stackoverflow

149

How to split a string in Haskell?

How do I split a string on a custom separator? I want the follo[illegible]ehavior:

```
split ‘,’ “my,comma,separated,list”  → [[illegible]ed”, “list”]
```

wanted: automatic programmer's assistant

157

You can implem[illegible]

```
split :: Char -> String -> [String]
split c s = case dropWhile (== c) s of
                  "" -> []
                  s' -> w : split c s’’
  where (w, s'') = break (== c) s'
```

api search with hoogle

Hoogλe

`(a -> b) -> [a] -> [b]`

Search

```
map :: (a -> b) -> [a] -> [b]
```

Data.List

`map f xs` is the list obtained by applying `f` to each element of `xs`, i.e.,

```
map f [x1, x2, ..., xn] == [f x1, f x2, ..., f xn]
```

hoogle needs synthesis!

Hoogλe

```
(a -> Maybe b) -> [a] -> Int
```

Search

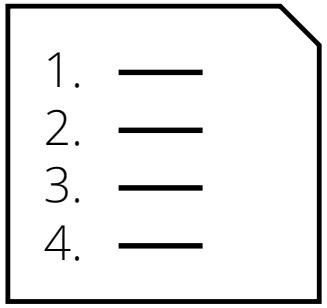
No results found!

hoogle+

specification

```
(a -> Maybe b)
  -> [a] -> Int
```

programs



Haskell libraries

future work

1. scaling up the search
2. eliminating irrelevant results
3. result comprehension
4. user interaction

future work

1. scaling up the search
2. eliminating irrelevant results
3. result comprehension
4. user interaction

three kinds of irrelevant results

type query: `xs:[Maybe a] → d:a → a`

1. discard arguments

```
\xs d → d
```

polymorphism
makes this worse

2. always crash

```
\xs d → foldr (head []) d xs
```

3. duplicates

```
\xs d → fromMaybe d (head xs)
```

```
\xs d → maybe d id (head xs)
```

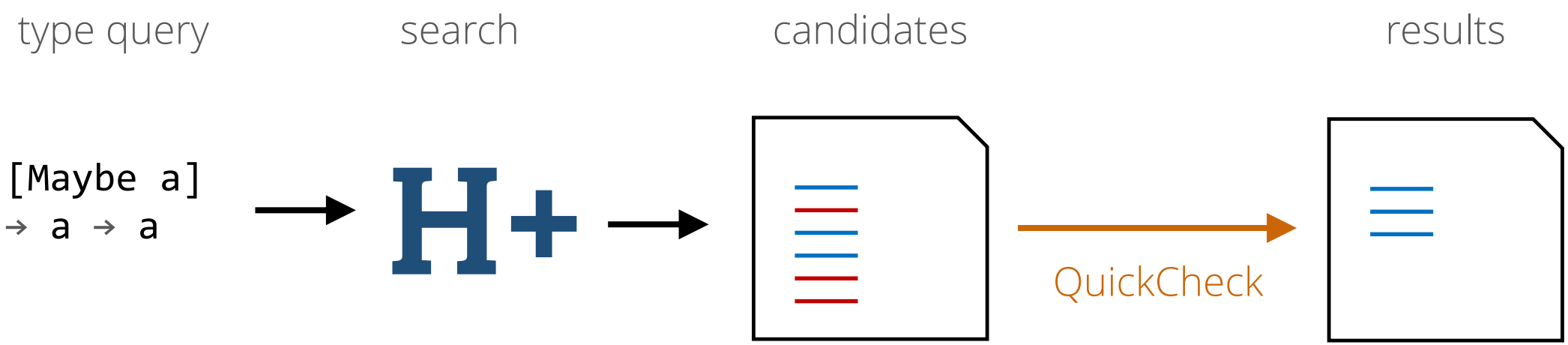
substructural types

program	variable use	structural unrestricted	relevant *at least* once	linear *exactly* once
`\xs d → d`		✅	❌	❌
`\xs d → bool d d (null xs)`		✅	✅	❌
`\xs d → fromMaybe d (listToMaybe (catMaybes xs))`		✅	✅	✅

definitely useless

likely useless

test-based filtering



1. find an input where c returns a value vs crashes/diverges (tricky with infinite data structures!)
2. find an input where c1 and c2 are different

future work

1. scaling up the search
2. eliminating irrelevant results
3. result comprehension
4. user interaction

future work: comprehension

Hoogλe+

```
xs:[Maybe a] -> x:a -> a
```

Search

```
fromMaybe x (head xs)
```

```
fromMaybe x (last xs)
```

```
fromMaybe x (listToMaybe (catMaybes xs))
```

how do I know what these programs do?

future work: comprehension

Hoogλe+

```
xs:[Maybe a] -> x:a -> a
```

Search

```
fromMaybe x (head xs)
```

```
[]                0 -> error
[Just 5, Nothing] 0 -> 5
```

```
fromMaybe x (last xs)
```

```
[]                0 -> error
[Just 5, Nothing] 0 -> 0
```

```
fromMaybe x (listToMaybe (catMaybes xs))
```

```
[]                0 -> 0
[Just 5, Nothing] 0 -> 5
```

future work

1. scaling up the search
2. eliminating irrelevant results
3. result comprehension
4. user interaction

Hoogle+

Welcome to the Hoogle+ Demo

Hoogle+ is a type-driven synthesis engine for Haskell - like Hoogle but able to find compositions of functions. Given a Haskell type, Hoogle+ generates terms that inhabit this type by composing library components. It supports polymorphism, type classes, and higher-order functions. Available library components are listed in the side bar.

n:Int -> xs:[a] -> [a]

Example Specifications:

New	arg0	arg1	result

No data

Arguments

- +

Search

1

\n xs -> GHC.List.drop n xs

drop n xs returns the suffix of xs after the first n elements, or [] if n > length xs: drop 6 "Hello World!" == "World!" drop 3 [1,2,3,4,5] == [4,5] drop 3 [1,2] == [] drop 3 [] == [] drop (-1) [1,2] == [1,2] drop 0 [1,2] == [1,2] It is an instance of the more general genericDrop, in which n may be of any integral type.

		arg0	arg1	result
Edit	Keep usage	4	[1,2,3]	[]
Edit	Keep usage	2	[1,1,1]	[1]
Edit	Keep usage	2	[Just 5, Nothing, Just 1]	[Just 1]

drop :: Int -> [a]

n :: Int

xs :: [a]