Patrones

Programación funcional para la física computacional

7 de abril de 2023

Identificación de patrones

La identificación de patrones es una de las funciones básicas de un lenguaje de programación. Al estudiar las computadoras como objeto teórico antes de llegar a los modelos computacionales, que ya vimos en una sección anterior, se estudian mecanismos más sencillos, regularmente llamados *sub-Turing*, ya que no llegan a la potencia de de una máquina de Turing o el cálculo λ, pero pueden verse como casos específicos de esos modelos.

Estos niveles de modelos computacionales han sido estratificados por el lingüista Noam Chomsky. La jerarquía de Chomsky pone el siguiente orden:

1. Lenguajes regulares
2. Gramáticas independientes de contexto
3. Gramáticas sin restricciones

El último nivel es el que puede identificarse completo con una máquina de Turing o el cálculo λ, los niveles inferiores pueden perfectamente simularse por una máquina de Turing, pero esos modelos no pueden simular toda máquina de Turing.

El primer nivel son las expresiones regulares, que están ligadas a la identificación de patrones. Las expresiones regulares son aceptadas por autómatas finitos deterministas, la forma más sencilla de mostrar que un lenguaje es regular es construyendo un autómata de este tipo. Un autómata finito determinista se define como:

Definición 1 *Un autómata finito determinista es la quinteta*

$$M = (Q, \Sigma, \delta, s, F)$$

- Q *el conjunto finito de estados;*
- Σ *conjunto finito, alfabeto de entrada;*
- $\delta : Q \times \Sigma \to Q$ *la función de transición;*
- $s \in Q$ *es el estado inicial;*
- $F \subseteq Q$*, los estados de aceptación o estados finales.*

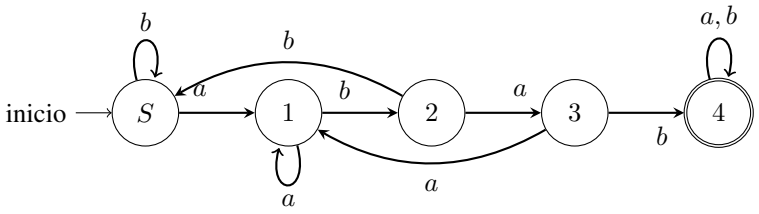

Figura 1: Automata finito determinista que acepta el lenguaje $\{w \in \{a,b\}^* \mid w$ contiene la subcadena $abab\}$.

Ya sea que se defina cada uno de los elementos o que se dibuje un diagrama como se muestra en la figura 1. Lo relevante es que pueden identificarse cadenas infinitas y de longitud infinita con sólo unos pocos estados y transiciones, misma que a su vez es la limitación para poder identificar cadenas en los lenguajes por arriba en la jerarquía de Chomsky. No hay más memoria que el estado en el que se encuentra.

Para identificar patrones en un lenguaje de programación debemos tomar en cuenta:

- Ser cuidadoso con mayúsculas y minúsculas, como carcateres son distintos y la identificación será distinta.
- Usar escapes para caracteres especiales
  - Caracteres de control ($\backslash t$, $\backslash n$, $\backslash r$.)
  - Caracteres numéricos (octal $\backslash nnn$, hexadecimal $\backslash xnn$, *unicode* $\backslash unnnn$)
  - Caracteres especiales de escape ($\backslash ($)
- Caracteres comodín: ., $\backslash w$, $\backslash s$,...
- Anclas : ¸ \$, $*$

Para identificar patrones en *haskell* necesitamos importar una librería, no está definido por *default* en el lenguaje.

```
import Text.Regex.Posix
let vocals = ``[aeiou]''
``Enunciado prueba'' =~ vocales :: Bool
``Enunciado prueba'' =~ vocales :: Int
``Enunciado prueba'' =~ vocales :: String
``Enunciado prueba'' =~ vocales :: (String,String,String)
```

Lo mismo se puede hacer con números, un ejemplo que puede parecer útil es hacer un programa que identifique un número de teléfono con un formato definido.

```
let tel = "\\([0-9]{3}\\)[0-9]{3}\\-[0-9]{4}"
"Mi numero de telefono es: (555)643-1212"=~ tel :: String
```

Nota como se hace la identificación de patrones de acuerdo a lo mencionado líneas arriba.

De manera similar para identificar patrones en *python*

```
import re
vocales = "[aeiou]"
re.search(vocales, "Enunciado de ejemplo")
re.search(vocales,"Enunciado de ejemplo").group()
```

Las funcionalidades necesarias están en la librería *re* (de *regular expressions*), al igual que en *haskell* se definen los patrones *vocales*. Pero la diferencia está en que si simplemente se aplica el método *re.search(vocales, "Enunciado de ejemplo")* lo que regresa es un objeto coincidente, en lugar del objeto que se busca. Para obtenerlo debe además escribirse otra línea idéntica pero finalizando con *.group()*. Pero además de la búsqueda se puede trabajar con cincidencias

```
import re
vocales = "[aeiou]"
re.match(vocales, "Enunciado de prueba")
re.match("E","Enunciado de ejemplo")
re.findall(vocales, "Enunciado de prueba")
```

Aún trabajando más con la identificación de patrones

```
import re
sPrueba = "Esta es\tuna cadena de prueba.\nA fuerzas"
espacio = "[\s]"
re.split(espacio,sPrueba)
```

Lo que hace el programa es dividir la cadena en todas las subcadenas que se encuentran entre espacios, sea un espacio simple, un tabulador o un salto de línea.