Programación Funcional Avanzada

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

El Ambiente Haskell

- Módulos
- Paquetes
- Cabal
- Cabal-install - Hackage
- Stack - Stackage
- Haddock
- HLint

Módulos en Haskell

Un programa Haskell consiste en una colección de módulos.

Un módulo define una colección de funciones, tipos de datos, clases, etc.

Los módulos permiten:

- controlar el espacio de nombres
- implementar tipos abstractos de datos
- definir unidades separadas de compilación (en GHC)

Sintaxis

module *nombre* [*exporta*] where *cuerpo*

- *nombre* es el nombre del módulo
- [*exporta*] es una lista opcional de entidades a exportar
- *cuerpo* es el contenido del módulo

Módulo *Main*

Por convención un programa Haskell debe tener un módulo que se llame *Main*

module *Main* where ...

el cual debe exportar una función *main*

main :: *IO a*

Si se omite la declaración de un módulo a este se le asigna por defecto el nombre *Main*.

Módulos jerárquicos

El nombre de un módulo consiste en uno o más identificadores separados por puntos, formando una jerarquía de nombres. Cada identificador comienza con mayúsculas.

Ejemplos: *Main*, *Data.List*, *A.B.C.D*

Para la mayoría de los compiladores se espera que el módulo *A.B.C.D* se encuentre en el archivo `A/B/C/D.hs`

`http://www.haskell.org/haskellwiki/Hierarchical_module_names`

Lista de exportaciones

Opcionalmente se puede especificar qué se desea exportar de un módulo

- funciones
- sinónimos de tipos
- tipos de datos
- nombres de campos de registros
- constructores
- type classes
- métodos
- módulos enteros

Las instancias de clases se exportan siempre
Lo que no se incluye en la lista no se exporta
Si no se provee de una lista de exportaciones, se exportan TODAS las declaraciones (top-level) del módulo

Sintaxis

La lista de exportaciones se encierra entre paréntesis curvos, cada elemento puede ser:

- nombre (cualificado) de función
- sinónimo de tipo
- constructor
- tipos de datos
  - listar constructores y campos entre paréntesis
  - (..) para exportar todo
  - () para hacer el tipo abstracto
- nombre de una type class
  - listar los métodos entre paréntesis
  - (..) para exportar todo
- el nombre de un módulo, precedido de la palabra clave module

Importación de módulos

import [*qualified*] *nombre* [*as nombre*] [*importa*]

Las declaraciones import sólo pueden aparecer en el cabezal del módulo.

Un módulo puede ser importando más de una vez, de formas diferentes.

Si un módulo se importa *qualified* se traen al scope sólo los nombres cualificados. En otro caso se traen también los no cualificados.

Un módulo puede ser renombrado usando *as nombre*. De esta forma los nombres cualificados se importan usando ese nuevo nombre.

Los conflictos de nombres se reportan de forma "perezosa".

Listas de importaciones

Se puede indicar qué se quiere importar de un módulo, de forma similar a las listas de exportaciones.

- Sólo lo que se exporta se puede importar
- La lista de importaciones restringe más la importación
- Si se agrega la palabra reservada *hiding* al principio, la lista indica qué NO se quiere importar

El módulo *Prelude*

El módulo *Prelude* se importa implícitamente como:

import *Prelude*

Si se agrega una importación explícita de *Prelude*, la implícita no se incluye

import *qualified Prelude*

hace que sólo se importen los nombres de *Prelude* en su forma cualificada

Ejemplos

module A (a, b) where

$a = 1$
$b = 2$

module B where

import A

$c = 3$

Todo lo que exporta *Prelude* (ej. *zip*, *Prelude.zip*, etc), a, $A.a$, b, $A.b$, c y $B.c$.

module C where
import *qualified Prelude*
import B *as* M

$d = 4$

Nombres cualificados de lo que exporta *Prelude*, c, $M.c$, d y $C.d$.

Paquetes Haskell

Los paquetes son colecciones de módulos que se distribuyen juntos

No son parte del estándar de Haskell, pero son soportados por GHC

Son versionados y pueden depender de otros paquetes

Sus módulos pueden ser expuestos u ocultos.

Teniendo paquetes, una entidad se distingue por su nombre, módulo, el nombre de su paquete y su número de versión.

Administrador de paquetes GHC

El administrador de paquetes se llama `ghc-pkg`

Los paquetes que conoce GHC se almacenan en una base de datos de configuración de paquetes, usualmente llamada `package.conf`

Pueden haber varias bases de datos de configuración de paquetes:

- una global por instalación de GHC
- una local por cada usuario
- otras locales para propósitos especiales

Administrador de paquetes GHC (2)

Los paquetes conocidos se listan con: `ghc-pkg list`

Se puede acceder a la descripción de un paquete con:
`ghc-pkg describe` *nombre*

El administrador de paquetes también se puede utilizar para registrar, "desregistrar" y actualizar paquetes; pero esto se hace usualmente usando Cabal.

Cabal

Cabal es el sistema de paquetes estandar para software Haskell.

Cabal es un paquete creado usando Cabal.

Está integrado dentro de un conjunto de paquetes que se incluyen con las distribuciones de GHC.

`http://www.haskell.org/cabal/`

`https://cabal.readthedocs.io/en/stable/`

Descripción de paquetes

Cada paquete incluye un archivo (con extensión `.cabal`) con la descripción del paquete.

El archivo debería contener alguna propiedades globales y ciertas secciones

- Las propiedades globales describen nombre, licencia, autor, etc.
- Se pueden declarar flags de configuración, que permiten habilitar y deshabilitar algunas características del paquete
- Sección (opcional) de biblioteca, propiedades e información de construcción de la biblioteca
- Secciones de ejecutable, describiendo programas ejecutables

Archivo de *Setup*

Además de la descripción del paquete se debe incluir un archivo *Setup.hs*, que en la mayoría de los casos es sólo:

```
import Distribution.Simple
main = defaultMain
```

Hay variantes para casos especiales, como instalar archivos que no tienen nada que ver con Haskell.

Usando Cabal

`runghc Setup configure`
Resuelve dependencias

`runghc Setup build`
Construye el paquete

`runghc Setup install`
Instala el paquete y lo registra como paquete GHC

Cabal-Install

Es un paquete que provee la herramienta de línea de comandos `cabal`, que automatiza la búsqueda, configuración, compilación e instalación de paquetes.

`cabal install` *nombre*

Hackage

Base de datos online de paquetes Cabal

Todo el mundo puede subir sus paquetes basados en Cabal

Construcción automática de paquetes

Acceso online a la documentación Haddock del paquete

`http://hackage.haskell.org/`

Stack

Stack es un sistema de compilación para Haskell cuyo objetivo es brindar compilaciones reproducibles

Gestiona todas sus dependencias y la versión de GHC que se utiliza para su proyecto, que quedan *congeladas*

Stackage es un repositorio que almacena conjuntos de paquetes compatibles entre sí (conteniendo todas sus dependencias), llamados *snapshots*. También incluyen la versión de GHC usada.

`https://docs.haskellstack.org/`

Haddock

Haddock es un generador de documentación para Haskell (como JavaDoc, Doxygen, etc.)

Parsea archivos Haskell con anotaciones

Soporta extensiones

Documentación de API y Programa

Backends: HTML y DocBook

Ver: `http://www.haskell.org/haddock/`

HLint

HLint lee programas Haskell y realiza sugerencias para hacerlos más sencillos de leer.

Permite deshabilitar sugerencias no deseadas

Permite agregar nuevas sugerencias

Ver: `https://github.com/ndmitchell/hlint#readme`

HLint (Ejemplo)

Si tenemos (en `ejemplo.hs`):

$$g\ x\ y = \text{if } x \text{ then } \mathit{True} \text{ else } y$$

Ejecutar
`hlint ejemplo.hs`
retorna:

```
ejemplo.hs:1:9: Error: Redundant if
Found:
  if x then True else y
Why not:
  x || y
```

Próxima Clase

- Pruebas usando Razonamiento Ecuacional
- Testing con QuickCheck
- Leer: Koen Claessen, John Hughes. **QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs.**