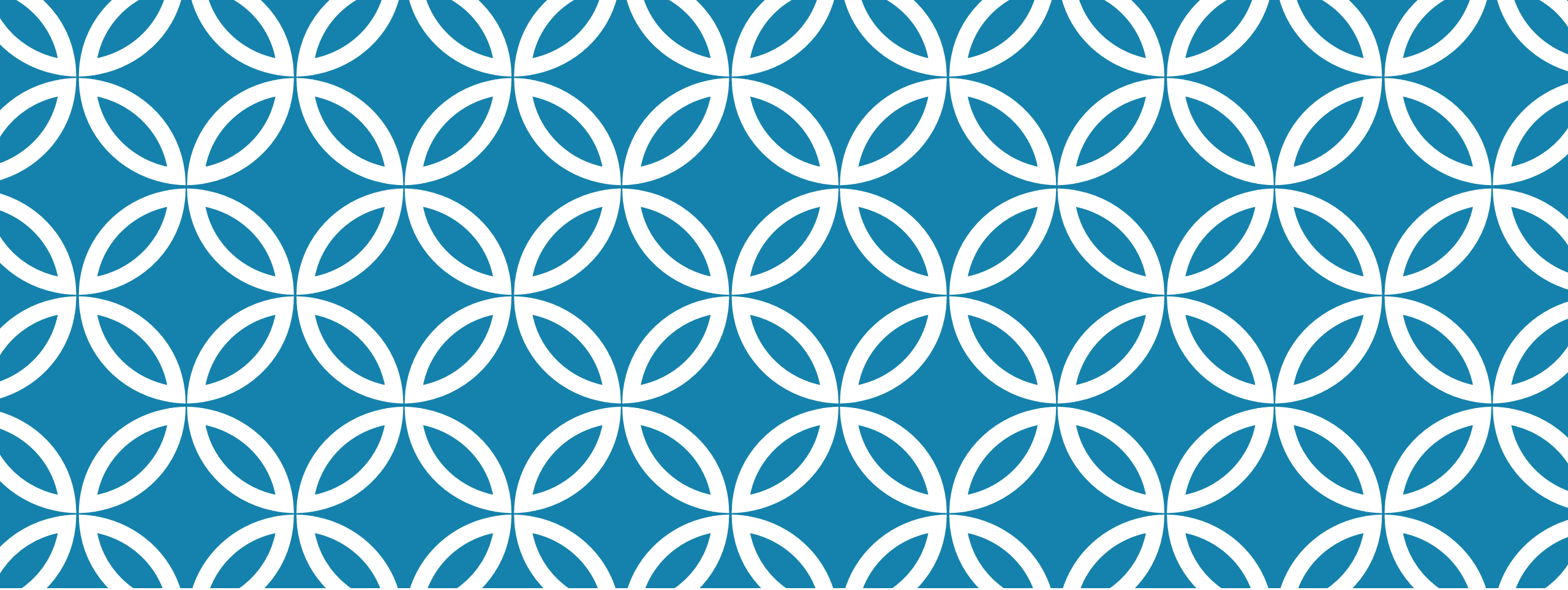

ESTRUCTURAS DISCRETAS

Javier Enríquez Mendoza Mauricio E. Hernández Olvera

Composición y aplicación de funciones, Gramáticas Formales, Lambdas.

COMPOSICIÓN DE FUNCIONES

En matemáticas podemos combinar dos funciones de tal forma que la saluda de una sea la entrada de la otra, esto se conoce como composición de funciones.

$$Sean\ f: X \to Y\ \ g: W \to X\ \ funciones$$

$$Definimos\ (f \circ g)\ como\ f\big(g(w)\big)\ con\ w \in W$$

Es decir, se aplicará g a w y al resultado de esto se le aplicará *f*.

Es importante notar que el contradominio de *g* debe ser igual al dominio de *f*

OPERADOR PUNTO

Como la programación funcional esta basada en el concepto matemático de funciones. Es fácil ver que también se pude hacer composición de funciones.

En Haskell, ya lo hemos hecho de la siguiente manera.

`foo ( bar x)` donde `foo` y `bar` son funciones y `x` es la entrada de `bar`

Pero Haskell nos ofrece una forma mas elegante y limpia de hacerlo.

El operador punto.

`foo . bar x ≡ foo ( bar x)`

OPERADOR $

La precedencia natural de la aplicación de función es la mas alta y es asociativa a la izquierda.

`f a b c = ((f a) b ) c`

Y para alterar la precedencia la única opción que conocemos es usar paréntesis, lo que se vuelve poco legible muy rápido. `f (a (b c))`

Como una alternativa mas elegante, tenemos el operador `$`.

`$` es una aplicación de función, pero mientras que la aplicación convencional tiene la mayor precedencia, `$` tiene el orden de precedencia mas bajo.

`f (a (b c)) ≡ f $ a $ b c`

GRAMÁTICAS FORMALES

Nos sirven para mostrar como se construyen ciertas expresiones.

Es un mecanismo sencillo de especificación de reglas de construcción.

Nos describen la forma o reglas sintácticas que deben tener las expresiones.

Construir una expresión a partir de las reglas sintácticas se le conoce como derivaciones.

Este proceso también puede verse como la construcción de un árbol.

A estos arboles se les conoce como **árbol de derivación.**

GRAMÁTICAS FORMALES EN HASKELL

Para poder implementar en Haskell una gramática formal vamos a utilizar tipos de datos algebraicos.

Escribiremos las reglas sintácticas como constructores del tipo de dato.

Y las hojas del árbol sintáctico serán constructores constantes y los nodos internos constructores que reciben parámetros.

De esta forma nuestro tipo de dato será una implementación del árbol de derivación.

LAMBDAS

En ocasiones vamos a querer usar una función una sola vez y jamás la volveremos a usar.

Entonces es innecesario definir la función y ponerle un nombre.

Para esto usamos funciones anónimas o lambdas.

Las lambdas son simplemente funciones que no están asociadas a ningún identificador.

Normalmente las usamos en funciones de orden superior.

SINTAXIS

```
(\ parametros -> cuerpo )
```

Ejemplos

Identidad:

```
(\x -> x )
```

Sucesor:

```
(\x -> x + 1)
```

Predecesor:

```
(\x -> x - 1)
```

Toda función puede escribirse como una lambda.