x$ de φ. Si una presencia de la variable x en la fórmula φ no está ligada, decimos que está libre en φ.*

Ejemplo 6.1 Sea $\varphi = \forall x \exists z\big(Q(y,z) \vee R(z,x,y)\big) \wedge P(z,x)$.
El alcance del cuantificador $\forall$ es la fórmula $\exists z(Q(y,z) \vee R(z,x,y))$; el alcance del cuantificador $\exists$ es la fórmula $Q(y,z) \vee R(z,x,y)$. En φ hay tres presencias de x, las dos primeras ligadas y la última libre; las presencias de z son cuatro, ligadas las tres primeras y libre la última; finalmente las dos presencias de y son libres.

Definición 5 *Sea φ una fórmula. El conjunto de variables libres de φ, se denota $FV(\varphi)$. Es decir, $FV(\varphi) = \{x \in \mathsf{Var} \mid x \text{ figura libre en } \varphi\}$. La notación $\varphi(x_1, \ldots, x_n)$ quiere decir que $\{x_1, \ldots, x_n\} \subseteq FV(\varphi)$.*

Obsérvese que una misma variable x puede figurar tanto libre como ligada en una fórmula.

Definición 6 *Una fórmula φ es* cerrada *si no tiene variables libres, es decir, si $FV(\varphi) = \varnothing$. Una fórmula cerrada también se conoce como enunciado o sentencia.*

Definición 7 *Sea $\varphi(x_1, \ldots, x_n)$ una fórmula con $FV(\varphi) = \{x_1, \ldots, x_n\}$. La cerradura universal de φ, denotada $\forall\varphi$, es la fórmula $\forall x_1 \ldots \forall x_n\ \varphi$. La cerradura existencial de φ, denotada $\exists\varphi$ es la fórmula $\exists x_1 \ldots \exists x_n\ \varphi$.*

Obsérvese que una cerradura se obtiene cuantificando todas las variables libres de una fórmula.

6.1. Implementación

Se deja como ejercicio implementar todos los conceptos de esta sección mediante las siguientes funciones:

- Lista de variables libres: `fv :: Form ->[Nombre]`
- Lista de variables ligadas: `bv :: Form ->[Nombre]`
- Cerradura universal: `aCl :: Form ->Form`
- Cerradura existencial: `eCl :: Form ->Form`
- Alcances en una fórmula : `alcF :: Form ->[(Form,Form)]`, al tomar como entrada P esta función devuelve una lista de pares de la forma (A, B) donde A es una cuantificación que es subfórmula de P y B es el alcance de A.

7. Sustitución

La noción de sustitución en la lógica de predicados es más complicada que en la lógica proposicional debido a la existencia de términos y de variables ligadas. A diferencia de lo que sucede en lógica proposicional, donde una sustitución es sólo una operación textual sobre las fórmulas, en la lógica de predicados las sustituciones son operaciones sobre los términos y sobre las fórmulas y en este último caso deben respetar el ligado de variables, por lo tanto no son operaciones textuales.

Definición 8 *Una sustitución en un lenguaje de predicados $\mathcal{L}$ es una tupla de variables y términos denotada como*

$$[x_1, x_2, \ldots, x_n := t_1, \ldots, x_n]$$

donde

- $x_1, \ldots, x_n$ *son variables distintas.*
- $t_1, \ldots, t_n$ *son términos de* $\mathcal{L}$
- $x_i \neq t_i$ *para cada* $1 \leq i \leq n$

Por lo general denotaremos a una sustitución con $[\vec{x} := \vec{t}]$

7.1. Implementación

Representaremos la sustitución $[\vec{x} := \vec{t}]$ mediante listas de pares

$$[(x_1, t_1), \ldots, (x_n, t_n)]$$

Para esto definimos el tipo de sustituciones:

```
type Subst = [(Nombre,Term)]
```

Obsérvese que esto no garantiza que si s : *Subst* entonces s sea una sustitución legal. Por lo tanto debe escribirse una función `verifSus:: Subst ->Bool` que verifique si la lista dada es una sustitución.

7.2. Aplicación de una sustitución a un término

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a un término r, denotada $r[\vec{x} := \vec{t}]$ se define como el término obtenido al reemplazar simultaneamente todas las presencias de x_i en r por t_i. Este proceso de define recursivamente como sigue:

$$\begin{aligned}
x_i[\vec{x} := \vec{t}] &= t_i \;\; 1 \leq i \leq n \\
z[\vec{x} := \vec{t}] &= z \;\; \text{si } z \neq x_i \; 1 \leq i \leq n \\
c[\vec{x} := \vec{t}] &= c \;\; \text{si } c \in \mathcal{C}, \text{ es decir, } c \text{ constante} \\
f(t_1, \ldots, t_m)[\vec{x} := \vec{t}] &= f(t_1[\vec{x} := \vec{t}], \ldots, t_m[\vec{x} := \vec{t}]) \qquad \text{con } f^{(m)} \in \mathcal{F}.
\end{aligned}$$

Obsérvese entonces que la aplicación de una sustitución a término es simplemente una sustitución textual tal y como sucede en lógica de proposiciones.

7.3. Implementación

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a un término t se implementa mediante una función

```
apsubT :: Term -> Subst -> Term
```

tal que si `s` implementa a la substitución $[\vec{x} := \vec{t}]$ entonces `apsubT r s` debe devolver el término $r[\vec{x} := \vec{t}]$.

7.4. Sustitución en Fórmulas

La aplicación de sustituciones a fórmulas, necesita de ciertos cuidados debido a la presencia de variables ligadas mediante cuantificadores. La aplicación de una sustitución textual a una fórmula puede llevar a situaciones problemáticas con respecto a la sintaxis y a la semántica de la lógica. En particular deben evitarse los siguientes problemas:

- *Sustitución de variables ligadas*: la definición de sustitución textual

  $$(\forall y\varphi)[\vec{x} := \vec{t}] =_{def} \forall(y[\vec{x} := \vec{t}])(\varphi[\vec{x} := \vec{t}])$$

  obliga a sustituir todas las variables de una fórmula, pero tal definición genera expresiones que ni siquiera son fórmulas, por ejemplo tendríamos que

  $$\big(\forall x P(y, f(x))\big)[x, y := g(y), z] = \forall g(y) P\big(z, f(g(y))\big)$$

  y la expresión de la derecha no es una fórmula puesto que la cuantificación sobre términos que no sean variables no está permitida. Lo mismo sucederá al definir la sustitución de esta manera para una fórmula existencial. En conclusión, la sustitución en fórmulas **NO** debe ser una sustitución textual puesto que las presencias ligadas de variables no pueden sustituirse.

  La solución inmediata al problema anterior consiste en definir la sustitución sólamente sobre variables libres, como sigue

  $$(\forall x\varphi)[\vec{x} := \vec{t}] =_{def} \forall x(\varphi[\vec{x} := \vec{t}\,]_x)$$

  donde $[\vec{x} := \vec{t}\,]_x$ denota a la sustitución que elimina al par $x := t$ de $[\vec{x} := \vec{t}]$. Por ejemplo $[z, y, x := y, f(x), c]_x = [z, y := y, f(x)]$ Así la aplicación de sustitución para la fórmula del primer ejemplo se ejecuta como sigue:

  $$\begin{array}{rcl} \big(\forall x P(y, f(x))\big)[x, y := g(y), z] & = & \forall x\big(P(y, f(x)[x, y := g(y), z]_x\big) \\ & = & \forall x\big(P(y, f(x))[y := z]\big) \\ & = & \forall x P\big(z, f(x)\big) \end{array}$$

  De esta manera se garantiza que el resultado de aplicar una sustitución a una cuantificación siempre será una fórmula. Análogamente podemos definir la sustitución para una fórmula existencial. Pasemos ahora a ver el segundo problema a evitar.

- **Captura de variables libres**: Si bien la solución dada al problema anterior es suficiente desde el punto de vista sintáctico, no lo es al entrar en juego la semántica.
  Considerese la fórmula $\forall x\varphi$, la semántica intuitiva nos dice que está fórmula será cierta siempre

y cuando $\varphi(x)$ sea cierta para cualquier valor posible de x, de manera que nos gustaría poder obtener, a partir de la de verdad $\forall x\varphi$, la verdad de cualquier aplicación de sustitución $\varphi[x := t]$. Veamos ahora que que sucede si consideramos la fórmula

$$\forall x\big(\exists y(\, x \neq y)\big)$$

Está fórmula expresa el hecho de que para cualquier individuo existe otro distinto de él y es cierta si el universo tiene al menos dos elementos. Ahora bien, al momento de calcular el caso particular $\big(\exists y(x \neq y)\big)[x := y]$ con la definición anterior de aplicación de sustituciones obtenemos:

$$\big(\exists y\,(x \neq y)\big)[x := y] = \exists y\big(\,(x \neq y)[x := y]_y\big) = \exists y\big(\,(x \neq y)[x := y]\big) = \exists y\,(y \neq y)$$

Pero esta última fórmula está diciendo que hay un objeto y que es distinto de si mismo lo cual es absurdo. De manera que con la definición dada no podemos garantizar la verdad de un caso particular de φ aun suponiendo la verdad de la cuantificación universal $\forall x\varphi$. ¿ Qué es lo que anda mal? obsérvese que la presencia libre de x en la fórmula $\exists y(x \neq y)$ se ligó al aplicar la sustitución $[x := y]$. Así que una variable libre que representa a un objeto particular se sustituyó por una variable que representa al objeto cuya existencia se está asegurando. Esta clase de sustituciones son inadmisibles, las posiciones que corresponden a una presencia libre de una variable representan a objetos particulares y el permitir que se liguen después de una sustitución modificará el significado intensional de la fórmula.

Existen dos maneras de solucionar el problema, la primera es aceptando la definición dada arriba mediante el uso de sustituciones de la forma $[\vec{x} := \vec{t}]_x$ pero prohibiendo la aplicación de sustituciones que liguen posiciones libres de variables. Para esto debe darse una una definición sustitución admisible para una fórmula, este método es el más usado en textos de lógica matemática.

Nosotros preferimos el segundo método, utilizado generalmente en la teoría de lenguajes de programación. La aplicación de una sustitución a una fórmula se define renombrando variables ligadas de manera que siempre podremos obtener una sustitución admisible.

Definición 9 *La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a una fórmula φ, denotada $\varphi[\vec{x} := \vec{t}]$ se define como la fórmula obtenida al reemplazar simultaneamente todas las presencias libres de x_i en φ por t_i, verificando que este proceso no capture posiciones de variables libres.*

La aplicación de una sustitución a una fórmula $\varphi[\vec{x} := \vec{t}\,]$ se define recursivamente como sigue

$$\begin{array}{rcl}
\bot[\vec{x} := \vec{t}\,] & = & \bot \\
\top[\vec{x} := \vec{t}\,] & = & \top \\
P(t_1, \ldots, t_m)[\vec{x} := \vec{t}\,] & = & P\big(t_1[\vec{x} := \vec{t}\,], \ldots, t_m[\vec{x} := \vec{t}\,]\big) \\
(t_1 = t_2)[\vec{x} := \vec{t}\,] & = & t_1[\vec{x} := \vec{t}\,] = t_2[\vec{x} := \vec{t}\,]
\end{array}$$

$$\begin{array}{rcl}
(\neg\varphi)[\vec{x} := \vec{t}\,] & = & \neg\big(\varphi[\vec{x} := \vec{t}\,]\big) \\
(\varphi \wedge \psi)[\vec{x} := \vec{t}\,] & = & \big(\varphi[\vec{x} := \vec{t}\,] \wedge \psi[\vec{x} := \vec{t}\,]\big) \\
(\varphi \vee \psi)[\vec{x} := \vec{t}\,] & = & \big(\varphi[\vec{x} := \vec{t}\,] \vee \psi[\vec{x} := \vec{t}\,]\big) \\
(\varphi \to \psi)[\vec{x} := \vec{t}\,] & = & \big(\varphi[\vec{x} := \vec{t}\,] \to \psi[\vec{x} := \vec{t}\,]\big) \\
(\varphi \leftrightarrow \psi)[\vec{x} := \vec{t}\,] & = & \big(\varphi[\vec{x} := \vec{t}\,] \leftrightarrow \psi[\vec{x} := \vec{t}\,]\big)
\end{array}$$

$$\begin{array}{rcl}
(\forall y\varphi)[\vec{x} := \vec{t}\,] & = & \forall y\big(\varphi[\vec{x} := \vec{t}\,]\big) \textbf{ si } y \notin \vec{x} \cup Var(\vec{t}) \\
(\exists y\varphi)[\vec{x} := \vec{t}\,] & = & \exists y\big(\varphi[\vec{x} := \vec{t}\,]\big) \textbf{ si } y \notin \vec{x} \cup Var(\vec{t})
\end{array}$$

Mencionamos ahora algunas propiedades de la sustitución que pueden verificarse mediante inducción sobre las fórmulas.

Proposición 1 *La operación de sustitución tiene las siguientes propiedades, considerando $x \notin \vec{x}$.*

- *Si $x \notin FV(\varphi)$ entonces $\varphi[\vec{x}, x := \vec{t}, r\,] = \varphi[\vec{x} := \vec{t}\,]$.*
- *Si $\vec{x} \cap FV(\varphi) = \varnothing$ entonces $\varphi[\vec{x} := \vec{t}\,] = \varphi$*
- *$FV(\varphi[\vec{x} := \vec{t}\,]) \subseteq (FV(\varphi) - \vec{x}) \cup FV(\vec{t})$*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $x \in FV(\varphi)$ si y sólo si $x \in FV(\varphi[\vec{x} := \vec{t}\,])$*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $\varphi[x := t][\vec{x} := \vec{t}\,] = \varphi\big[\vec{x} := \vec{t}\,\big]\big[x := t[\vec{x} := \vec{t}\,]\big]$*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $\varphi[\vec{x}, x := \vec{t}, t] = \varphi[\vec{x} := \vec{t}\,][x := t]$*

7.5. Implementación

La aplicación de una sustitución $[\vec{x} := \vec{t}\,]$ a una fórmula A se implementa mediante una función

```
apsubF :: Form -> Subst -> Form
```

tal que si `s` implementa a la substitución $[\vec{x} := \vec{t}\,]$ entonces `apsubF A s` debe devolver la fórmula $A[\vec{x} := \vec{t}\,]$.

En el caso de una sustitución en una cuantificación se debe verificar la condición de variables y de no cumplirse convenimos en devolver la fórmula de entrada tal cual. Por ejemplo el resultado de $(\forall x P(x, y))[y := f(x)]$ debe ser $\forall x P(x, y)$

7.6. La relación de α-equivalencia

La definición de sustitución en fórmulas cuenta con una restricción aparente en el caso de los cuantificadores, por ejemplo, la sustitución

$$\forall x(Q(x) \to R(z,x))[z := f(x)]$$

no está definida puesto que x figura en $f(x)$, es decir, $x \in Var(f(x))$, con lo que no se cumple la condición necesaria para aplicar la sustitución. Por lo tanto la aplicación de una sustitución a una fórmula es una función parcial.

Esta aparente restricción desaparece al observar que los nombres de las variables ligadas no importan por ejemplo, las fórmula $\forall x P(x)$ y $\forall y P(y)$ significan exactamente lo mismo, a saber que todos cumplen la propiedad P.

Por lo tanto convenimos en identificar fórmulas que sólo difieren en sus variables ligadas, esto se hace formalmente mediante la llamada relación de α-equivalencia definida como sigue:

Definición 10 *Decimos que dos fórmulas φ_1, φ_2 son α-equivalentes y escribimos $\varphi_1 \sim_\alpha \varphi_2$ si y sólo si φ_1 y φ_2 difieren a lo más en los nombres de sus variables ligadas.*

Por ejemplo las siguientes expresiones son α-equivalentes.

$$\forall x P(x,y) \to \exists y R(x,y,z) \sim_\alpha \forall w P(w,y) \to \exists v R(x,v,z) \sim_\alpha \forall z P(z,y) \to \exists u R(x,u,z)$$

Más tarde demostraremos que las fórmulas α-equivalentes también son lógicamente equivalentes y por lo tanto son intercambiables en cualquier contexto o bajo cualquier operación.

Usando la α-equivalencia, la operación de sustitución en fórmulas se vuelve una función total por lo que siempre está definida. Por ejemplo se tiene que

$$\forall x(Q(x) \to R(z,x))[z := f(x)] = \forall y(Q(y) \to R(z,y))[z := f(x)] = \forall y(Q(y) \to R(f(x),y))$$

Veamos otros ejemplos

Ejemplo 7.1 Las siguientes sustituciones se sirven, en caso necesario, de la α-equivalencia

$$\begin{aligned}
\forall x R(x,y,f(z))[y,z := d, g(w)] &= \forall x R(x,d,f(g(w))) \\
\forall x P(x,y)[y,z := f(c), w] &= \forall x P(x,f(c)) \\
Q(y,z,y)[y,z := g(d), f(y)] &= Q(g(d),f(y),g(d)) \\
\exists w Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(g(d),f(y),g(d)) \\
\exists y Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(w,f(y),w) \\
\exists z Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(g(d),w,g(d)) \\
\forall x\big(Q(z,y,x) \wedge \exists z T(f(z),w,y)\big)[x,y,z := a,z,g(w)] &= \forall u\big(Q(g(w),z,u) \wedge \exists v T(f(v),w,z)\big) \\
\forall x\big(Q(z,y,x) \wedge \exists z T(f(z),w,y)\big)[z,w,y := g(x),h(z),w] &= \forall u\big(Q(g(x),w,u) \wedge \exists v T(f(v),h(z),w)\big)
\end{aligned}$$

7.7. Implementación

Se deja como ejercicio implementar un test para la α-equivalencia así como redefinir la función de sustitución usando el renombre de variables ligadas. Estos ejercicios son más complicados que los anteriores.

- `vAlfaEq:: Form ->Form ->Bool` verifica si dos fórmulas son α-equivalentes.
- `renVL:: Form ->Form` renombra las variables ligadas de una fórmula de manera que las listas de variables libres y ligadas sean ajenas. Esto es un caso particular de la siguiente función.
- `renVLconj :: Form ->[Nombre] ->Form` renombra las variables ligadas de una fórmula de forma que sus nombres sean ajenos a los de una lista dada.
- `apSubF2 :: Form ->Subst ->Form` que implemente la sustitución en fórmulas usando la α-equivalencia.