Lógica Computacional, 2017-2

Práctica 2: Gramáticas / Lógica proposicional

Manuel Soto Romero Víctor Zamora Gutiérrez

Fecha de inicio: 15 de febrero de 2017

Fecha de término: 1 de marzo de 2017

Instrucciones

- Completar de manera clara y ordenada las funciones de los archivos `ea.hs` y `prop.hs`.
- Para tener derecho a calificación, la práctica debe ejecutarse sin errores ni advertencias. No está permitido utilizar primitivas de Haskell que resuelvan directamente los ejercicios, ni modificar la firma de ninguna función.
- La entrega es por **equipos de 3 a 4 integrantes**. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/logica-computacional-2017-2/laboratorio/lineamientos`.

Parte I: Gramáticas

Ejercicios

1. **Expresiones aritméticas**

   Anexo a esta práctica, se encuentra un archivo `ea.hs` que contiene la definición de varias gramáticas y sinónimos para el guión que permite trabajar con expresiones aritméticas.

   a) Completar el cuerpo de la función `showEA :: EA -> String` que toma una expresión aritmética y regresa su representación como cadena. Los símbolos para los operadores son: `+`, `-`, `*` y `/`. Ejemplos:

```
> showEA (Var A)
A
> showEA (Mul (Cte (Suc (Suc Cero))) (Var B))
2 * B
> showEA (Sum (Var X) (Var Y))
X + Y
> showEA (Sum (Paren (Sum (Var X) (Var Y))) (Cte (Suc Cero)))
(X + Y) + 1
```

   b) Completar el cuerpo de la función `evalua :: EA -> Env -> Int` que toma una expresión aritmética y la evalúa como el entero que la representa. Para evaluar las expresiones se pasa como segundo parámetro un ambiente de evaluación que le indica a la función el valor de las variables. Ejemplos:

```
> evalua (Var A) [(A, 2)]
2
> evalua (Mul (Cte (Suc (Suc Cero))) (Var B)) [(A, 3),(D,8), (B, 10),(E,2)]
20
> evalua (Sum (Var X) (Var Y)) []
Exception: No se encontró la variable en el ambiente
```

2. **Árboles**

   En el ejercicio semanal 1 se definieron los árboles binarios (AB) cuyos únicos nodos etiquetados (con elementos del conjunto C) son sus hojas y se dio una definición recursiva de los mismos.

   El ejercicio consiste en definir estos árboles como una gramática `AB` en Haskell dentro del archivo `arboles.hs`. El tipo de dato debe tener los siguientes constructores:

   - El constructor `Hoja a`.
   - El constructor `Mkt (AB a) (AB a)`.

   Los árboles deben almacenar datos de cualquier tipo.

   Una vez definida la gramática se deben completar las siguientes funciones:

   a) Completar las funciones `nh`, `nni`, `elemA` e `inorderA` que se resolvieron en el ejercicio semanal.

   b) Completar la función `agregaHoja :: AB a -> a -> AB a` que agrega una hoja al árbol.

   c) Completar la función `mapA :: AB a -> (a -> b) -> AB b` que aplica la función recibida a cada elemento del árbol.

   d) Completar la función `profundidad :: AB a -> Int` que regresa la profundidad del árbol.

Parte II: El lenguaje PROP

Para esta parte se anexa el archivo `prop.hs` donde se encuentran definidas las gramáticas necesarias para resolver cada ejercicio y que fueron explicadas en clase.

Ejercicios

1. **Sustitución simultánea**

   Completar la función `sustSimult` para que realice una sustitución simultánea de variables. Ejemplos:

```
> sustSimult (Neg (FA (Var P))) [(Q,(FA (Cte V))), (P, (Op (FA (Var R)) Conj (FA
(Var S))))]
2
Neg (Op (FA (Var R)) Conj (FA (Var S)))
```

2. **Interpretación**

   Completar la función `interpreta` para que regrese el valor de la función de interpretación aplicada a una fórmula en los estados recibidos como parámetros. Ejemplos:

```
> interpreta (Neg (FA (Var P))) [(P, V)]
F
```

3. **Simplificación**

   Completar la función simplifica para que elimine:

   - Dobles negaciones.

- Disyunciones o conjunciones de la misma variable.
- Disyunciones con constantes. Ejemplos:

```
> simplifica (Neg (Neg (FA (Var P))))
FA (Var P)
> simplifica (Op (FA (Var P)) Conj (FA (Var P)))
FA (Var P)
> simplifica (Op (FA (Var P)) Conj (FA (Cte V)))
Fa (Var P)
> simplifica (Op (FA (Cte F)) Disy (FA (Var P)))
FA (Var P)
```

4. **Formas normales**

   Completar el cuerpo de las funciones `formaNN` y `formaNC` para que regresen la forma normal negativa de una expresión y la forma normal conjuntiva de una expresión respectivamente. Ejemplos:

```
> formaNN (Neg (Op (FA (Var P)) Impl (FA (Var q))))
Op (FA (Var P)) Conj (Neg (FA (Var Q)))
> formaNC (Neg (Op (FA (Var P)) Impl (FA (Var Q))))
Op (FA (Var P)) Conj (Neg (FA (Var Q)))
```

5. **Tautologías**

   Completar el cuerpo de la función `esTautologia` para que verifique si una fórmula es tautología. Ejemplos:

```
> esTautologia (Op (FA (Var P)) Impl (FA (Var Q)))
F
> esTautologia (Op (FA (Var P)) Imp (FA (Var P)))
V
```

6. **Satisfacibilidad**

   Completar el cuerpo de la función `esSatisfacible` para que decida si una fórmula es satisfacible. Ejemplos:

```
> esSatisfacible (Op (FA (Var P)) Impl (FA (Var Q)))
V
> esSatisfacible (Op (FA (Var P)) Conj (Neg (FA (Var P))))
F
```

7. **Cláusulas**

   Completar el cuerpo de la función `clausulas` para que obtenga las cláusulas de una fórmula. Ejemplos:

```
> clausulas (Op (FA (Var P)) Syss (FA (Var Q)))
[Op (Neg (FA (Var P))) Disy (FA (Var Q)), Op (Neg (FA (Var Q))) Disy (FA (Var
P))]
```