

Verificación formal de arboles rojinegros en Haskell con Coq

David Felipe Hernández Chiapa

Facultad de Ciencias
Universidad Nacional Autónoma de México

5 de diciembre de 2018

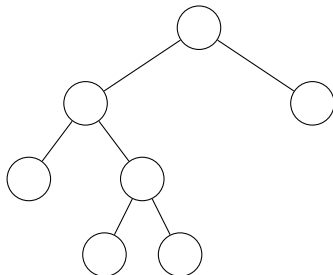


Arboles Binarios

Un Árbol binario es una estructura de datos no lineal, una definición formal de estos seria:

- Gráfica conexa con vértices de grado a lo mas 3.

Ejemplo:



Árbol binario.

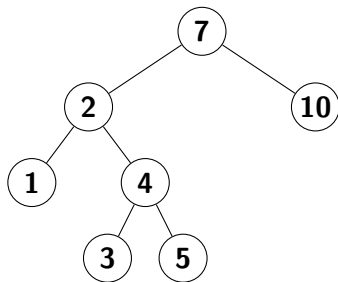
Arboles Binarios de Búsqueda

Un Árbol binario de búsqueda es un caso particular de un árbol binarios, el cual tiene las siguientes invariantes:

- Todos los elementos del subárbol izquierdo son menores al elemento de la raíz.
- Todos los elementos del subárbol derecho son mayores al elemento de la raíz.
- Ambos subárboles también son de búsqueda.

Arboles Binarios de Búsqueda

Ejemplo:



Árbol binario de búsqueda.

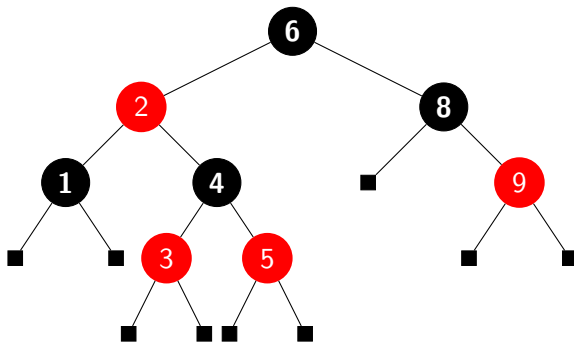
Arboles Rojinegros

Son un caso particular de arboles binarios de búsqueda, estos a su vez incluyen mas invariantes para garantizar una mayor eficiencia en las operaciones que se pueden realizar con ellos. Las invariantes son:

- Todos los vértices son negros o rojos.
- La raíz es negra.
- Todas las hojas son negras, como la raíz.
- Un vértice rojo siempre tiene dos hijos negros.
- Para todo vértice v de este árbol, todas las trayectorias de v a alguna de sus hojas descendientes tiene el mismo número de vértices negros.

Arboles Rojinegros

Ejemplo:



Árbol roji-negro con vacíos.

Verificación de Árboles Rojinegros.

Los lenguajes de programación en los que generalmente se implementan este tipo de árboles son del paradigma imperativo, por ejemplo Java, en los cuales una manera de intentar verificar que la estructura funciona es usar pruebas unitarias.

Pero esto no es verificar.

Verificación formal en Haskell

Haskell es un lenguaje con una base muy grande de desarrolladores que constantemente están generando mas programas escritos en el.

Una de las cosas que se dice de Haskell es que la verificación de su código es bastante sencilla.

¿Pero qué tan cierto y escalable es esto?

Verificación formal en Coq

A diferencia de Haskell, Coq es un asistente de pruebas, con el cual se puede escribir un programa y a la par de demostrar su especificación formal.

Verificación formal en Coq

Las diferencias mas grandes entre la escritura de programas entre Haskell y Coq, es que Coq solo acepta funciones totales y que estas terminen en todos sus casos.

Problema

Nos gustaría una manera de traducir módulos de Haskell (con funciones totales) a Coq para poder verificarlas formalmente de una manera mas sencilla, ordenada y escalable.

hs-to-coq

Es una herramienta en desarrollo por un equipo de la Universidad de Pensilvania (<https://github.com/antalsz/hs-to-coq>).

En esta herramienta ya existen bibliotecas de Haskell traducidas a Coq y también permite traducir cualquier programa de Haskell.

hs-to-coq

Esta herramienta es creada para facilitar la verificación, siguiendo los siguientes pasos:

- 1 Escribir un módulo de Haskell, digamos un módulo de Árboles Rojinegros.
- 2 Probar ese código en Haskell por medio de ejemplos.
- 3 Utilizar hs-to-coq para traducir el código a Coq.
- 4 ¡A verificar!

Ventajas

Esto simplifica mucho la verificación en varios frentes:

- La traducción no se hace a mano.
- La cooperación en un equipo de trabajo se hace mas sencilla.

Ejemplos

Definición de árboles rojinegros en Haskell

```
data Color = R | B
```

```
data RB a = E | T Color (RB a) a (RB a)
```

Ejemplos

Código de balance de Árboles Rojinegros en Haskell.

```
balance :: RB a → a → RB a → RB a
balance (T R a x b) y (T R c z d) = T R (T B a x b) y (T B c z d)
balance (T R (T R a x b) y c) z d = T R (T B a x b) y (T B c z d)
balance (T R a x (T R b y c)) z d = T R (T B a x b) y (T B c z d)
balance a x (T R b y (T R c z d)) = T R (T B a x b) y (T B c z d)
balance a x (T R (T R b y c) z d) = T R (T B a x b) y (T B c z d)
balance a x b = T B a x b
```


Ejemplos

Código de balance de Árboles Rojinegros en Coq, traducido con hs-to-coq.

```

Definition balance {a} : RB a → a → RB a → RB a :=
  fun arg_0__ arg_1__ arg_2__ =>
    match arg_0__, arg_1__, arg_2__ with
    | T R a x b, y, T R c z d => T R (T B a x b) y (T B c z d)
    | T R (T R a x b) y c, z, d => T R (T B a x b) y (T B c z d)
    | T R a x (T R b y c), z, d => T R (T B a x b) y (T B c z d)
    | a, x, T R b y (T R c z d) => T R (T B a x b) y (T B c z d)
    | a, x, T R (T R b y c) z d => T R (T B a x b) y (T B c z d)
    | a, x, b => T B a x b
  end.

```

Ejemplos

Código de inserción de Árboles Rojinegros en Haskell.

```

ins :: Ord a => a -> RB a -> RB a
ins x E = T R E x E
ins x s@(T B a y b)
  | x<y = balance (ins x a) y b
  | x>y = balance a y (ins x b)
  | otherwise = s
ins x s@(T R a y b)
  | x<y = T R (ins x a) y b
  | x>y = T R a y (ins x b)
  | otherwise = s

```

Ejemplos

Código de inserción de Árboles Rojinegros en Coq, traducido con hs-to-coq.

```

Definition ins {a} '{GHC.Base.Ord a} : a → RB a → RB a :=
  fix ins arg_0__ arg_1__
    := match arg_0__, arg_1__ with
      | x, E ⇒ T R E x E
      | x, (T B a y b as s) ⇒
        if x GHC.Base.< y : bool then balance (ins x a) y b else
        if x GHC.Base.> y : bool then balance a y (ins x b) else
        s
      | x, (T R a y b as s) ⇒
        if x GHC.Base.< y : bool then T R (ins x a) y b else
        if x GHC.Base.> y : bool then T R a y (ins x b) else
        s
    end.

```

Trabajo en desarrollo

La herramienta que se usa (hs-to-coq) esta en desarrollo y se le esta integrando mas funcionalidades de Haskell, como clases, polimorfismos, etc.

Este es un trabajo en curso, se busca poder traducir y verificar las operaciones de los arboles rojinegros y tambien as implementacioenes de estos (Implementaciones funcionales de árboles roji-negros, Graciela López Campos).