Práctica 2

Lógica Computacional, 2018-2
Facultad de Ciencias, UNAM

Noé Salomón Hernández Sánchez
no.hernan@gmail.com

María del Carmen Sánchez Almanza
carmensanchez@ciencias.unam.mx

Albert Manuel Orozco Camacho
alorozco53@ciencias.unam.mx

1 de abril de 2018

1. Objetivo

La riqueza del lenguaje natural pone una limitante natural en la expresividad de la lógica proposicional: resulta poco ortodoxo manejar algunos *cuantificadores* (todos, algunos, cada, solo) así como aspectos más generales de cualquier oración declarativa. El uso de un *cálculo* de **predicados** nos permite incluir en nuestro conepto semántico el cumplimiento, o no, de las propiedades de un universo.

De esta manera, contamos con un marco lógico-formal en el cual importa tanto el significado real de cada propiedad en un universo, como la construcción sintáctica de un argumento. En este contexto, Haskell nos permite codificar de manera elegante la noción de un *predicado* P, como una función binaria (a saber, *booleana*) la cual se satisface cada vez que los argumentos estén relacionados entre sí, de acuerdo a P.

2. Ejercicios

El esqueleto de código del ejercicio semanal se encuentra en . Considere el siguiente tipo de datos, en el que se implementan los *términos* de la lógica de predicados:

```
data Term = VarP String
          | Funct String [Term]
          deriving Show
```

Tenemos dos constructores: `VarP` dedicado para variables y `Funct` dedicado para funciones (con nombre y argumentos). Obsérvese que si la lista `[Term]` de una función es nula, entonces estaremos refiriéndonos a una **constante**. Ahora, considere el siguiente tipo de datos para fórmulas de la lógica de predicados:

```
data Predicate = Pred String [Term]
               | Neg Predicate
               | Conj Predicate Predicate
               | Disj Predicate Predicate
               | Imp Predicate Predicate
               | Equiv Predicate Predicate
               | All String Predicate
               | Ex String Predicate
               deriving Show
```

El constructor `Pred` codifica un predicado mediante un nombre (`String`) y una lista de argumentos (`[Term]`). Aquí lo importante a destacar es que dicha lista ***nunca*** debe de ser nula, por definición. Por otro lado, los operadores lógicos se implementan análogamente a lo que se vio en la Práctica 1, mientras que los cuantificadores se especifican mediante una variable (*ligada*) al predicado que tienen como segundo argumento.

2.1. Sintaxis

Considere el tipo

```
type Sub = (String, Term)
```

para una sustitución en términos de la lógica de primer orden. Por ejemplo, la sustitución

$$[x := f(a)]$$

se implementaría como una función `sub1 :: Sub` de Haskell, donde

```
sub1 :: Sub
sub1 = ("x", Funct "f" [Funct "a" []])
```

1. Dé una función `varsT` que, dado un término, devuelva una lista que contenga todas sus variables.

2. Dé una función `varsP` que, dada una fórmula de primer orden, devuelva una lista que contenga todas sus variables, incluyendo ligadas y no ligadas.

3. Escriba una función `subTerm` que dado un término t y una sustitución s, aplique s en t. Como sugerencia, tome en cuenta la definición presentada en la Sección 2.6.1 de la Nota 06 del curso.

4. Implemente una función `subPred` que aplique una sustitución s en una fórmula de primer orden ψ. Es importante considerar que las variables ligadas de ψ no deben de ser sustituídas, así como no repetir variables ligadas al aplicar sustituciones en variables no ligadas. Por ello, se recomienda seguir la definición presente en la Sección 2.6.2 de la Nota 06, la cual resuelve los problemas previamente mencionados mediante la introducción de *variables nuevas*. (Para esto, se necesitaría realizar una función que obtenga todas las variables de una fórmula y otra que genere una variable que no haya sido utilizada previamente.)

2.2. Semántica

A continuación, mostramos los tipos necesarios para la implementación de la semántica de la lógica de predicados. Éstos se basan totalmente en la explicación de la Nota 08 del curso.

```
type Universe a = [a]
type Ambient a = String -> a
type FunctCtx a = String -> [a] -> a
type PredCtx a = String -> [a] -> Bool
type Model a = (Universe a, Ambient a, FunctCtx a, PredCtx a)
```

Implementaremos un universo como una lista de elementos de un *tipo* dado (a). Dicho tipo nos servirá como ancla para definir las demás partes de un modelo:

- un **ambiente** es un mapeo entre variables y elementos de tipo a;
- un **contexto de funciones** relaciona *nombres de funciones* y sus respectivas listas de argumentos con elementos de tipo a;
- un **contexto de predicados** relaciona *nombres de predicados* y sus respectivas listas de argumentos con valores de verdad (`Bool`);
- finalmente, un **modelo** es una 4-tupla formada por un universo, un ambiente, y contextos de funciones y predicados.

Obsérvese que la introducción de contextos de funciones y predicados permite no solamente definir los nombres válidos para cada uno de éstos, sino que también definir su comportamiento. Es importante señalar, que las listas de argumentos de tipo `[a]` implican que, forzozamente, *todos* los argumentos de cada predicado o función deben de ser previamente interpretados antes de interpretar el predicado o función en cuestión.

5. Escriba una función `interpTerm` que, dado un término t y un modelo $\mathcal{M}$, realice la interpretación de las variables de t, de acuerdo al ambiente descrito en $\mathcal{M}$. Por otro lado, deberá de interpretar cada función (y, por ende, cada constante) en t de acuerdo al contexto de funciones descrito en $\mathcal{M}$.

6. Dé una función `interpPred` que, dada una fórmula de primer orden ψ y un modelo $\mathcal{M}$, decida su satisfacibilidad $\mathcal{M} \models \psi$. Para llevar esto a cabo, la implementación sugerida se debería de basar en la Definición 3 de la Nota 08 del curso. Tome en cuenta que, para el caso de los cuantificadores, la interpretación requiere la realización de nuevos ambientes, a saber, $l[x \mapsto a]$ donde l es el ambiente dado en el modelo $\mathcal{M}$, x es la variable cuantificada y a es una elemento del universo. Por ello se sugiere, revisar el concepto de *funciones anónimas* (lambdas), por ejemplo, de `éste tutorial`.

3. Entrega

La fecha de entrega es el próximo **jueves 12 de abril de 2018** por la plataforma de *Google Classroom* del curso y siguiendo los lineamientos del laboratorio.