Guía de lenguajes 3.1.1

Wollok - Haskell - Prolog

Elementos Comunes

Sintaxis básica

	Wollok	**Haskell**	**Prolog**
Comentario	`// un comentario` `/* un comentario multilínea */`	`-- un comentario` `{- un comentario multilínea -}`	`% un comentario` `/* Un comentario multilínea */`
Strings	`"uNa CadEna"` `'uNa CadEna'`	`"uNa CadEna"`	NA
Caracteres	NA	`'a'`	NA
Símbolos/Átomos	NA	NA	unAtomo
Booleanos	`true false`	`True False`	NA
Set	`#{}` `#{1, 2}`	NA	NA
Lista	`[]` `[1, 1, 2]`	`[]` `[1,1,2]`	`[]` `[1,a,true]`
Patrones de listas	NA	`(cabeza:cola)` `(cabeza:segundo:cola)`	`[Cabeza|Cola]` `[Cabeza,Segundo|Cola]`
Tuplas	NA	`(comp1, comp2)`	`(Comp1, Comp2)`
Data/Functores	NA	`Constructor comp1 comp2`	`functor(Comp1, Comp2)`
Bloques sin parámetros	`{algo}`	NA	NA
Bloques / Exp. lambda (De un parámetro)	`{x => algo con x}`	`(\x -> algo con x)`	NA
Bloques / Exp. lambda (Más de un parámetro)	`{x, y => algo con x e y}`	`(\x y -> algo con x e y)`	NA
Variable anónima	NA	_	_

Operadores lógicos y matemáticos

	Wollok	**Haskell**	**Prolog**
Equivalencia	==	==	NA
Identidad	===	NA	NA
~ Equivalencia	!=	/=	\=
Comparación de orden	> >= < <=	> >= < <=	> >= < =<

Disyunción (O lógico)	`||` `or`	`||`	`NA, usar múltiples cláusulas`
Conjunción (Y lógico)	`&&` `and`	`&&`	`,`

Negación	`! unBool`
	`unBool.negate()`
	`not unBool`
		`not unBool`	`not( Consulta )`

Operadores aritméticos	`+ - * /`	`+ - * /`	`+ - * /`
División entera	`dividendo.div(divisor )`	`div dividendo divisor`	`dividendo // divisor`
Resto	`dividendo % divisor`	`mod dividendo divisor`	`dividendo mod divisor`
Valor absoluto	`unNro.abs()`	`abs unNro`	`abs(Nro)`
Exponenciación	`base ** exponente`	`base ^ exponente`	`base ** exponente`
Raíz cuadrada	`unNro.squareRoot()`	`sqrt unNro`	`sqrt(Nro)`
Máximo entre dos números	`unNro.max(otroNro)`	`max unNro otroNro`	`NA`
Mínimo entre dos números	`unNro.min(otroNro)`	`min unNro otroNro`	`NA`
Par	`unNro.even()`	`even unNro`	`NA`
Impar	`unNro.odd()`	`odd unNro`	`NA`

Operaciones simples sin efecto sobre colecciones / listas

	Wollok	**Haskell**	**Prolog**
Longitud	`coleccion.size()`	`length :: [a] -> Int` ** `genericLength :: Num n => [a] -> n` *	`length/2`
Si está vacía	`coleccion.isEmpty()`	`null :: [a] -> Bool` **	`NA`
Concatenación	`coleccion + otraColeccion`	`(++) :: [a] -> [a] -> [a]`	`append/3`
Unión	`set.union(coleccion)`	`union :: Eq a => [a] -> [a] -> [a]` *	`union/3`
Intersección	`set.intersection( coleccion)`	`intersect ::` `  Eq a => [a] -> [a] -> [a]` *	`intersection/3`
Acceso por índice	`lista.get(indice)` *(base 0)*	`(!!) :: [a] -> Int -> a` *(base 0)*	`nth0/3` `nth1/3`
Pertenencia	`coleccion.contains(elem)`	`elem :: Eq a => a -> [a] -> Bool` **	`member/2`
Máximo	`coleccionOrdenable.max()`	`maximum :: Ord a => [a] -> a` **	`max_member/2`
Minimo	`coleccionOrdenable.min()`	`minimum:: Ord a => [a] -> a` **	`min_member/2`
Sumatoria	`coleccionNumerica.sum()`	`sum :: Num a => [a] -> a` **	`sumlist/2`

Aplanar	`coleccionDeColecciones.flatten()`	`concat :: [[a]] -> [a]` **	`flatten/2`
Primeros n elementos	`lista.take(n)`	`take :: Int -> [a] -> [a]`	NA
Primer elemento	`lista.head()` `lista.first()`	`head :: [a] -> a`	NA
Último elemento	`lista.last()`	`last :: [a] -> a`	NA
Cola	NA	`tail :: [a] -> [a]`	NA
Segmento inicial (sin el último)	NA	`init :: [a] -> [a]`	NA
Apareo de listas	NA	`zip :: [a] -> [b] -> [(a, b)]`	NA
Sin repetidos	`coleccion.asSet()`	NA	NA

Operaciones avanzadas (de orden superior) sin efecto sobre colecciones/listas

	Wollok	**Haskell**
Sumatoria según transformación	`coleccion.sum(bloqueNumericoDe1)`	NA
Filtrar	`coleccion.filter(bloqueBoolDe1)`	`filter :: (a->Bool) -> [a] -> [a]`
Transformar	`coleccion.map(bloqueDe1)`	`map :: (a->b)-> [a] -> [b]`
Todos cumplen (true para lista vacía)	`coleccion.all(bloqueBoolDe1)`	`all :: (a->Bool) -> [a] -> Bool`
Alguno cumple (false para lista vacía)	`coleccion.any(bloqueBoolDe1)`	`any :: (a->Bool) -> [a] -> Bool`
Transformar y aplanar	`coleccion.flatMap(bloqueDe1)`	`concatMap :: (a->[b]) -> [a] -> [b]`
Reducir/plegar a izquierda	`coleccion.fold(valorInicial, bloqueDe2)`	`foldl :: (a->b->a) -> a -> [b] -> a` `foldl1 :: (a->a->a) -> [a] -> a`
Reducir/plegar a derecha	NA	`foldr :: (b->a->a) -> a -> [b] -> a` `foldr1 :: (a->a->a) -> [a] -> a`
Apareo con transformación	NA	`zipWith ::` `(a->b->c) -> [a] -> [b] -> [c]`
Primer elemento que cumple condición	`coleccion.find(bloqueBoolDe1)` `coleccion.findOrElse( bloqueBoolDe1, bloqueSinParametros)`	`find :: (a->Bool) -> [a] -> a` * **
Cantidad de elementos que cumplen condición	`coleccion.count(bloqueBoolDe1)`	NA
Obtener colección ordenada.	`coleccion.sortedBy(bloqueBoolDe2)`	`sort :: Ord a => [a] -> [a]` * **
Máximo según criterio.	`coleccion.max(bloqueOrdenableDe1)`	NA
Mínimo según criterio.	`coleccion.min(bloqueOrdenableDe1)`	NA

Wollok

Mensajes de colecciones con efecto

Agregar un elemento.	`coleccion.add(objeto)`
Agregar todos los elementos de la otra colección	`coleccion.addAll(otraColeccion)`
Evaluar el bloque para cada elemento.	`coleccion.forEach(bloqueConEfectoDe1)`
Eliminar un objeto.	`coleccion.remove(objeto)`
Eliminar todos los elementos.	`coleccion.clear()`
Deja ordenada la lista según un criterio.	`lista.sortBy(bloqueBoolDe2)`

Haskell

Funciones de orden superior sin listas

Aplica una función con un valor (con menor precedencia que la aplicación normal)	`($) :: (a->b) -> a -> b`
Compone dos funciones	`(.) :: (b->c) -> (a->b) -> (a->c)`
Invierte la aplicación de los parámetros de una función	`flip :: (a->b->c) -> b -> a -> c`

Funciones de generación de listas

Genera una lista que repite infinitamente al elemento dado	`repeat  :: a -> [a]`
Para iterate f x, genera la lista infinita [x, f x, f (f x), ...]	`iterate :: (a->a) -> a -> [a]`
Genera una lista que repite una cierta cantidad de veces al elemento dado	`replicate :: Int -> a -> [a]`
Para cycle xs, genera la lista infinita xs ++ xs ++ xs ++ ...	`cycle  :: [a] -> [a]`

Prolog

Predicados de orden superior

Para todo	`forall(Antecedente, Consecuente)`
Define una lista a partir de una consulta	`findall(Formato, Consulta, Lista)`

Notas

NA: “No Aplica”. No existe o no se recomienda su uso.

* Declarada en Data.List

** El tipo presentado es una versión simplificada del tipo real