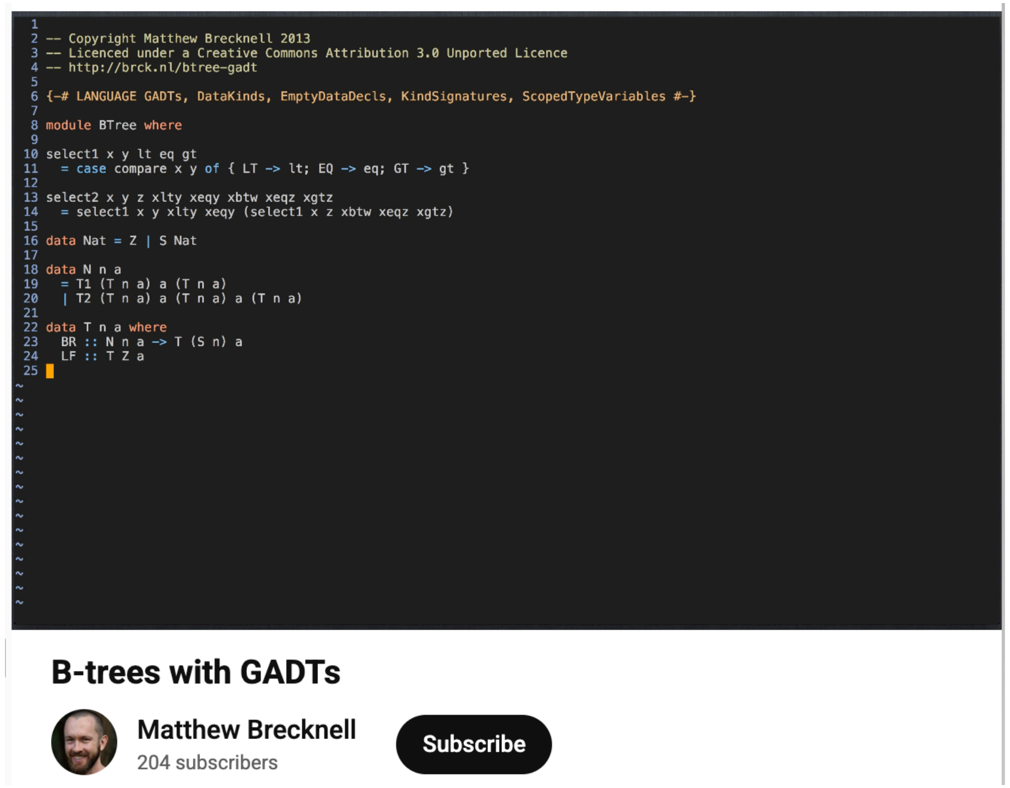
Why Write 10,000 Tests When You Can Write 10 Type Parameters?

What is this talk really about?

- This talk is really an excuse for me to build things that I don't need to but really want to.
- I really wanted to implement some `Tree` data structures and I really wanted to try advanced OCaml features.
- So, when I saw this YouTube video a few weeks ago, all I needed was an excuse to go and do it.

B-trees with GADTs by Matthew Brecknell



```haskell
-- Copyright Matthew Brecknell 2013
-- Licenced under a Creative Commons Attribution 3.0 Unported Licence
-- http://brck.nl/btree-gadt

{-# LANGUAGE GADTs, DataKinds, EmptyDataDecls, KindSignatures, ScopedTypeVariables #-}

module BTree where

select1 x y lt eq gt
  = case compare x y of { LT -> lt; EQ -> eq; GT -> gt }

select2 x y z xlty xeqy xbtw xeqz xgtz
  = select1 x y xlty xeqy (select1 x z xbtw xeqz xgtz)

data Nat = Z | S Nat

data N n a
  = T1 (T n a) a (T n a)
  | T2 (T n a) a (T n a) a (T n a)

data T n a where
  BR :: N n a -> T (S n) a
  LF :: T Z a
```

B-trees with GADTs

Matthew Brecknell

204 subscribers

https://www.youtube.com/watch?v=VIeZW4TSSHg

For real, what is this talk really about?

- How we can maintain invariants and ensure correctness?
- What tools can we leverage for this: unit testing, property testing, fuzz testing, type systems

> Wait, wait, did I read that right? Did you say "type systems"?

Let's talk Binary Search Trees

- A simple tree data structure with a single invariant.
- BST invariant: $left < right$

What are the benefits of using BSTs?

- Excellent data structures for "search" use-cases
- Serves as the underlying implementation for all ordered sets.

Well, what makes BSTs special?

- Really good performance!

Operation	Average	Worst case
Search	Θ(log n)	Θ(n)
Insert	Θ(log n)	Θ(n)
Delete	Θ(log n)	Θ(n)

Is there any way to avoid the worst case?

- Yes! A BSTs performance is dependent on the structure of its branches.
- If we can constrain the structure of the tree, for example, by limiting its height, we can avoid the worst case.
- This is called balancing.
- By maintaining an optimum height of the tree, we can get a great deal of performance.

AVL tree

- AVL Invariant: the height of two subtrees can only differ by 1
-

Operation	Amortized	Worst case
Search	Θ(log n)	Θ(log n)
Insert	Θ(log n)	Θ(log n)
Delete	Θ(log n)	Θ(log n)

- Correctness in this case not only gives us peace of mind, but also performance boost!

More trees?!

- Yes, there are a dozen-or-so balancing schemes that involve height or weight balancing such as Red-Black trees, AA trees, AVL trees and so on...
- Then, there are `BTree`s where nodes can hold more than 2 keys and trees are balanced by merging and splitting nodes.
- There are also many kinds of `BTree`s such as 2-3 tree, B+ tree, B* tree.

How are invariants maintained and correctness ensured?

- But they all have one thing in common: they need certain invariants to be maintained.

We do that using:

- Using runtime checks
- Lots and lots of tests: https://gitlab.com/helsing/software-testing-workshop
- Randomized testing

But what if we could prove that our code is correct?

Don't worry this talk is not about format methods or `Coq`.

Let's talk `Ocaml`

```
type sum =
  | Red
  | Green
  | Blue

type product = {
  red : int;
  green : int;
  blue : int;
}
```

```
enum Sum {
    Red,
    Green,
    Blue
}

struct {
    red: u32,
    green: u32,
    blue: u32,
}
```

Polymorphic types (aka, generics in `Rust`)

```
type 'a option =
  | Some of 'a
  | None
```

```
enum Option<T> {
    Some(T),
    None
}
```

Code detour!

Did you notice?

- We had to be very particular about our property testing inputs?

- The fuzz tester was very untuitive to use?

- The GADT code had no height comparisons?
- The GADT code had no runtime assertions?

Why GADTs?

- GADTs allow us to encode algorithmic invariants into types!

More things to nerd about

- Dependent types
- Fuzz and property testing
- ...

Github

https://github.com/whereistejas/trees-in-gadt

More OCaml stuff?

- https://github.com/whereistejas/ocron

Maybe some Lisp?

- https://gist.github.com/whereistejas/fe2014735f3d4429068e341d54cfbfa6