Sintaxis de LP en `Haskell`
Lógica Computacional 2017-2

Lourdes del Carmen González Huesca
Roberto Monroy Argumedo
Fernando A. Galicia Mendoza

Facultad de ciencias,UNAM

Miércoles, 8 de febrero del 2016

Clases

Recordemos que una un tipo es una colección de elementos, definimos una **Clase** como una colección de tipos que soportan ciertas operaciones sobrecargadas llamados **métodos**.
`Haskell` provee de ciertas clases básicas que están construidas dentro del mismo lenguaje.

Eq

Esta clase contiene los tipos cuyos valores pueden ser comparados por la relación de igualdad.
Métodos:

- `(==) :: a -> a -> Bool`: Igualdad de dos elementos de tipo `a`.
- `(/=) :: a -> a -> Bool`: Desigualdad de dos elementos de tipo `a`.

Ord

Esta clase contiene los tipos que son instancias de la clase `Eq`, pero agregando las relaciones de orden.
Métodos:

- `(<) :: a -> a -> Bool`: Relación menor que de dos elementos de tipo `a`.
- `(>) :: a -> a -> Bool`: Relación mayor que de dos elementos de tipo `a`.
- `(<=) :: a -> a -> Bool`: Relación menor o igual que de dos elementos de tipo `a`.
- `(>=) :: a -> a -> Bool`: Relación mayor o igual que de dos elementos de tipo `a`.
- `min :: a -> a -> Bool`: El mínimo entre dos elementos de tipo `a`.
- `max :: a -> a -> Bool`: El máximo entre dos elementos de tipo `a`.

Show

Esta clase contiene los tipos cuyos valores pueden ser convertidos a cadenas de caracteres.
Método:

```
show :: a -> String
```

Read

Contraria a la clase `Show`, esta clase contiene los tipos cuyos valores pueden ser convertidos de cadenas de caracteres. Método:

```
read :: a -> String
```

Num

Esta clase contiene instancias de las clases `Show` y `Eq` agregando las operaciones aritméticas.
Métodos:

- `(+) :: a -> a -> a`
- `(-) :: a -> a -> a`
- `(*) :: a -> a -> a`
- `negate :: a -> a`
- `abs :: a -> a`
- `signum :: a -> a`

Integral

Esta clase contiene instancias de la clase `Num` con la suposición de que sus valores son enteros.
Métodos:

- `div :: a -> a -> a`
- `mod :: a -> a -> a`

Fractional

Esta clase contiene instancias de la clase `Num` con la suposición de que sus valores son reales.
Método:

- `(/) :: a -> a -> a`

Derivando o instanciando

Para los tipos que definamos podemos definir los métodos de cada clase, esto se le dice **derivar** o **instanciar** una clase.
Derivar lo que va hacer es darle una forma predefinida por el lenguaje y para poder hacer esto hay que agregar la sentencia:

```
derving(Clase1,...,Clasen)
```

Instanciar es un proceso por el cual nosotros le indicamos a `Haskell` cual va a ser el resultado del método, para poder hacer esto debemos hacer lo siguiente:

```
instance Clase TipoUtilizado => Clase Tipo where
                                f x = y
```

La parte `Clase TipoUtilizado => Clase Tipo` indica que el tipo `TipoUtilizado` ya tiene definidos los métodos de `Clase`.

Complejidad descriptiva

Complejidad descriptiva

¿P = NP?

Complejidad descriptiva

$$¿\mathrm{P} = \mathrm{NP}?$$

Problemas como SAT, la 3-coloración, agente viajero se sabe que con **NP**-completos, un área que se dedica al estudio de las complejidades es la *complejidad descriptiva*.

Teorema

Todo vértice tiene un color y no tiene dos vértices adyacentes del mismo color.

$$\exists R(\exists Y(\exists B(\forall x((R(x) \vee Y(x) \vee B(x)) \wedge (\forall y(E(x,y) \rightarrow \neg(R(x) \wedge B(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y)))))))$$

Complejidad descriptiva

$$\text{¿P} = \text{NP?}$$

Problemas como SAT, la 3-coloración, agente viajero se sabe que con **NP**-completos, un área que se dedica al estudio de las complejidades es la *complejidad descriptiva*.

Teorema

Todo vértice tiene un color y no tiene dos vértices adyacentes del mismo color.

$$\exists R(\exists Y(\exists B(\forall x((R(x) \vee Y(x) \vee B(x)) \wedge (\forall y(E(x,y) \rightarrow \\ \neg(R(x) \wedge B(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y)))))))$$

El teorema anterior es la sentencia para determinar si una gráfica es 3-coloreable, dentro de la teoría $A_G = \langle \{1, 2, \ldots, n\}, E^G \rangle$

Álgebra relacional

Cuando se hacen consultas en bases de datos usualmente se utilizan sentencias que nos hace recordar de forma inmediata a la lógica de primer orden.
Estas son las las razones:

Álgebra relacional

Cuando se hacen consultas en bases de datos usualmente se utilizan sentencias que nos hace recordar de forma inmediata a la lógica de primer orden.
Estas son las las razones:

- Variantes sintácticas.

Álgebra relacional

Cuando se hacen consultas en bases de datos usualmente se utilizan sentencias que nos hace recordar de forma inmediata a la lógica de primer orden.
Estas son las las razones:

- Variantes sintácticas.
- Álgebra relacional.

Álgebra relacional

Cuando se hacen consultas en bases de datos usualmente se utilizan sentencias que nos hace recordar de forma inmediata a la lógica de primer orden.
Estas son las las razones:

- Variantes sintácticas.
- Álgebra relacional.
- Paralelismo.

Álgebra relacional

Cuando se hacen consultas en bases de datos usualmente se utilizan sentencias que nos hace recordar de forma inmediata a la lógica de primer orden.
Estas son las las razones:

- Variantes sintácticas.
- Álgebra relacional.
- Paralelismo.

Muchas de nuestras <<funciones>> son funciones parciales, entonces se redefinen a relaciones para poder evitar inconsistencias ya que el estado de `error` no es un estado formal de cualquier teoría.

Teoría de tipos

La teoría de tipos nació como marco conceptual para el diseño, análisis e implementación de lenguajes de programación.
Esta teoría ayuda a clarificar conceptos como abstracción de datos, polimorfismo y herencia.

Teoría de tipos

La teoría de tipos nació como marco conceptual para el diseño, análisis e implementación de lenguajes de programación.
Esta teoría ayuda a clarificar conceptos como abstracción de datos, polimorfismo y herencia.
La teoría de tipos es el estudio de sistemas de tipado, los cuales se definen como una forma de gramáticas libres de contexto que imponen restricciones en la creación de programas que aseguran una larga clase de errores, aquellos que surgen por errores semánticos.

Teoría de tipos

La teoría de tipos nació como marco conceptual para el diseño, análisis e implementación de lenguajes de programación.
Esta teoría ayuda a clarificar conceptos como abstracción de datos, polimorfismo y herencia.
La teoría de tipos es el estudio de sistemas de tipado, los cuales se definen como una forma de gramáticas libres de contexto que imponen restricciones en la creación de programas que aseguran una larga clase de errores, aquellos que surgen por errores semánticos.

$$
\begin{aligned}
T &::= \mathit{Nat} \mid \mathit{Bool} \mid T \to T \\
e &::= x \mid n \mid e \circ e \mid \mathit{true} \mid \mathit{false} \mid \mathit{if}\ e\ \mathit{then}\ e\ \mathit{else}\ e \mid \\
&\quad \mathit{fun}\ f(x : T) : T\ \mathit{in}\ e \mid e(e) \\
v &::= x \mid n \mid \mathit{true} \mid \mathit{false} \mid \mathit{fun}\ f(x : T) : T\ \mathit{in}\ e \mid e(e)
\end{aligned}
$$

Razonamiento automatizado

David Hilbert propuso un programa el cual contuviera toda la teoría matemática y este respondiese si un teorema es correcto o no, tal sistema debía ser dado en un lenguaje formal y que al mismo tiempo en correcto y completo.
Esto como respuesta a la fundación de la matemática.
A lo cual Gödel respondió:

Razonamiento automatizado

David Hilbert propuso un programa el cual contuviera toda la teoría matemática y este respondiese si un teorema es correcto o no, tal sistema debía ser dado en un lenguaje formal y que al mismo tiempo en correcto y completo.
Esto como respuesta a la fundación de la matemática.
A lo cual Gödel respondió:

¡JA! . . . cosita

Razonamiento automatizado

David Hilbert propuso un programa el cual contuviera toda la teoría matemática y este respondiese si un teorema es correcto o no, tal sistema debía ser dado en un lenguaje formal y que al mismo tiempo en correcto y completo.
Esto como respuesta a la fundación de la matemática.
A lo cual Gödel respondió:

¡JA! . . . cosita

Pero la idea se analizó y obtuvo resultados como: Demostradores automáticos, verificador de modelos y asistentes de prueba.

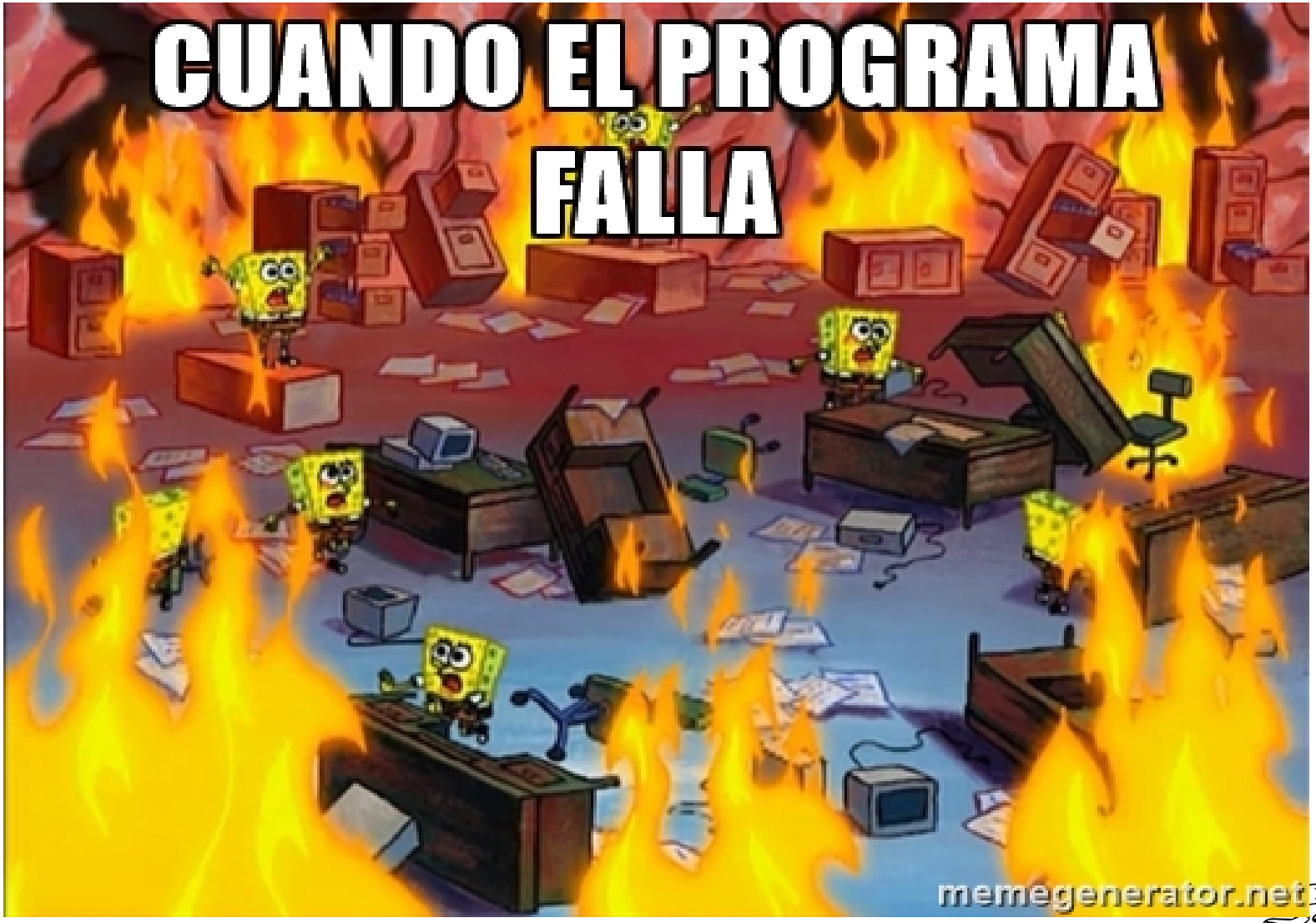

Figura: Evitemos esto...

Lógica proposicional (LP)

Recordemos que la gramática de la lógica proposicional es la siguiente:

$$\varphi \ ::= \ \top \mid \bot \mid x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \varphi \leftrightarrow \varphi$$

con $x \in \texttt{ATOM}$.
Ahora pasemos a `Haskell`