Autómatas y Lenguajes Formales 2019-2
Facultad de Ciencias UNAM
Ejercicio Práctico 1: Pertenencia a un Lenguaje.

Favio E. Miranda Perea Javier Enríquez Mendoza

6 de Marzo del 2019
Fecha de entrega: 19 de Marzo del 2019

1. Objetivo

Se busca crear un programa en el lenguaje de programación Haskell que dada una cadena w y una expresión regular α, decida si $w \epsilon L[\![\alpha]\!]$.

Para esto usaremos dos técnicas, denotación de expresiones regulares y derivadas de lenguajes. Ambos temas se trataron en las clases de teoría y se puede encontrar una descripción detallada de éstos en las notas del curso.

Para éste ejercicio usaremos la siguiente definición del tipo de dato algebraico `Regex` que representa las expresiones regulares.

```
data Regex = Void
           | Epsilon
           | Symbol Char
           | Star Regex
           | Concat Regex Regex
           | Add Regex Regex
          deriving (Eq)
```

También se usará el sinónimo `Language` para representar un lenguaje como una lista de cadenas.

```
type Language = [String]
```

2. Denotación

En está sección calcularemos el lenguaje denotado por α y verificaremos si w pertenece al lenguaje.

1. Definir la función `simpl` que recibe una `Regex` y la simplifica usando las equivalencias vistas en clase y los axiomas de Salomaa.

   ```
   simpl :: Regex -> Regex
   ```

   ```
   > simpl (Star (Add (Symbol 'a') Void))
   a*
   > simpl (Concat Epsilon (Add (Symbol 'a') (Symbol 'b')))
   (a + b)
   ```

2. Definir la función `denot` que recibe una `Regex` α y regresa un `Language` con todas las cadenas del lenguaje denotado por α.

```
denot :: Regex -> Language
```

```
> denot (Star (Add (Symbol 'a') Void))
["","a","aa","aaa","aaaa","aaaaa","aaaaaa","aaaaaaa" ...
> denot (Concat Epsilon (Add (Symbol 'a') (Symbol 'b')))
["a","b"]
```

3. Definir la función `matchD` que recibe una cadena w y una `Regex` α y nos dice si w pertenece al lenguaje denotado por α regresando verdadero si $w \epsilon L[\![\alpha]\!]$ y falso en otro caso, usando la denotación de expresiones regulares.

```
matchD :: String -> Regex -> Bool
```

```
> matchD "aaaaaaaaa" (Star (Add (Symbol 'a') Void))
True
> matchD "aba" (Concat Epsilon (Add (Symbol 'a') (Symbol 'b')))
False
```

3. Derivada

Es fácil observar que la técnica usada en la sección anterior es correcta, pues calculamos todas las cadenas pertenecientes al lenguaje denotado por α y solo vemos si w es una de ellas. El problema con esto surge cuando tratamos con lenguajes infinitos, pues la función `matchD` podría nunca terminar al buscar una cadena en un lenguaje infinito.

Para solucionar este problema, haremos uso del concepto de derivada de una expresión regular usando la proposición $w \, \epsilon \, L[\![\alpha]\!]$ si y solo si $\varepsilon \, \epsilon \, L[\![\partial_w(\alpha)]\!]$ ya demostrada anteriormente.

1. Definir la función `deriv` que recibe una cadena w y una `Regex` α y regresa la derivada de α respecto a w, es decir $\partial_w(\alpha)$. Usando la **Proposición 3** de la nota 2 del curso.

   Hint: Definir las funciones auxiliares que calculan la derivada respecto a un símbolo y la nulidad de una expresión regular.

```
deriv :: String -> Regex -> Regex
```

```
> deriv "aaaaaaaaa" (Star (Add (Symbol 'a') Void))
a*
> deriv "aba" (Concat Epsilon (Add (Symbol 'a') (Symbol 'b')))
∅
```

2. Definir la función `matchV` que recibe una cadena w y una `Regex` α y nos dice si w pertenece al lenguaje denotado por α regresando verdadero si $w \epsilon L[\![\alpha]\!]$ y falso en otro caso, usando la derivada de expresiones regulares.

```
matchV :: String -> Regex -> Bool
```

```
> matchV "aaaaaaaaa" (Star (Add (Symbol 'a') Void))
True
> matchV "aba" (Concat Epsilon (Add (Symbol 'a') (Symbol 'b')))
False
```

Especificación

- Las funciones `matchD`, `matchV` y `deriv` deben regresar expresiones regulares simplificadas.
- En las listas que representan los lenguajes no puede haber cadenas repetidas.
- Todas las funciones deben estar debidamente comentadas con su especificación y descripción.
- En caso de ser necesarias, las funciones auxiliares deben de incluir ademas como comentario el por qué eran necesarias.
- Se debe preservar las firmas de las funciones como aparecen en el documento.
- El ejercicio debe resolverse en un archivo `regex.hs`

Entrega.

- Se entrega antes de las 23:59 horas del día fijado como fecha de entrega.
- El ejercicio debe ser entregado de forma individual.
- Debe incluirse un archivo README.txt con el nombre completo del alumno, así como comentarios, opiniones o ideas sobre el ejercicio.
- Guardar los archivos requeridos en un directorio que tenga como nombre el número de cuenta del alumno. Se entregará éste directorio comprimido.
- Entregarse al correo del ayudante con el asunto [AyLF192-EP01].
- **Cualquier copia o plagio será calificado automáticamente con cero.**