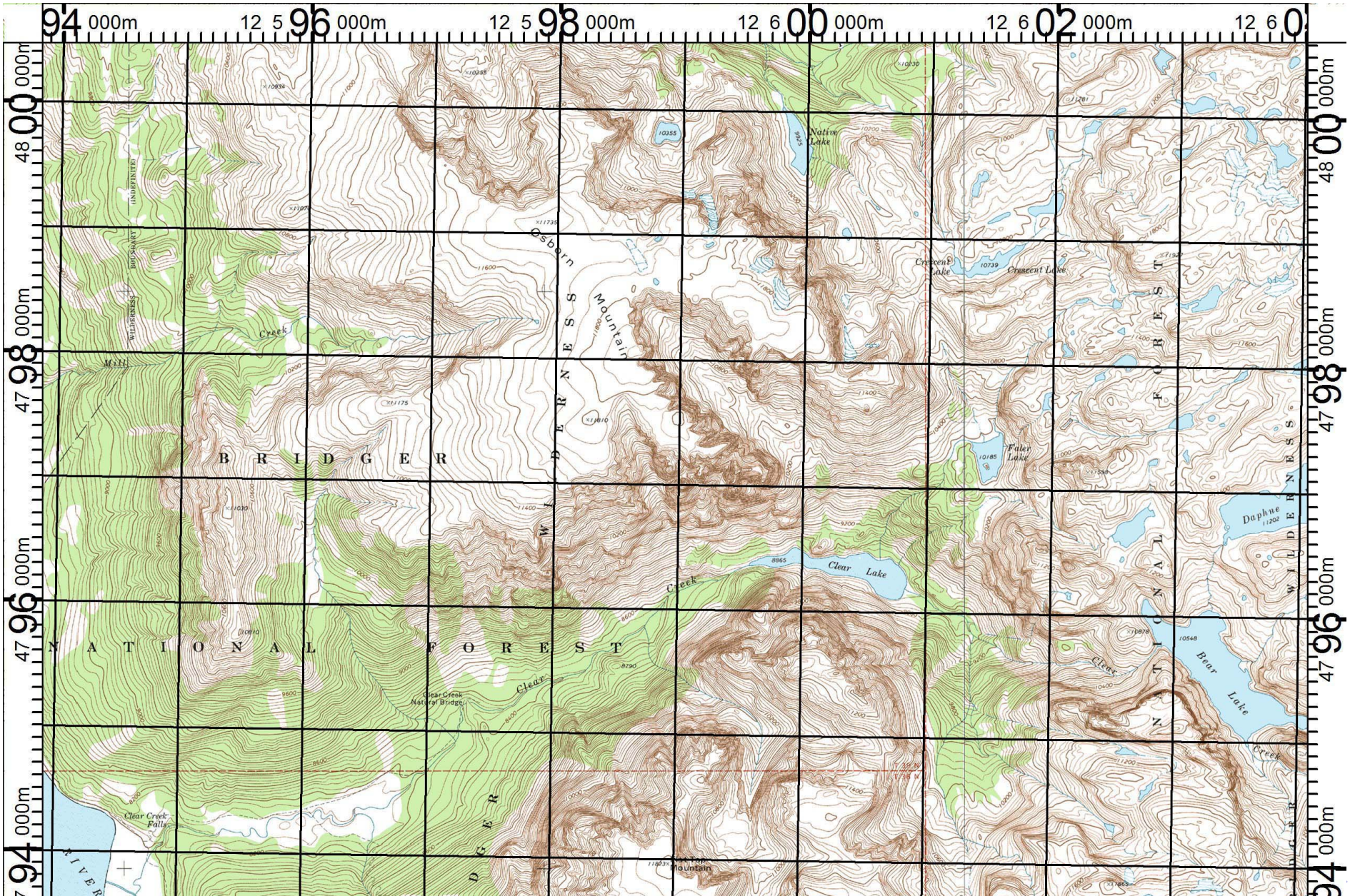
Type-directed search with dependent types

Ben Sherman

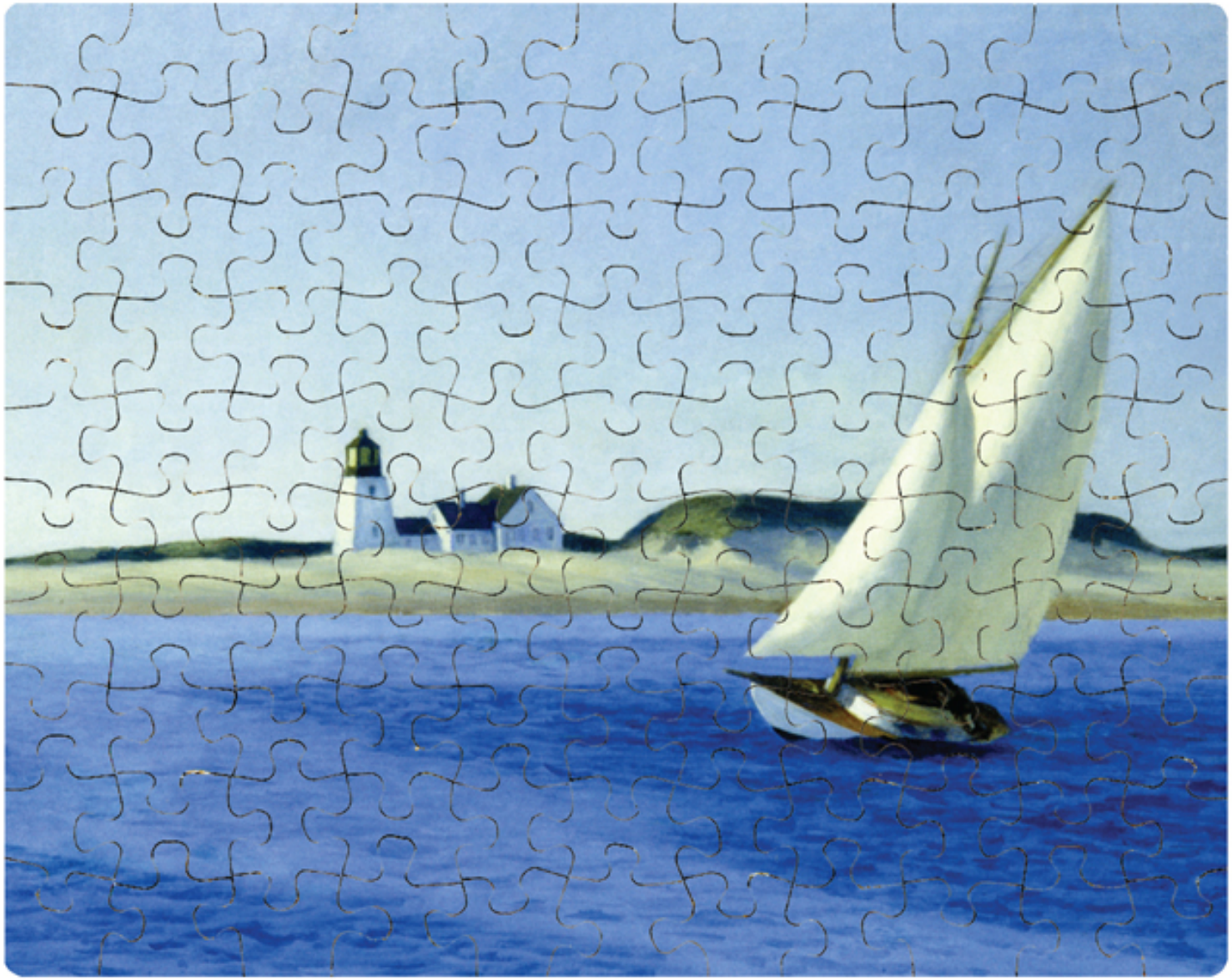
October 18, 2014

Type systems are like jigsaw puzzles

No types


94 000m
96 000m
98 000m
00 000m
02 000m
48 00 000m
47 98 000m
47 96 000m
47 94 000m
B R I D G E R
N A T I O N A L F O R E S T
W I L D E R N E S S
Native Lake
Crescent Lake
Faler Lake
Daphne
Clear Lake
Bear Lake
Clear Creek Falls
Clear Creek Natural Bridge
Osborn Mountain
Mill Creek

Simple types



Expressive types

Sorting a list

Haskell:

```
1 sort :: Ord a ⇒ [a] → [a]
```

Idris (my example, $>$ 150 LOC):

```
1 quickSort : TotalOrder a lte
2           → (xs : List a)
3           → (ys : List a ** (IsSorted lte ys, Permutation xs ys))
```

Type-driven development

Types

- Prove properties stronger than any test can show
- Are documentation that is never wrong or outdated
- Provide an exact specification

Type-directed search

- Tool of choice for type-driven developers
- Can choose your name; can't choose your type!

Distinction without a difference

- Even though $\mathrm{a} \to \mathrm{b} \to \mathrm{c}$ and $(\mathrm{a},\ \mathrm{b}) \to \mathrm{c}$ are distinct types, they "mean the same thing."

Type isomorphism

Definition

Types A and B are *isomorphic* if there are functions

$f : A \to B$ and $g : B \to A$

such that

$(x : A) \to (g \circ f)(x) = x$ and

$(y : B) \to (f \circ g)(y) = y$.

(Type *equivalence* in HoTT)

(What does $=$ mean?)

Isomorphism is not enough!

Suppose we want to compare two values whose type has instance Ord for equality. We search

```
1 Ord a ⇒ a → a → Bool
```

We'd like to find

```
1 (==) :: Eq a ⇒ a → a → Bool
```

Its type is strictly *more* general than what we asked for.

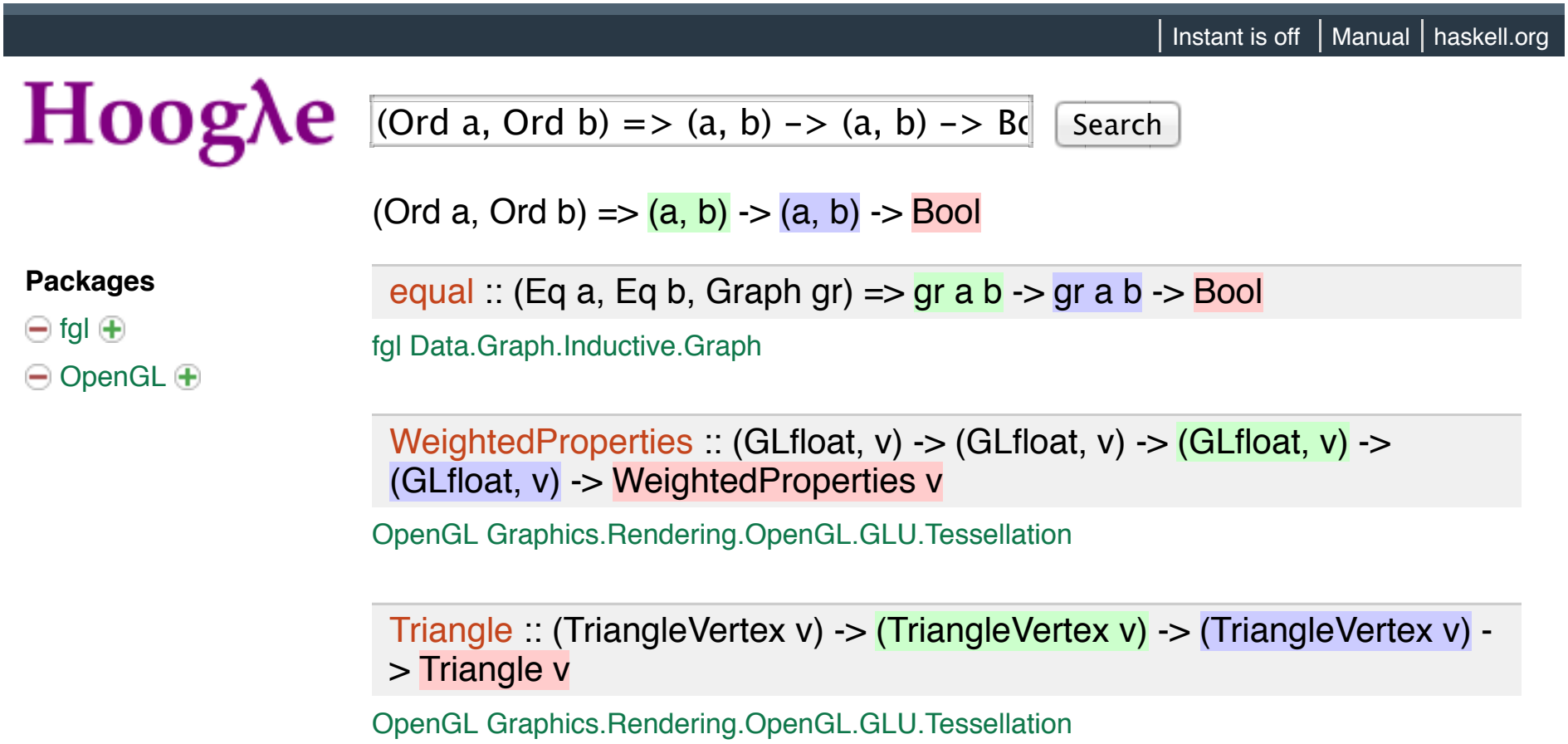
Type containment

We want a partial order $\leq$ that defines isomorphism: that is,

If $A \leq B$ and $B \leq A$, then $A \cong B$.

Demo

Beating Hoogle



"Kind" search for free

Instant is off | Manual | haskell.org

Hoogλe

* -> *

Search

* -> *

Parse error: (line 1, column 2): unexpected " " expecting letter

For information on what queries should look like, see the user manual.

The algorithm

- Try messing with the types in lots of different ways.
- Find the best results first (Dijkstra's algorithm)!

(More details: `https://github.com/idris-lang/Idris-dev/wiki/Type-directed-search-(:search)`)

Next up?

- Produce the corresponding "data" for the search results
- Inlining single-constructor datatypes
- Find isomorphic datatypes
- What do **you** think?

Pi in the sky

- Big database of libraries (with code that feels like programs and code that feels like proofs)
- Type-driven development
- Search the types you must implement; if there's a result, use the library with confidence that it meets the specification