Guía de lenguajes

Aclaraciones

- (*) Significa que sólo funciona para colecciones con índice (Smalltalk)
- (**) Significa que está declarada en Data.List (Haskell)
- (***) Significa que el tipo presentado acá es una versión simplificada del tipo real (Haskell)
- NA significa “No Aplica”. En otras palabras, ó no existe ó no se recomienda su uso.

Elementos Comunes

Comentarios

Smalltalk	**Haskell**	**Prolog**
“un comentario”	-- un comentario {- un comentario multilínea -}	% un comentario /* Un comentario multilínea */

Valores Literales

	Smalltalk	**Haskell**	**Prolog**
Strings	‘uNa CadEna’	“uNa CadEna”	NA
Caracteres	$a	‘a’	NA
Símbolos/Átomos	#unSimbolo	NA	unSimbolo
Booleanos	true false	True False	NA
Array/Lista	{1. 2. 3} {1. $a. ‘hola’}	[1, 2, 3]	[1, 2, 3] [1, unSimbolo, [ ]]
Tuplas	NA	(1, True, [1, 2])	NA
Data/Functores	NA	Constructor 1 True	nombreFunctor(1, unSimbolo)
Bloques/Funciones Anónimas	[:param1 :param2 | cuerpo]	\param1 param2 -> cuerpo	NA

Operadores lógicos y matemáticos

	Smalltalk	**Haskell**	**Prolog**
Equivalencia	=	==	NA
Identidad	==	NA	NA
~ Equivalencia	~=	/=	\=
~ Identidad	~~	NA	NA
Comparación de orden	> >= < <=	> >= < <=	> >= < =<
Disyunción (O lógico)	| (ansiosa) or: (perezosa)	||	NA
Conjunción (Y lógico)	& (ansiosa) and: (perezosa)	&&	,
Negación	unBool not	not unBool	not(Consulta)

Operadores matemáticos

	Smalltalk	**Haskell**	**Prolog (sólo como parte de un is)**
Operadores aritméticos comunes	`+ - * /`	`+ - * /`	`+ - * /`
División entera	`dividendo // 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 raisedTo: exponente`	`base ^ exponente`	`base ** exponente`
Raíz cuadrada	`unNro sqrt`	`sqrt unNro`	`sqrt(Nro)`
Máximo ó mínimo entre dos números	`unNro max: otroNro` `unNro min: otroNro`	`max unNro otroNro` `min unNro otroNro`	NA

Operaciones “simples” sin efecto (efecto colateral) sobre colecciones / listas

	Smalltalk (mensajes)	**Haskell (funciones)**	**Prolog (predicados)**
Longitud	`size`	`length :: [a] -> Int` `genericLength :: Num n => [a] -> n`	`length/2`
Concatenación	`, (*)`	`++`	`append/3`
Unión e intersección	`union:`	`union (**)`	`union/3`
Intersección	`intersection:`	`intersect (**)`	`intersection/3`
Acceso por índice base 0	NA	`unaLista !! unNro`	`nth0/3`
Acceso por índice base 1	`at: (*)`	NA	`nth1/3`
Pertenencia	`includes:`	`elem`	`member/2`
Máximo ó mínimo de un conjunto de números	`max` `min`	`maximum` `minimum`	`max/3` `min/3`
Buscar el máximo o mínimo según condición booleana	`detectMax:` `detectMin:`	`maximumBy :: (a -> a -> Bool) -> [a] -> a` `minimumBy (**) (***)`	NA
Sumatoria de un conjunto de números	`sum`	`sum`	`sumlist/2`

Operaciones “avanzadas” sin efecto (efecto colateral) sobre colecciones/listas

	Smalltalk	Haskell
Sumatoria de un conjunto de elementos según una transformación	`sum:`	NA
Primeros n elementos de un conjunto	`first:`	`take`
Cantidad de ocurrencias	`ocurrencesOf:`	NA
Filtrar	`select:`	`filter`
Rechazar (filtrar los que no cumplen la condicion)	`reject:`	NA
Mapear	`collect:`	`map`
Aplanar colección de colecciones / lista de listas	`flatten`	`concat`
Aplanar y mapear	`gather:`	`concatMap`
Reducir/plegar a izquierda	`inject: into:`	`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`
Todos cumplen (verdadero para lista vacía)	`allSatisfy:`	`all`
Alguno cumple (falso para lista vacía)	`anySatisfy`	`any`
Primer elemento	`first`	`head`
Último elemento	`last`	`last`
Cola	`allButFirst`	`tail`
Segmento inicial (todos menos el último)	`allButLast`	`init`

Apareo de listas	NA	`zip :: [a] -> [b] -> [(a, b)]`
		`zipWith ::`
		`(a -> b -> c) -> [a] -> [b] -> [c]`

Buscar el primer elemento que cumpla una condición	`detect:` `detect:  ifNone:`	`find  :: (a -> Bool) -> [a] -> a` `(**) (***)`
Posición en la que se encuentra la primera ocurrencia de un elemento	`indexOf: (*)`	NA
Si comienza/termina con	`beginsWith: (*)` `endsWith: (*)`	`isPreffixOf (**)` `isSuffixOf (**)`
Sublista entre dos posiciones	`copyFrom: to: (*)`	NA
Cantidad de elementos que cumplen una condición	`count:`	NA

Patrones

	Haskell	**Prolog**

Listas	`[]`
	`(cabeza:cola)`
	`(cabeza:segundo:cola)`
		`[]`
		`[Cabeza|Cola]`
		`[Cabeza,Segundo|Cola]`

Tuplas	`(componente1, componente2)`	NA
Data/Functores	`Constructor componente1 componente2`	`nombreFunctor(componente1,componente2)`
Variable anónima	_	_

Funciones de Haskell

Operaciones sobre funciones

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

Unfolds (generación de listas a partir de una semilla)

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

Predicados de Prolog

Predicados fundamentales de orden superior

- not(Consulta)
- findall(Formato, Consulta, Lista)
- forall(Antecedente, Consecuente)

Mensajes de Smalltalk

Control de flujo imperativo

`unBool ifTrue: [unasSentencias ]`	Ejecuta *unasSentencias* dependiendo del valor de verdad de *unBool.*
`unBool ifTrue: [unasSentencias ]` `ifFalse: [otrasSentencias ]`	Ejecuta *unasSentencias* u *otrasSentencias* dependiendo del valor de verdad de *unBool.*
`[unBool ] whileTrue: [sentencias]`	Ejecuta iterativamente las *sentencias* mientras *unBool* sea verdadero.
`unNro timesRepeat: [sentencias]`	Ejecuta iterativamente las *sentencias* exactamente *unNro* de veces.
`unNro to: otroNro` `do: [:indice | sentencias ]`	Ejecuta iterativamente las *sentencias* la cantidad de veces comprendida entre *unNro* y *otroNro*. *Indice* varía en cada iteración, desde *unNro* hasta *otroNro*.

Tipos de Colecciones

- Sin orden:
  - Bag: Tamaño variable, sin subíndice.
  - Set: Tamaño variable, sin subíndice, no permite repetidos.
- Con orden:
  - Array: Tamaño fijo, con subíndice, orden de acuerdo al subíndice. En este sentido un String se comporta como un Array
  - OrderedCollection: Tamaño variable, con subíndice, orden de acuerdo al subíndice.
  - SortedCollection: Tamaño variable, con subíndice, orden de acuerdo a criterio que se especifica.

  Las colecciones con orden respetan el orden de los elementos en do: / select: / collect: / etc..
- Dictionary: Tamaño variable, acceso por clave, no permite claves repetidas
  do: / select: / collect: / etc. funcionan sobre los valores incluidos, no se tienen en cuenta las claves.

Mensajes de diccionarios

unaCol **at:** *unaClave*	Devuelve el valor asociado a *unaClave*, *nil* si unaClave no tiene asociado ningún valor.
unaCol **at:** *unaClave* **put:** *unObjeto*	Coloca *unObjeto* como valor asociado a *unaClave*.

Mensajes de conversión de colecciones (devuelven nuevas colecciones)

unaCol **asBag** *unaCol* **asSet** *unaCol* **asOrderedCollection** *unaCol* **asArray**	Devuelve una **nueva** colección de la clase indicada con todos los elementos de *unaCol*.
unaCol **asSortedCollection:** [:*anterior* :*siguiente* | *unaCondicion*]	Devuelve una **nueva** colección con todos los elementos de *unaCol* ordenados según *unaCondicion*. *unaCondicion* es una expresión de valor booleano en la que intervienen *anterior* y *siguiente*. *anterior* quedará delante de *siguiente* cuando *unaCondicion* sea verdadera.

Mensajes de colecciones con efecto (efecto colateral)

unaCol **add:** *unObjeto*	Agrega *unObjeto* a *unaCol.* Para las colecciones con subíndice se agregan al final. **Devuelve *unObjeto***
unaCol **addAll:** *otraCol*	Agrega todos los elementos de *otraCol* a *unaCol.* Para las colecciones con subíndice se agregan al final. **Devuelve *otraCol***
unaCol **do:** *unBloque*	Ejecuta *unBloque* con efecto colateral para cada elemento de *unaCol.*
unaCol **removeAllSuchThat:** [:*unElem* | *unaExpr*]	Remueve de *unaCol* los *unElem* que hagan verdadera a *unaExpr*. Devuelve **esa misma *unaCol* modificada**.
unaCol **remove:** *unObjeto*	Elimina *unObjeto* de *unaCol.* **Devuelve *unObjeto.***
unaCol **removeAll**	Elimina todos los elementos de *unaCol*. **Devuelve *unaCol.***
unaCol **at:** *unNro* **put:** *unObjeto* (*)	Coloca *unObjeto* en la posicion *unNro* de *unaCol*. Inválido para SortedCollection.