Last time: abstraction and parametricity

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

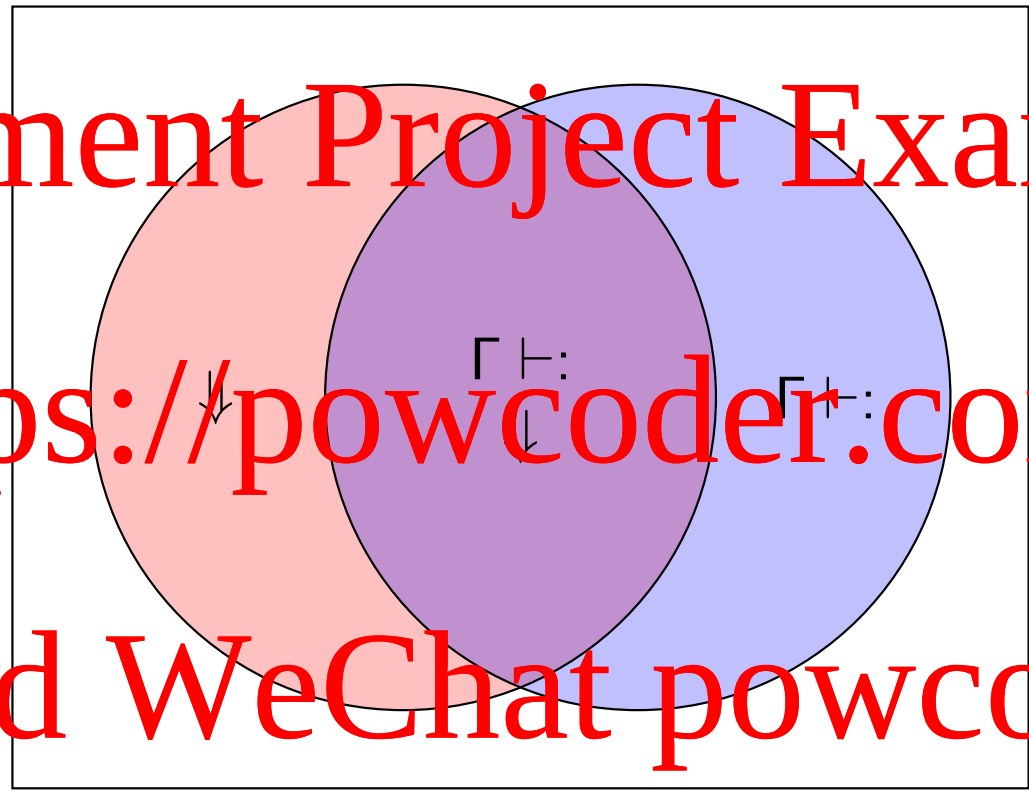
This time: GADTs

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

What we gain

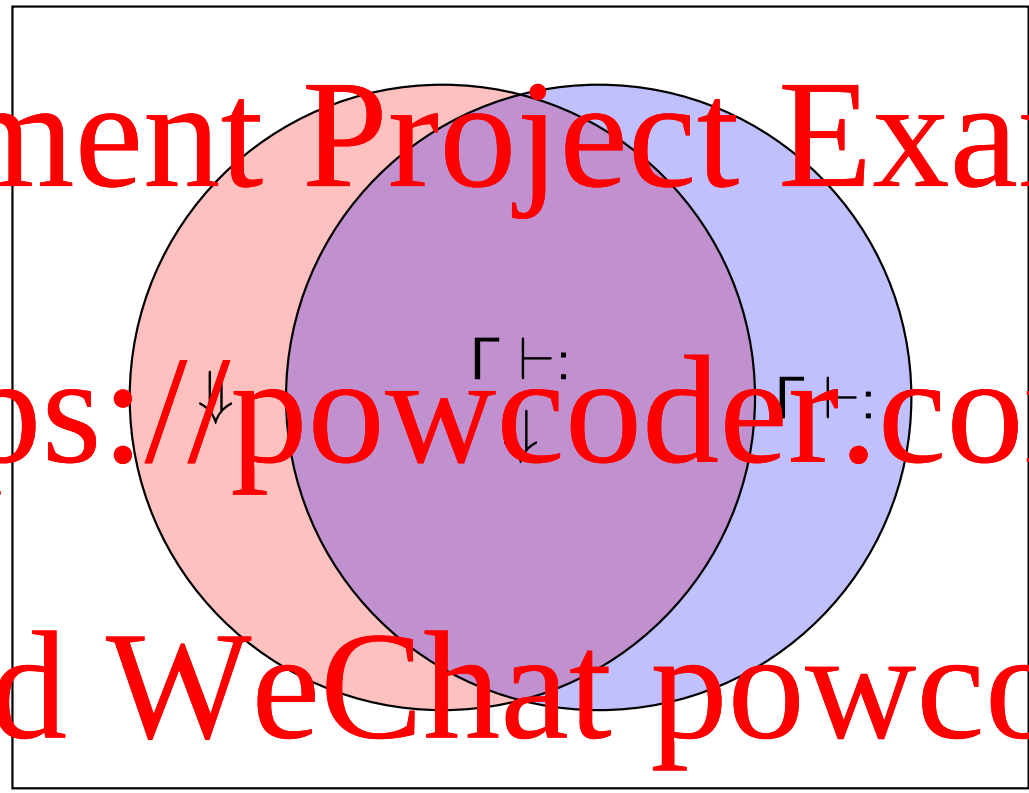


Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

What we gain



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

(Addtionally, some programs become faster!)

What it costs

Assignment Project Exam Help

We'll need to:

- **describe our data** more precisely
- strengthen the **relationship between data and types**
- look at programs through **a propositions-as-types lens**

https://powcoder.com

Add WeChat powcoder

What we'll write

Assignment Project Exam Help

Non-regularity in constructor return types

```
type _ t = T : t1 → t2 t
```

https://powcoder.com

Locally abstract types:

```
let f : type a b. a t → b t = function ...

let g (type a) (type b) (x : a t) : b t = ...
```

Add WeChat powcoder

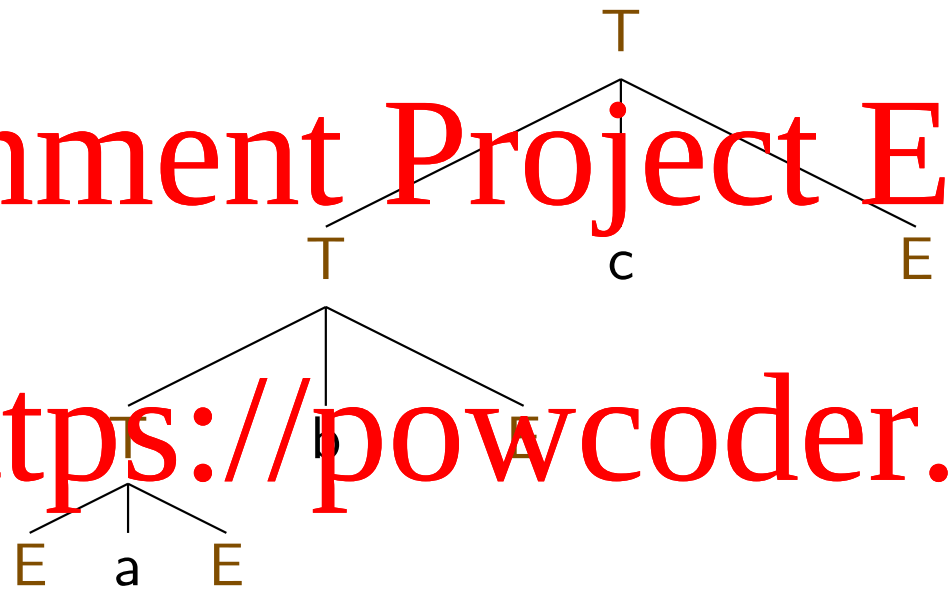
Assignment Project Exam Help

https://powcoder.com

Nested types

Add WeChat powcoder

Unconstrained trees



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type 'a tree =
  Empty : 'a tree
| Tree : 'a tree * 'a * 'a tree → 'a tree
```

Functions on unconstrained trees

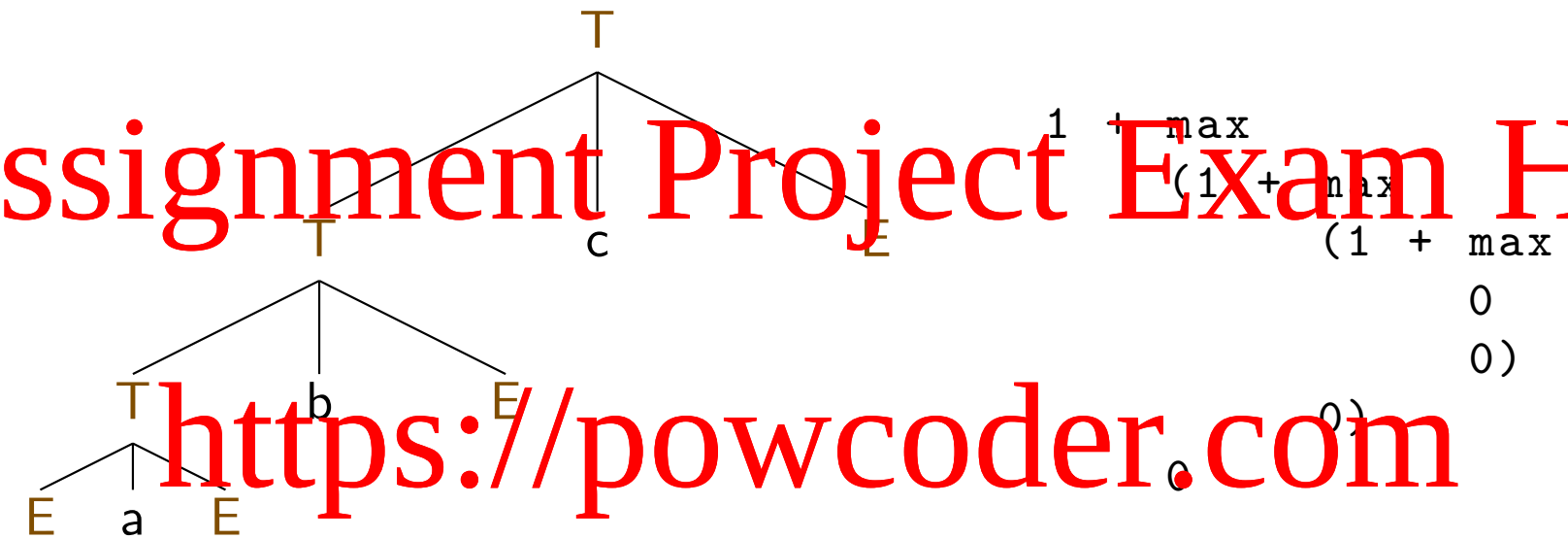
Assignment Project Exam Help

```
val ? : 'a tree → int

val ? : 'a tree → 'a option

val ? : 'a tree → 'a tree
```

https://powcoder.com

Add WeChat powcoder

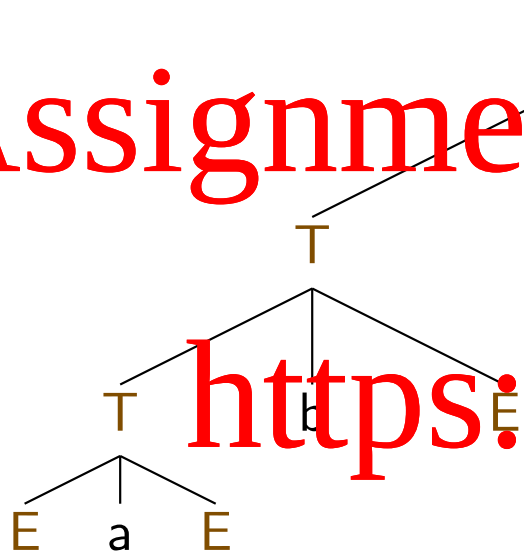
Unconstrained trees: depth



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec depth : 'a.'a tree → int =
 function
    Empty → 0
  | Tree (l,_,r) → 1 + max (depth l) (depth r)
```

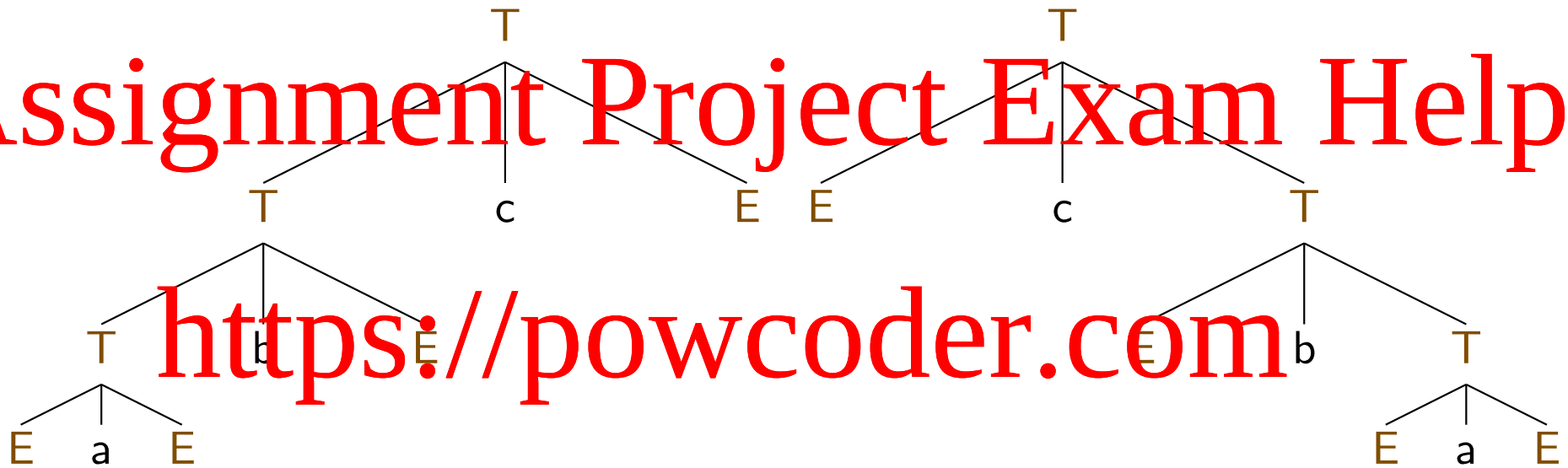
Unconstrained trees: top



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let top : 'a.'a tree → 'a option =
  function
    Empty → None
  | Tree (_,v,_) → Some v
```

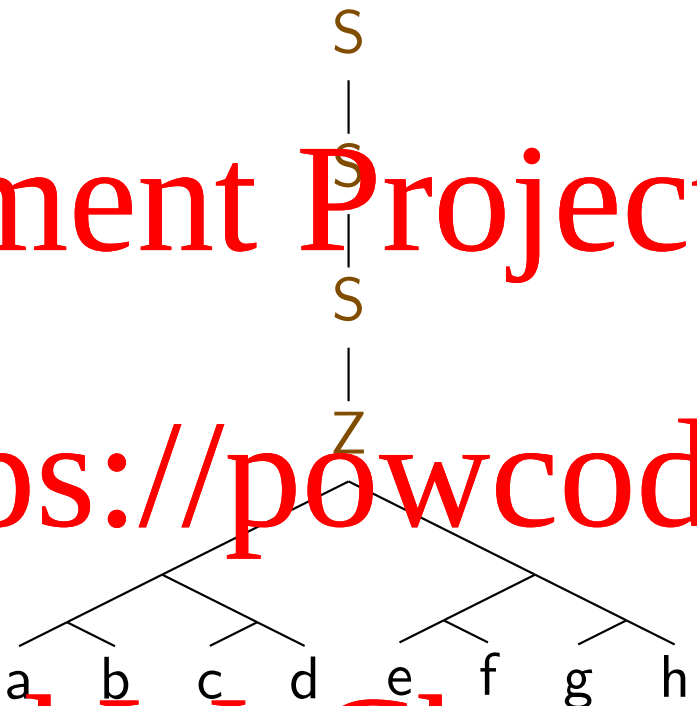
Unconstrained trees: swivel



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec swivel : 'a.'a tree → 'a tree =
 function
   Empty → Empty
 | Tree (l,v,r) → Tree (swivel r, v, swivel l)
```

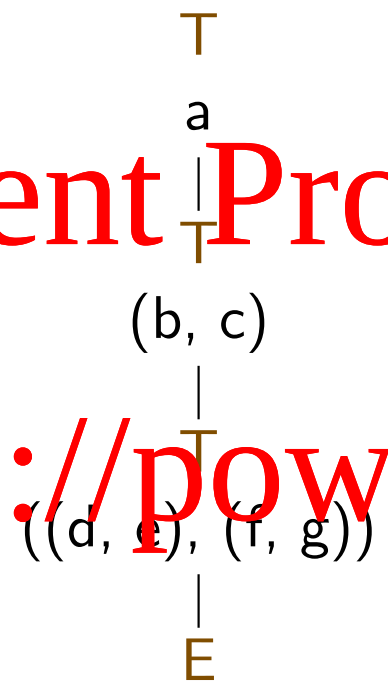
Perfect leaf trees via nesting



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type 'a perfect =
    ZeroP : 'a → 'a perfect
  | SuccP : ('a * 'a) perfect → 'a perfect
```

Perfect (branch) trees via nesting



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type _ ntree =
    EmptyN : 'a ntree
  | TreeN : 'a * ('a * 'a) ntree → 'a ntree
```

Functions on perfect nested trees

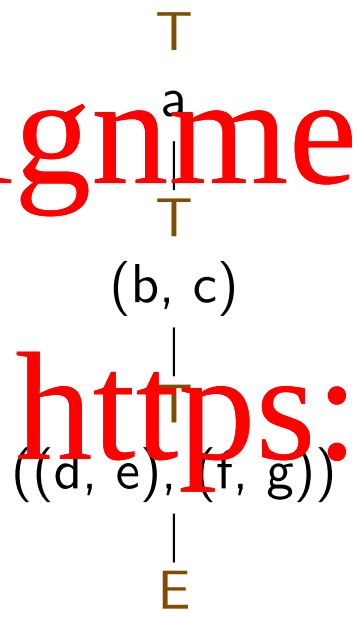
Assignment Project Exam Help

```
val ? : 'a ntree → int

val ? : 'a ntree → 'a option

val ? : 'a ntree → 'a ntree
```

https://powcoder.com

Add WeChat powcoder

Perfect trees: depth

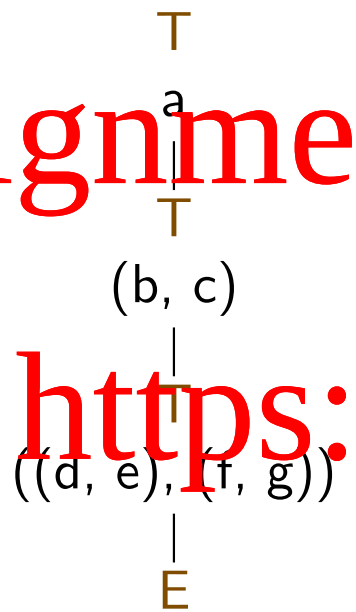



```
1 +
1 +
0
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec depthN : 'a. 'a nrtree → int =
  function
    EmptyN → 0
  | TreeN (_,t) → 1 + depthN t
```

Perfect trees: top



Some a

```
let rec topN : 'a 'a ntree -> 'a option =
  function
    EmptyN → None
  | TreeN (v,_) → Some v
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Perfect trees: swivel

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec swiv : 'a.('a→'a) → 'a ntree → 'a ntree =
  fun f t → match t with
    EmptyN → EmptyN
  | TreeN (v,t) →
    TreeN (f v,swiv (fun (x,y) → (f y, f x)) t)

let swivelN p = swiv id p
```

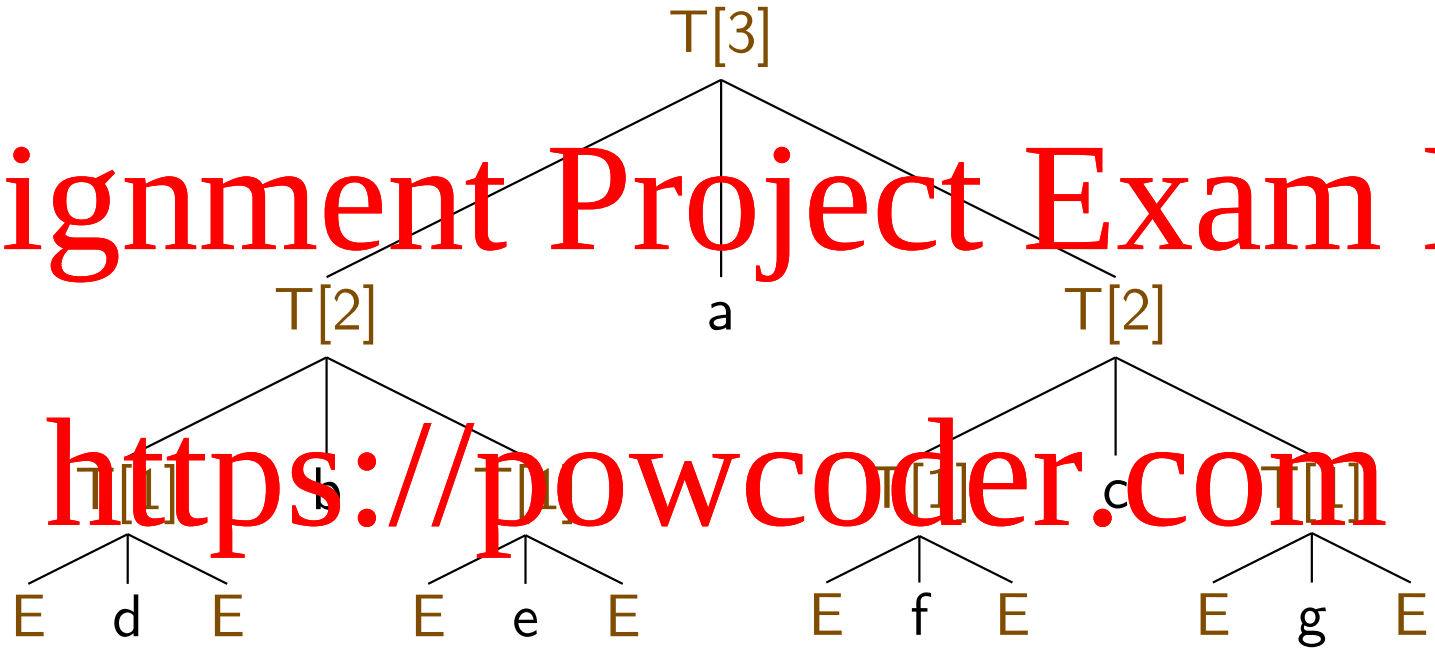
Assignment Project Exam Help

https://powcoder.com

GADTs

Add WeChat powcoder

Perfect trees, take two



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type ('a, _) gtree =
  EmptyG : ('a,z) gtree
| TreeG : ('a,'n) gtree * 'a * ('a,'n) gtree → ('a,'n s)
    gtree
```

Natural numbers

Assignment Project Exam Help

```
type z = Z
type _ s = S : 'n → 'n s
```

https://powcoder.com

```
# let zero = Z;;
val zero : z = Z
# let three = S (S (S Z));;
val three : z s s s = S (S (S Z))
```

Add WeChat powcoder

Functions on perfect trees (GADTs)

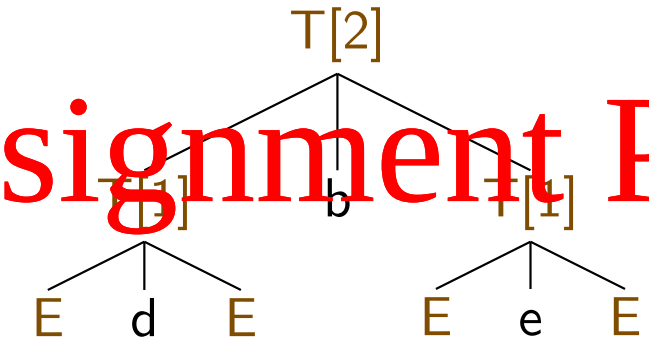
Assignment Project Exam Help

```
val ? : ('a,'n) gtree → 'n

val ? : ('a,'n s) gtree → 'a

val ? : ('a,'n) gtree → ('a,'n) gtree
```

https://powcoder.com

Add WeChat powcoder

Perfect trees (GADTs): depth




```
S (depth
S (depth
Z))
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec depthG : type a n.(a, n) gtree → n =
  function
    EmptyG → Z
  | TreeG (l,_,_) → S (depthG l)
```

Perfect trees (GADTs): depth

```
type ('a, _) gtree =
  EmptyG : ('a,z) gtree
| TreeG : ('a,'n) gtree * 'a * ('a,'n) gtree → ('a,'n s)
    gtree

let rec depthG : type a n.(a, n) gtree → n =
  function
    EmptyG → Z
  | TreeG (l, _, _) → S (depthG l)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Type refinement

In the `EmptyG` branch: `n ≡ z`

In the `TreeG` branch: `n ≡ m s` (for some `m`)

`l : (a, m)gtree`

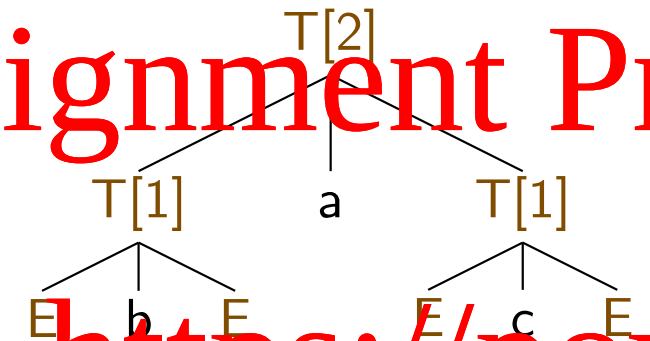
`depthG l : m`

Polymorphic recursion

The argument to the recursive call has size `m` (s.t. `s m ≡ n`)

Perfect trees (GADTs): top

Assignment Project Exam Help



https://powcoder.com

Add WeChat powcoder

```
let topG : type a n.(a,n s) gtree → a =
  function TreeG (_,v,_) → v
```

Perfect trees (GADTs): depth

Assignment Project Exam Help

```
type ('a, _) gtree =
  EmptyG : ('a,z) gtree
| TreeG : ('a,'n) gtree * 'a * ('a,'n) gtree → ('a,'n s)
    gtree

let topG : type a n. (a,n s) gtree → a =
  function TreeG (_,v,_) → v
```

https://powcoder.com

Add WeChat powcoder

Type refinement

In an `EmptyG` branch we would have: `n s ≡ z`

— impossible!

Perfect trees (GADTs): swivel

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec swivelG : type a n.(a,n) gtree2 → (a,n) gtree2 =
 function
    EmptyG → EmptyG
  | TreeG (l,v,r) → TreeG (swivelG r, v, swivelG l)
```

Perfect trees (GADTs): swivel

```
type ('a, _) gtree =
  EmptyG : ('a,z) gtree
| TreeG : ('a,'n) gtree * 'a * ('a,'n) gtree → ('a,'n s)
    gtree

let rec swivelG : type a n.(a,n) gtree → (a,n) gtree =
 function
    EmptyG → EmptyG
  | TreeG (l, v, r) → TreeG (swivelG r, v, swivelG l)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Type refinement

In the `EmptyG` branch: `n ≡ z`

In the `TreeG` branch: `n ≡ m s` (for some `m`)

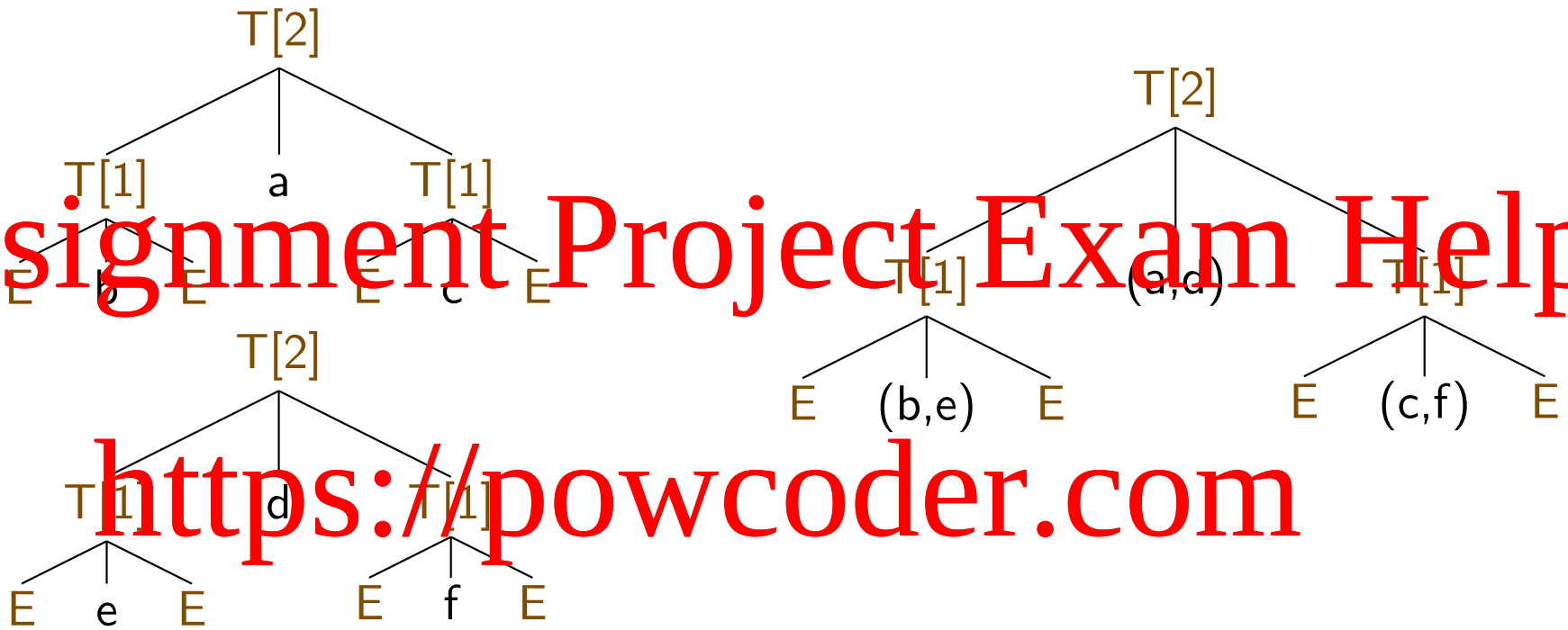
`l, r : (a, m)gtree`

`swivelG l : (a, m)gtree`

Polymorphic recursion

The argument to the recursive call has size `m` (s.t. `s m ≡ n`)

Zipping perfect trees



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec zipTree :
  type a b n.(a,n) gtree → (b,n) gtree →
              (a * b,n) gtree =
  fun x y → match x, y with
    EmptyG, EmptyG → EmptyG
  | TreeG (l,v,r), TreeG (m,w,s) →
    TreeG (zipTree l m, (v,w), zipTree r s)
```

Zipping perfect trees

```
type ('a, _) gtree =
  EmptyG : ('a,z) gtree
| TreeG : ('a,'n) gtree * 'a * ('a,'n) gtree → ('a,'n s)
    gtree

let rec zipTree :
  type a b n.(a,n) gtree → (b,n) gtree → (a * b,n)
    gtree =
  fun x y → match x, y with
    EmptyG, EmptyG → EmptyG
  | TreeG (l,v,r), TreeG (m,w,s) →
    TreeG (zipTree l m, (v,w), zipTree r s)
```

Type refinement

In the `EmptyG` branch: `n ≡ z`

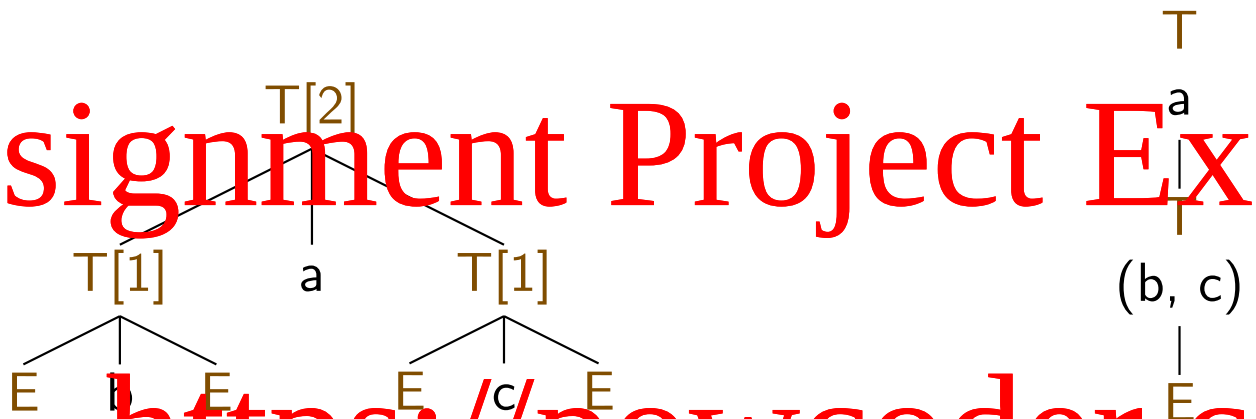
In the `TreeG` branch: `n ≡ m s` (for some `m`)

`EmptyG, TreeG _` produces `n ≡ z` **and** `n ≡ m s`

— impossible!

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

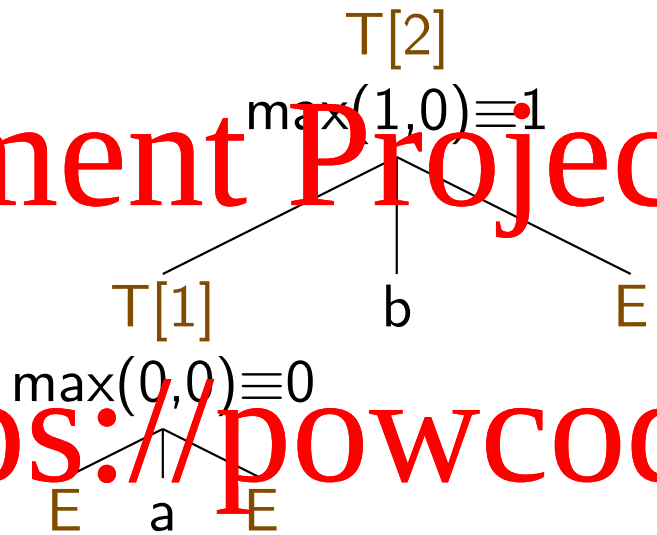
Conversions between perfect tree representations



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec nestify : type a n.(a,n) atree → a ntree =
  function
    EmptyG → EmptyN
  | TreeG (l, v, r) →
    TreeN (v, nestify (zipTree l r))
```

Depth-annotated trees



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type ('a,'z) dtree =
  EmptyD : ('a,z) dtree
| TreeD : ('a,'m) dtree * 'a * ('a,'n) dtree * ('m,'n,'o) max
        → ('a,'o s) dtree
```

The untyped maximum function

Assignment Project Exam Help

```
val max : 'a → 'a → 'a
```

https://powcoder.com

Parametricity: `max` is one of:

```
fun x _ → x
fun _ y → y
```

Add WeChat powcoder

A typed maximum function

Assignment Project Exam Help

```
val max : ('a,'b,'c) max → 'a → 'b → 'c
```

https://powcoder.com

```
(max (a,b)≡ c) → a → b → c
```

Add WeChat powcoder

A typed maximum function: a max predicate

Assignment Project Exam Help

```
type (_,_,_) max =
    MaxEq : ('a,'a,'a) max
  | MaxFlip : ('a,'b,'c) max → ('b,'a,'c) max
  | MaxSuc : ('a,'b,'a) max → ('a s,'b,'a s) max
```

https://powcoder.com

$$a \equiv b \quad \rightarrow \quad max(a,b) \equiv a$$

$$max(a,b) \equiv c \quad \rightarrow \quad max(b,a) \equiv c$$

$$max(a,b) \equiv a \quad \rightarrow \quad max(a+1,b) \equiv a+1$$

Add WeChat powcoder

A typed maximum function

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
type (_,_) eql = Refl : ('a,'a) eql

type (_,_,_) max =
    MaxEq : ('a,'a,'a) max
  | MaxFlip : ('a,'b,'c) max → ('b,'a,'c) max
  | MaxSuc : ('a,'b,'c) max → ('a s,'b,'a s) max

let rec max
  : type a b c.(a,b,c) max → a → b → c
  = fun mx m n → match mx,m with
    MaxEq, _ → m
  | MaxFlip mx', _ → max mx' n m
  | MaxSuc mx' , S m' → S (max mx' m' n)
```

A typed maximum function

```
type (_,_,_) max =
    MaxEq : ('a,'a,'a) max
  | MaxFlip : ('a,'b,'c) max → ('b,'a,'c) max
  | MaxSuc : ('a,'b,'a) max → ('a s,'b,'a s) max

let rec max : type a b c.(a,b,c) max → a → b → c
  = fun mx m n → match mx,m with
    MaxEq , _ → m
    MaxFlip mx', _ → max mx' n m
  | MaxSuc mx' ,S m' → S (max mx' m' n)
```

Type refinement

In the `MaxEq` branch: `a≡b` `a≡c`
`m : c`

In the `MaxFlip` branch: *no refinement*

In the `MaxSuc` branch: `a≡d s`, `c≡d s` (for some `d`)
`mx' : (d, b, d)max`
`m' : d`
`max mx' m' n : d`

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Functions on depth-annotated trees

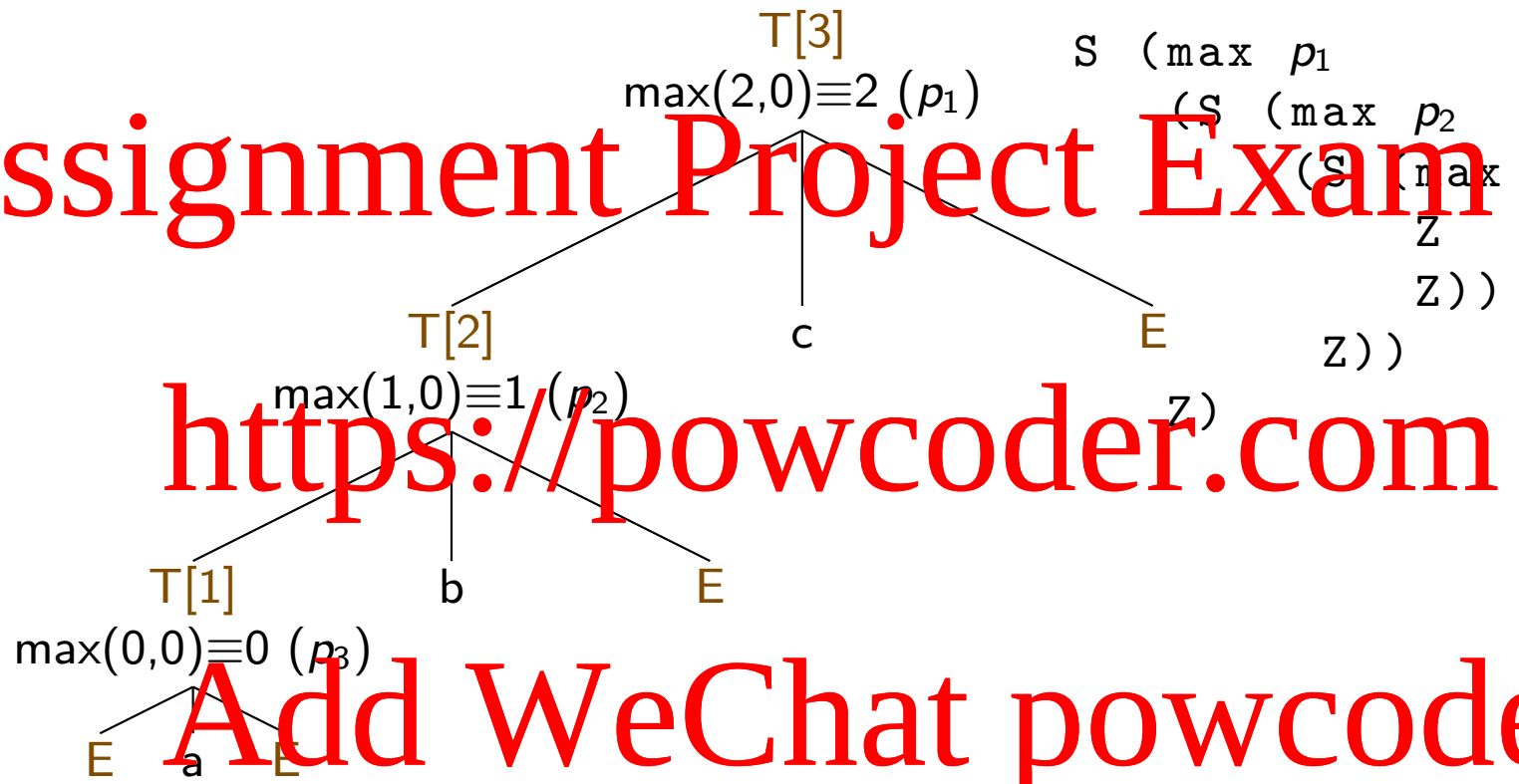
Assignment Project Exam Help

```
val ? : ('a,'n) dtree → 'n

val ? : ('a,'n) dtree → 'a

val ? : ('a,'n) dtree → ('a,'n) dtree
```

https://powcoder.com

Add WeChat powcoder

Depth-annotated trees: depth

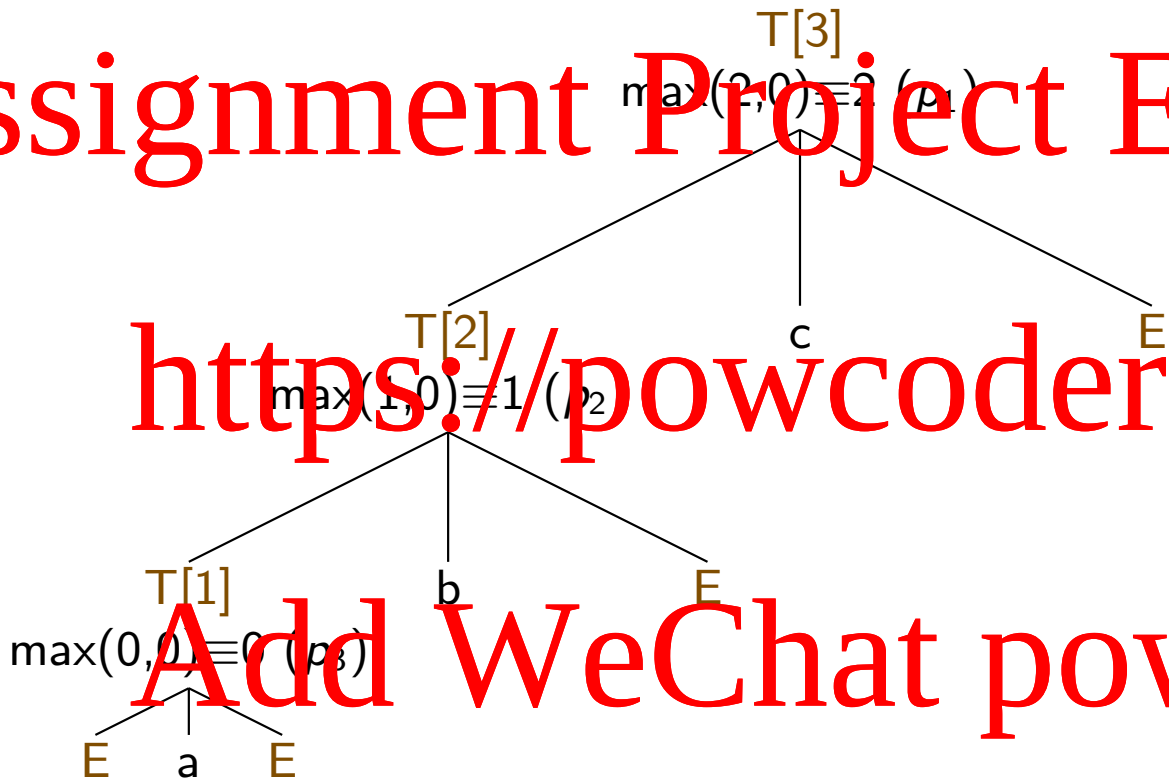


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec depthD : type a n.(a,n) dtree → n =
 function
  EmptyD → Z
| TreeD (l,_,r,mx) → S (max mx (depthD l) (depthD r))
```

Depth-annotated trees: top

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder




```
let topD : type a n.(a,n s) dtree → a =
  function TreeD (_,v,_,_) → v
```

Depth-annotated trees: swivel

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
let rec swivelD :
  type a r.(a,r) dtree → (a,r) dtree =
  function
    EmptyD → EmptyD
  | TreeD (l,v,r,m) →
     TreeD (swivelD r, v, swivelD l, MaxFlip m)
```

Next time

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

GADTs in practice