Hoogλe
Finding Functions from Types

Neil Mitchell
haskell.org/hoogle
community.haskell.org/~ndm/
Hooble Synopsis

“Hooble is a Haskell API search engine, which allows you to search many standard Haskell libraries by either function name, or by approximate type signature.”

Or, Google for Haskell libraries
Solving the Jigsaw

"static typing is ... putting pieces into a jigsaw puzzle"

Real World Haskell

Find a function to go here
Which function do we want?

1. \( a \rightarrow [(a,b)] \rightarrow b \)

2. \([\text{Int}] \rightarrow \text{String}\)

3. \(\text{Set } a \rightarrow a \rightarrow \text{Bool}\)

4. \(\text{Ord } a \Rightarrow [a] \rightarrow [a]\)

5. \(\text{Char} \rightarrow \text{Bool}\)

6. \((a \rightarrow b) \rightarrow [a] \rightarrow [b]\)
The Problem

Given a type signature, rank a set of functions with types by appropriateness

Order types by closeness, efficiently

Heuristics/Psychic powers

Algorithms
String: Ordering by closeness

- Equality, perhaps case insensitive
- Prefix/Suffix/Substring matching
- Levenshtein/edit distance
- Tries, KMP, FSA, Baeza-Yates…

\[
\text{search} :: [(\text{String}, \varphi)] \rightarrow (\text{String} \rightarrow [\varphi])
\]
String: Edit Distance

• How many “steps”
  – Insertion or deletion
  – Substitution (just a cheap insert and delete?)

Hello $\approx$ Hell
Hell $\approx$ Sell

• $O(nm)$, result is bounded by $\max(n,m)$
Type: Ordering by closeness

Ignoring performance, we can write:

match :: Type → Type → Maybe Closeness

How “close” are two Type values?

(May not be commutative)
Brainstorm

match :: Type → Type → Maybe Closeness

What is Closeness?  
How is it calculated?
Ideas

• Alpha equality (Hoogo 1)
• Isomorphism (Rittri, Runciman - 1980’s)
• Textual type searching (Hayoo!)
• Unification (Hoogo 2)
• Edit distance (Hoogo 3)
• Full edit distance (Hoogo 3.5)
• Structural edit distance (Hoogo 4)
• Result indexed edit distance (Hoogo 5)
Alpha equality

• Take a type signature, and “normalise” it
• Rename variables to be sequential
• Then do an exact text match

• $k \rightarrow v \rightarrow \text{Map } k \; v$
• $a \rightarrow b \rightarrow \text{Map } a \; b$

No psychic powers
Isomorphism

- Only match types which are isomorphic
  - Long before type classes
- Ismorphism is about equal structure
  - $a \to b \to c \equiv (a, b) \to c$

uncurry :: $(a \to b \to c) \to (a, b) \to c$

:: $(a \to b \to c) \to a \to b \to c$

Less useful for modern code
Textual Type Searching

• Alpha normalise + strength reduced alpha normalisation

• $k \rightarrow v \rightarrow \text{Map } k \ v$

• $a \rightarrow b \rightarrow \text{Map } a \ b \ & \ a \rightarrow b \rightarrow c \ a \ b$

• Plus substring searching
Unification

• Unify against each result, like a compiler
• The lookup problem:
  – $a \rightarrow [(a,b)] \rightarrow b \neq a \rightarrow [(a,b)] \rightarrow \text{Maybe b}$

• Works OK, but not great, in practice
  – More general is fine, what about less general?
  – $a \equiv \text{everything}$?
  – is undefined really the answer?

Not what humans want
Edit Distance

• What changes do I need to make to equalise these types
• Each change has a cost

\[
a \rightarrow [(a,b)] \rightarrow b
\]

\[
a \rightarrow [(a,b)] \rightarrow \text{Maybe } b
\]

\[
\text{Eq } a \Rightarrow a \rightarrow [(a,b)] \rightarrow \text{Maybe } b
\]

A nice start, lots of details left
Ideas Compared

- Alpha equality
- Unification
- My Type
- Edit distance

Textual search = superset of alpha equality
All but Textual search can have argument reordering added
Edit Distance Costs

• Alias following (String $\leftrightarrow$ [Char])
• Instances (Ord $a \Rightarrow a \leftrightarrow a$)
• Subtyping (Num $a \Rightarrow a \leftrightarrow \text{Int}$)
• Boxing ($a \leftrightarrow m\ a$, $a \leftrightarrow [a]$)
• Free variable duplication ($\langle(a,b) \leftrightarrow (a,a)\rangle$)
• Restriction ($[a] \leftrightarrow m\ a$, Bool $\leftrightarrow a$)
• Argument deletion ($a \rightarrow b \rightarrow c \leftrightarrow b \rightarrow c$)
• Argument reordering
Edit Distance Examples

\[ [\text{Int}] \rightarrow \text{String} \]

\[ \text{Int} \rightarrow \text{String} \]

\[ [\text{Int}] \rightarrow [\text{Char}] \]

\[ \text{Show a } \Rightarrow a \rightarrow \text{String} \]

\[ [\text{a}] \rightarrow [\text{Char}] \]

\[ [\text{a}] \rightarrow [\text{b}] \]

\[ ([\text{a}] \rightarrow [\text{b}]) \rightarrow [\text{a}] \rightarrow [\text{b}] \]
A note on “subtype”

\[ \text{Num } a \Rightarrow a \rightarrow a \]

\[ \text{Double } \rightarrow \text{Double} \]

\[ a \rightarrow a \]

Given instance Num Double:

\[ \text{Double } \subset (\text{Num } a \Rightarrow a) \subset a \]
A note on “boxing”

- Eq \( a \Rightarrow a \rightarrow [a] \rightarrow \text{Int} \)
- Eq \( a \Rightarrow a \rightarrow [a] \rightarrow \text{Maybe\ Int} \)
- Eq \( a \Rightarrow a \rightarrow [a] \rightarrow [\text{Int}] \)

Most boxes add a little info:
- Maybe - this might fail/optional arg
- List - may be multiple results
- IO - you need to be in the IO monad
Edit Distances

• Which types of edits should be used?
  – Lots of scope for experimentation

• Can the edits be implemented efficiently?

• What environment do we need?
  – Aliases? Instances?
Ordering Closeness

type Closeness = [Edit]

compare ::
    Closeness → Closeness → Ordering

compare = compare `on` score

score :: Closeness → Double

score = sum . map rank

rank :: Edit → Double
Ranking Edits

• Initial attempt: Make up numbers manually
  – Did not scale at all, hard to get right, like solving a large constraint problem in your head

• Solution: Constraint solver!
Ranking Examples

• Keep a list of example searches, with ordered results
• When someone complains, add their complaint to this list

• Generate a set of constraints, then solve
  – I use the ECLiPSe constraint solver
Performance Target:

As-you-type searches against all current versions of all Haskell libraries
Naive Edit Distance

\[ (t, x) \leftarrow \text{database} \]
\[ \text{Just } c \leftarrow [\text{match user } t] \]
\[ \text{order by } c \]

- let \( n = \text{length database} \)
  - \( \Theta(n) \) to search all items (ignoring sort)
  - \( \Theta(n) \) to find the best result

\( n = 27,396 \) today
(target of 296,871)
Decomposing Edit Distance

Functor $f \Rightarrow (a \rightarrow b) \rightarrow f\ a \rightarrow f\ b$

- subtyping/context
- different variables
- swap arguments
- same variables
Interactive Lists

data Barrier o α = Value o α | Barrier o

Given (Barrier o₁:xs),
∀Value o₂ x ∈ xs, o₁ < o₂

bsort :: Ord o ⇒ [Barrier o α] → [α]
Per Argument Searching

• The idea: Search for each argument separately, combine the results

  a → b → c

• combine $ search arguments a `merge` search arguments b `merge` search results c

Use interactive lists for search/combine
Implementing Search

- Have type graphs, annotated with costs
  - Dijkstra’s graph search algorithm
Implementing Combine

• Combine is fiddly
• Needs to apply costs such as instances, variable renaming, argument deletion
• Check all arguments are present
• Ensure no duplicate answers

• Fast to search for the best matches
The Problem

- Finds the first result quickly
- Graphs may be really big
- But a particular search may match many results in many ways
  - Finding all results can take some time
  - 5000 functions, ~5 seconds

- Need to be more restrictive with matching
Structure Matching

• We can decompose any type into a structure and a list of terms
  
  Either (Maybe a) (b,c)
  
  \[ \equiv \ ? \ (? \ ?) \ (? \ ? \ ?) + \text{Either Maybe a (,) b c} \]

• Can now find types quickly
  
  – 22 distinct argument structures in base library
  – Very amenable to hashing/interning
  – Not as powerful as edit distance
Structure + Aliases

String ≈ [Char]

? + String ≠ ?? + [] Char

• Solution: Expand out all aliases
  – Penalise for all mismatched aliases used
  – i.e. left uses String, but right doesn’t
  – Imprecise heuristic
Structure + Boxing

Maybe a ≈ a
? ? + Maybe a = ? + a

• Solution: Only allow top-level boxes
  – Maybe [a] ≠ Maybe a
  – Now have at most 3 structure lookups

Boxing is 3x expensive
Step 1: Restrict Search

- Use structure for type search
- Many fewer answers
  - 5,000 types, ~0.5 seconds
- Target: 300,000 types, ~0.1 seconds
Step 2: Restrict Combine

- Start by looking at the result first

Map Structure
(Map Int
([(Type,[(Structure,Type)],[φ]]))

Not yet finished implementation
The Hoogle Tool

• Over 6 years old
• 4 major versions (each a complete rewrite)
  – Version 1 in Javascript, 2-4 in Haskell

• Web version
• Firefox plugin, iPhone version, command line tool, custom web server
Hooogle Statistics

- 1.7 million searches up until 1st Jan 2011
- Between 1000 to 2500 a day
Academia + Real World

• Academia
  – Theory of type searching

• Real World
  – Generating databases of type signatures
  – Web server, AJAX interface, interactivity
  – Lots of user feedback, including logs
  – 1/6 of searches are type based
Fixing User Searches

- double to integer

  Did you mean: Double → Integer

- where

  keyword where
Conclusions

• I now use Hoogle every day
  – Name search lets you look up types/docs
  – Type search lets you look up names
  – Both let you find new functions

• Edit distance works for type search

• Having an online search engine is handy!
  haskell.org/hoogle
Funny Searches

- eastenders
- california public schools portable classes
- Bondage
- diem chuan truong dai hoc su pham ha noi 2008
- Messenger freak
- ebay consistency version
- Simon Peyton Jones Genius
- free erotic storeis
- videos pornos gratis
- gia savores de BARILOCHE
- name of Peanuts carton bird
- Colin Runciman