Braun Trees in Agda

Camilo Andrés Rodríguez Garzón
University EAFIT, Medellín
Colombia

May 25, 2017

Logic and Computation Research Group

What are Braun trees?

Among the many types of balanced binary trees, the Braun trees (Braun & Rem, 1983) are perhaps the most limited. A Braun tree is a binary tree which is as balanced as it can possibly be, every node satisfies the following condition:

- The left subtree has either the same number of nodes as the right subtree or one more.

Braun trees

A binary tree is a Braun tree if:

- It is empty or
- Its left and right subtrees are Braun trees

Braun trees are balanced, their maximum depth is O ($\log_2$ n), where n is the number of elements in the tree.

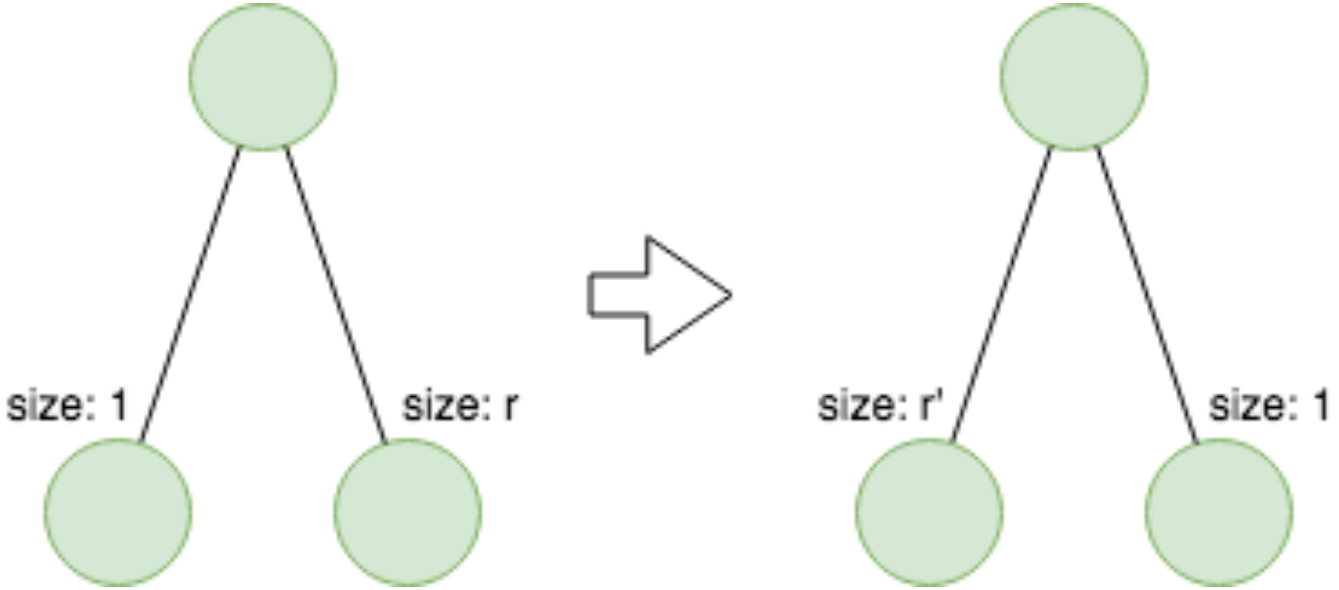
Why in Agda?

With dependent types (Stump, 2015) we can define the type of height-balanced trees of a certain size, the type BraunTree is indexed by a natural number which represents the size of the tree.

BraunTree 0
BraunTree 1
:
:
:
BraunTree n

Property balanced of Braun trees

The trick for maintain the property of balanced Braun trees occur during insertion data.



Data type Braun trees

The index n is the size of the tree (number of elements of type A)

```
postulate
  A    : Set
  _<A_ : A → A → 𝔹

data BraunTree : (n : ℕ) → Set where
  empty : BraunTree 0
  node  : ∀ {m n}
        → A → BraunTree m → BraunTree n
        → m ≡ n ∨ m ≡ suc n
        → BraunTree (suc (m + n))
```

Data type Braun trees

```
postulate
 a : A

data₁ : BraunTree 0
data₁ = empty

data₂ : BraunTree 1
data₂ = node a
                empty
                empty
                (inj₁ refl)
```

```
data₃ : BraunTree 2
data₃ = node a
                (node a
                  empty
                  empty
                  (inj₁ refl))
                empty
                (inj₂ refl)
```

Method of Braun trees

- Insert

```
{- we will keep smaller (_<A_) elements closer to the root of the Braun tree as we insert -}
btInsert : ∀ {n} → A → BraunTree n → BraunTree (suc n)
btInsert x empty = node x empty empty (inj₁ refl)
btInsert x (node{m}{n} y treeₗ treeᵣ p)
  rewrite +comm m n
  with    p         | if x <A y then (x , y) else (y , x)
...  | inj₁ m≡n    | (v₁ , v₂) = node v₁ (btInsert v₂ treeᵣ) treeₗ (inj₂ (cong suc (sym m≡n)))
...  | inj₂ m≡sucn | (v₁ , v₂) = node v₁ (btInsert v₂ treeᵣ) treeₗ (inj₁ (sym m≡sucn))
```

Method of Braun trees

- Insert

```
insert₁ : BraunTree 2
insert₁ = btInsert a
                   (btInsert a empty)

insert₂ : BraunTree 1
insert₂ = btInsert a
                   empty
```

```
insert₃ : BraunTree 3
insert₃ = btInsert a
                   data₃

insert₄ : BraunTree 2
insert₄ = btInsert a
                   data₂
```

Bibliography

- W. Braun. and M . Rem. (1983) A logarithmic implementation of flexible arrays. Memorandum MR83/4. Eindhoven University of Technology.
- A. Stump. (2015) Verified Functional Programing in Agda. [Online]. Disponible: https://play.google.com/books/reader?id=kMwvDAAAQBAJ&printsec=frontcover&output=reader&hl=es&pg=GBS.PP1

THANKS!