Trabajo Práctico 1
Programación Funcional

Paradigmas de Lenguajes de Programación
1er cuatrimestre, 2015

Fecha de entrega: 23 de abril

1. Introducción

El objetivo de este trabajo práctico es realizar un intérprete de la lógica modal básica.'

La signatura del lenguaje modal básico consta de un conjunto infinito de variables proposicionales, al que llamaremos $PROP$. El conjunto de fórmulas está definido como:

$$FORM := p \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Diamond\phi \mid \Box\phi$$

en donde $p \in PROP$ y $\phi, \psi \in FORM$.

Una fórmula de esta lógica se evalúa en una estructura conocida como modelo de Kripke. Un modelo de Kripke es, básicamente, un grafo en donde en cada uno de sus nodos los símbolos proposicionales pueden ser verdaderos o falsos.

En el contexto de este trabajo, los modelos de Kripke que nos interesan están formados por:

- W un conjunto de nodos o mundos.
- R una relación binaria en W. Notar que $\langle W, R\rangle$ conforma un grafo dirigido con nodos W.
- $V : PROP \to \mathcal{P}(W)$ es una función de valuación que indica, para un símbolo proposicional, en qué mundos es verdadero.

La semántica de la lógica modal básica está definida por las siguientes reglas:

$$\begin{array}{lll}
\mathcal{M}, w \models p & sii & w \in V(p) \text{ para p } \in PROP \\
\mathcal{M}, w \models \neg\phi & sii & \mathcal{M}, w \not\models \phi \\
\mathcal{M}, w \models \phi \wedge \psi & sii & \mathcal{M}, w \models \phi \text{ y } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \phi \vee \psi & sii & \mathcal{M}, w \models \phi \text{ o } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \Diamond\phi & sii & \exists w' \in W \cdot R(w, w') \text{ y } \mathcal{M}, w' \models \phi \\
\mathcal{M}, w \models \Box\phi & sii & \forall w' \in W \cdot R(w, w') \text{ y } \mathcal{M}, w' \models \phi
\end{array}$$

siendo $\mathcal{M}$ un modelo.

Ejemplo: En el siguiente modelo de Kripke se cumple que $\mathcal{M}, w \models p \wedge \Box q$, $\mathcal{M}, w \models p \wedge \Diamond r$ y $\mathcal{M}, w \not\models \Box r$.

Notar que para el modelo de Kripke $\mathcal{M}$ de este ejemplo $V(p) = \{w\}$.

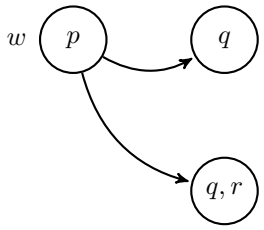


Figura 1: Modelo de Kripke de ejemplo

2. Diseño

La implementación está dividida en los siguientes módulos:

Grafo para representar grafos de cualquier tipo usando Grafo a.

Tipos para definición de los tipos usados dentro de los cuales se define Exp para representar las fórmulas.

Parser para facilitar la creación de fórmulas por medio de un parser. Cortesía de la cátedra y Happy[1].

Lomoba donde se definen las operaciones de **ló**gica **mo**dal **bá**sica usando el módulo Grafo y Tipos.

3. Grafos

Un grafo se representa mediante la siguiente declaración:

data Grafo a = G [a] (a → [a])

Un grafo G ns r está formado por los nodos de la lista ns donde, para un nodo n ∈ ns, sus vecinos son (r n). Notar que la lista no debe tener duplicados y la función debe ser total, aunque no se espera un comportamiento particular para elementos fuera del grafo.

Pueden agregar restricciones al tipo paramétrico en las funciones de ser estrictamente necesario. Por ejemplo, para contar con la igualdad de a agregar (Eq a) ⇒.

Ejercicio 1

vacio :: Grafo a debe dar un grafo vacío.

Ejercicio 2

nodos :: Grafo a → [a] debe devolver los nodos del grafo.

Ejercicio 3

vecinos :: Grafo a → a → [a] debe devolver los vecinos del nodo dado en el grafo. Asumir que el nodo está en el grafo.

Ejercicio 4

agNodo :: a → Grafo a → Grafo a agrega un nodo sin vecinos al grafo.

[1] http://www.haskell.org/happy/

Ejercicio 5

sacarNodo :: a → Grafo a → Grafo a saca un nodo del grafo (y sus ejes asociados).

Ejercicio 6

agEje :: (a,a) → Grafo a → Grafo a agrega un eje en el grafo, desde el nodo de la primera componente de la tupla hacia el segundo.

Ejercicio 7

lineal :: [a] → Grafo a dada una lista sin repetidos, arma un grafo lineal con dichos elementos. Si la lista son los nodos $w_0 \dots w_{n-1}$, cada nodo w_i, está relacionado solamente con w_{i+1}. El grafo correspondiente a (lineal [1..4]) es:

Ejercicio 8

union :: Grafo a → Grafo a → Grafo a devuelve un grafo con la unión de los nodos de los otros dos. Los nodos pueden estar en ambos grafos, con lo cual hay que unir los vecinos de forma adecuada.

Ejercicio 9

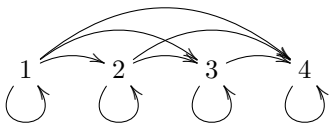
clausura :: Grafo a → Grafo a devuelve un grafo que resulta de clausurar el otro grafo. Dado el grafo $\langle W, R\rangle$, su clausura es $\langle W, R^*\rangle$ donde $R^* = \cup_{i=0}^{\infty} R^i$, o sea, R^* es la clausura reflexo transitiva de R. Resolver usando una función puntofijo :: (Eq a) ⇒ (a → a) → (a → a) que genere la función que devuelve el punto fijo para cualquier valor de entrada.

Ejemplos:

Sea f una función tal que f 0 ⇝ 5, f 5 ⇝ 2, f 2 ⇝ 2, entonces (puntofijo f) 0 ⇝ 2.

Sea un grafo g:

El resultado de clausura g es:



4. Tipos

Se definen los siguientes tipos para el resto del trabajo:

```
type Mundo = Integer
type Prop = String
data Modelo = K (Grafo Mundo) (Prop → [Mundo])
data Exp = Var Prop | Not Exp | Or Exp Exp | And Exp Exp | D Exp | B Exp
```

Un modelo de Kripke se representa con Modelo, donde se incluye el grafo subyacente y la función de valuación. Dicha función es total.

Las fórmulas se representan con Exp según la sintaxis antes vista. En la siguiente sección se muestran ejemplos de fórmulas.

5. Parser

Este módulo exporta la función parse :: String → Exp que permite traducir de un String a una Exp usando la "sintaxis" natural. Ejemplos:

- p (parse "p") ⇝ (Var "p")
- $p \wedge q$ (parse "p && q") ⇝ (And (Var "p") (Var "q"))
- $p \vee q$ (parse "p || q") ⇝ (Or (Var "p") (Var "q"))
- $\neg(p \vee q)$ (parse "!p || q") ⇝ (Or (Not (Var "p")) (Var "q"))
- $\Diamond p \wedge q$ (parse "<>p && q") ⇝ (And (D (Var "p")) (Var "q"))
- $\Box p \wedge q$ (parse "[]p && q") ⇝ (And (B (Var "p")) (Var "q"))
- $\Diamond(p \wedge q)$ (parse "<>(p && q)") ⇝ (D (And (Var "p") (Var "q")))
- $\Box(p \wedge q)$ (parse "[](p && q)") ⇝ (B (And (Var "p") (Var "q")))

6. Lomoba

Ejercicio 10

Definir fold para el tipo Exp. Se permite usar recursión explícita.

Ejercicio 11

visibilidad :: Exp → Integer calcula la visibilidad de la fórmula. La visibilidad indica, en cierto modo, cuánto del grafo se utiliza efectivamente para evaluar la fórmula. En caso de que no aparezcan $\Diamond$ o $\Box$ la visibilidad es 0. Cada vez que aparece uno de estos, como la semántica del lenguaje indica que se debe dar un paso en el grafo, la visibilidad se incrementa en 1.
Ejemplos:

$$\begin{aligned}
\text{visibilidad}(p) &= 0 \\
\text{visibilidad}(\Diamond p) &= 1 \\
\text{visibilidad}(\Diamond\neg\Diamond p) &= 2 \\
\text{visibilidad}(\Diamond\Diamond p \vee \Diamond\Diamond q) &= 2 \\
\text{visibilidad}(\Diamond(\Diamond p \vee \Diamond\Diamond q)) &= 3 \\
\text{visibilidad}(\Box(\Diamond p \wedge \Diamond\Box q)) &= 3
\end{aligned}$$

Ejercicio 12

extraer :: Exp → [Prop] lista las variables proposicionales que aparecen en la fórmula, sin repetir.

Ejercicio 13

eval :: Modelo → Mundo → Exp → Bool dados el modelo $\mathcal{M}$, el mundo w y la fórmula ψ, indica si $\mathcal{M}, w \models \psi$. Sugerencia: definir eval' :: Modelo → Exp → Mundo → Bool.

Ejercicio 14

valeEn :: Exp → Modelo → [Mundo] dados la fórmula ψ y el modelo $\mathcal{M}$, genera todos los mundos w tales que $\mathcal{M}, w \models \psi$.

Ejercicio 15

quitar :: Exp → Modelo → Modelo dados la fórmula ψ y el modelo $\mathcal{M}$, genera un modelo donde se eliminaron todos los mundos w tales que $\mathcal{M}, w \not\models \psi$ (es decir $\mathcal{M}, w \models \neg\, \psi$).

Ejercicio 16

cierto :: Modelo → Exp → Bool dados el modelo $\mathcal{M}$ y la fórmula ψ, indica si para todos los mundos w es cierto que $\mathcal{M}, w \models \psi$.

Pautas de Entrega

Se debe entregar el código impreso con la implementación de las funciones pedidas. Cada función debe contar con un comentario donde se explique su funcionamiento. Cada función asociada a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Haskell a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PF]` seguido inmediatamente del nombre del grupo.
- El código Haskell debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto (puede adjuntarse un `.zip` o `.tar.gz`).
- El código entregado **debe** incluir tests que permitan probar las funciones definidas.

El código debe poder ser ejecutado en Haskell2010. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté **adecuadamente** comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código). Los objetivos a evaluar son:

- Corrección.
- Declaratividad.
- Prolijidad: evitar repetir código innecesariamente y usar adecuadamente las funciones previamente definidas (tener en cuenta tanto las funciones definidas en el enunciado como las definidas por ustedes mismos).
- Uso adecuado de funciones de alto orden, currificación y esquemas de recursión.

Salvo donde se indique lo contrario, **no se permite utilizar recursión explícita**, dado que la idea del TP es aprender a aprovechar las características enumeradas en el ítem anterior. Se permite utilizar listas por comprensión y esquemas de recursión definidos en el preludio de Haskell y los módulos `Prelude`, `List`, `Maybe`, `Data.Char`, `Data.Function`, `Data.List`, `Data.Maybe`, `Data.Ord` y `Data.Tuple`. Las sugerencias de los ejercicios pueden ayudar, pero no es obligatorio seguirlas. Pueden escribirse todas las funciones auxiliares que se requieran, pero estas no pueden usar recursión explícita (ni mutua, ni simulada con fix).

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Tests: se recomienda la codificación de tests. Tanto HUnit https://hackage.haskell.org/package/HUnit como HSpec https://hackage.haskell.org/package/hspec permiten hacerlo con facilidad.

Para instalar HUnit usar: `> cabal install hunit`

Para instalar cabal ver: https://wiki.haskell.org/Cabal-Install

Referencias del lenguaje Haskell

Como principales referencias del lenguaje de programación Haskell, mencionaremos:

- **The Haskell 2010 Language Report:** el reporte oficial de la última versión del lenguaje Haskell a la fecha, disponible online en http://www.haskell.org/onlinereport/haskell2010.
- **Learn You a Haskell for Great Good!**: libro accesible, para todas las edades, cubriendo todos los aspectos del lenguaje, notoriamente ilustrado, disponible online en http://learnyouahaskell.com/chapters.
- **Real World Haskell**: libro apuntado a zanjar la brecha de aplicación de Haskell, enfocándose principalmente en la utilización de estructuras de datos funcionales en la "vida real", disponible online en http://book.realworldhaskell.org/read.
- **Hoogle**: buscador que acepta tanto nombres de funciones y módulos, como signaturas y tipos *parciales*, online en http://www.haskell.org/hoogle.
- **Hayoo!**: buscador de módulos no estándar (i.e. aquéllos no necesariamente incluídos con la plataforma Haskell, sino a través de **Hackage**), online en http://holumbus.fh-wedel.de/hayoo/hayoo.html.