Suggesting Valid Substitutions for Typed Holes

Improving discoverability when working with libraries in Haskell

Matthías Páll Gissurarson

Department of Computer Science and Engineering
Chalmers University of Technology

Lambda Days, 2018-02-23

lambda DAYS

What are Typed Holes?

- A typed hole is a "hole" denoted by `_` in the code which can be used to find terms that "fit" the hole.
- Available in `GHC` version `7.8.1`
- Let's look at the hole in the following code:

```
f :: [String]
f = _ "hello, world"
```

If you compile with `GHC` version `8.2.1`, you'll get

- `Found hole: _ :: [Char] -> [String]`
- ```
  In the expression: _
  In the expression: _ "hello, world"
  In an equation for ‘f’: f = _ "hello, world"
  ```
- ```
  Relevant bindings include
    f :: [String] (bound at t.hs:2:1)
  ```

lambda DAYS

What are Typed Holes?

- The message tells you the type of the hole:
  - `Found hole: _ :: [Char] -> [String]`
- Where it occurs:
  - ```
    In the expression: _
    In the expression: _ "hello, world"
    In an equation for ‘f’: f = _ "hello, world"
    ```
- And any "relevant" bindings in local scope:
  - ```
    Relevant bindings include
      f :: [String] (bound at t.hs:2:1)
    ```

lambda DAYS

DEMO

What are Valid Substitutions?

- A valid substitution is something that you can replace the hole with directly, and the program will type check.
- Will be available in `GHC` version `8.4.1`
- Let's look the following code again.

```
f :: [String]
f = _ "hello, world"
```

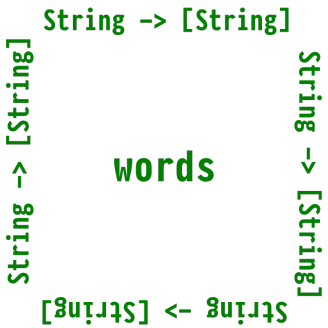
- Compiling this will now get you a list of valid substitutions:

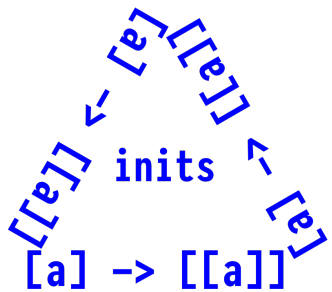
```
Valid substitutions include
  lines :: String -> [String]
  words :: String -> [String]
  inits :: [a] -> [[a]]
  read :: Read a => String -> a
  repeat :: a -> [a]
  return :: Monad m => a -> m a
  ...
```

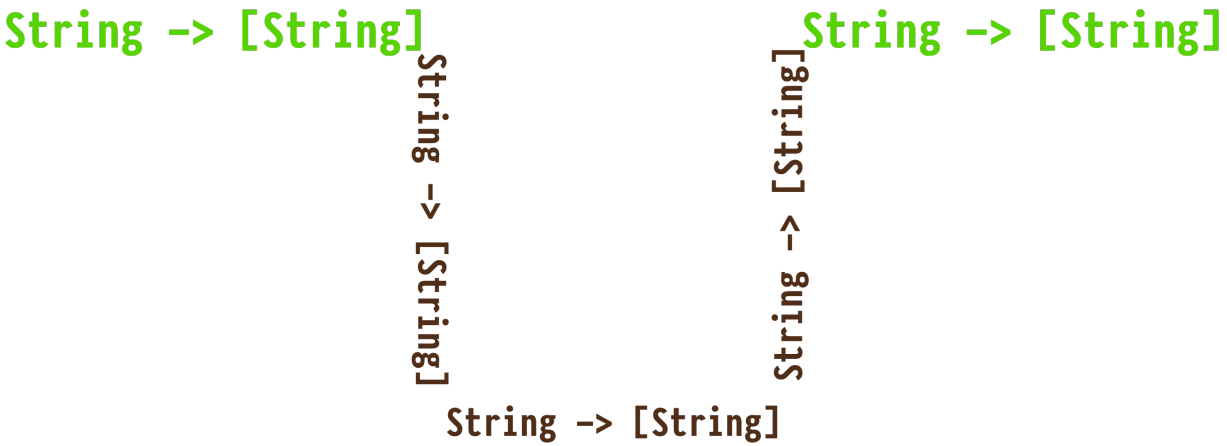
lambda DAYS

What are Valid Substitutions?

- A valid substitution is not only items with the exact same type of the hole, but polymorphic functions that can be made to fit the hole.







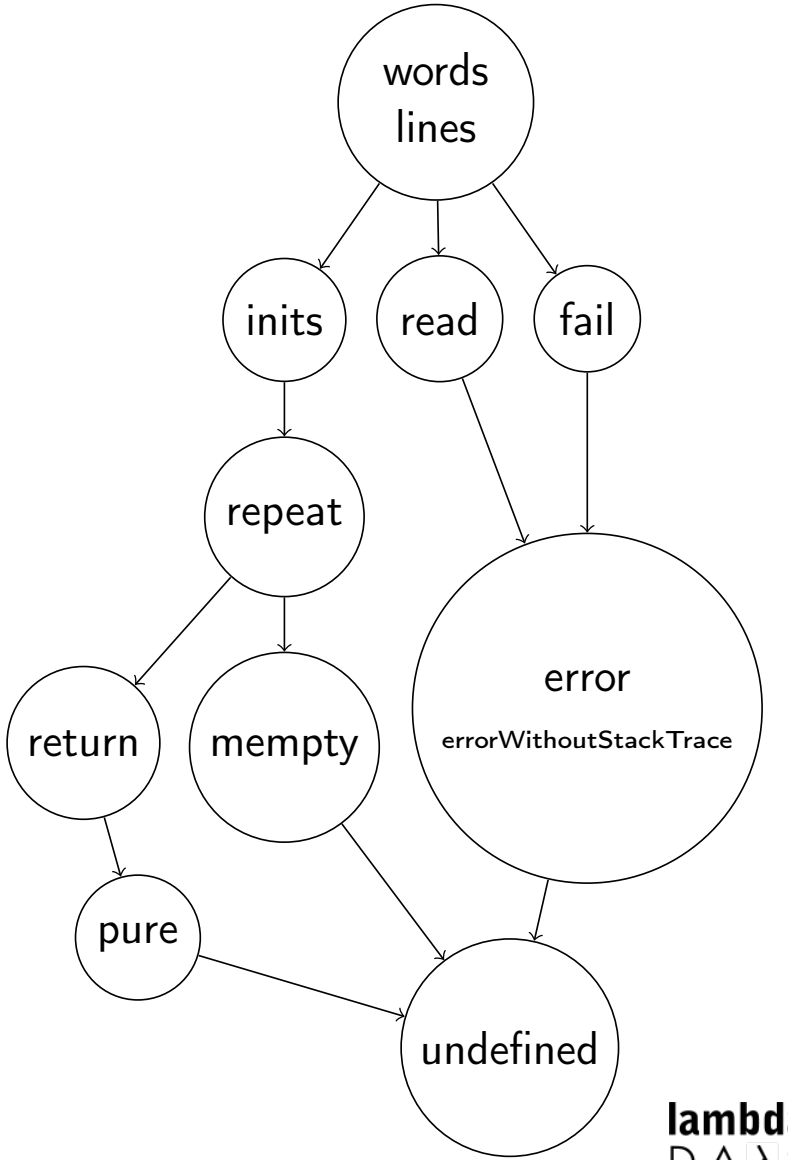
Valid Substitution Demo

DEMO

lambda DAλS

Sorting the Output

- We sort the suggestions by constructing a subsumption graph, and then sorting the suggestions by a topological sort on that graph.
- This makes the most specific suggestions (like `lines` and `words`) appear first, and the more general like `undefined` appear last.



lambda DAYS

Refinement Substitutions

- Often when looking for valid substitutions, the answer isn't a single identifier, but a combination of identifiers.
- An example might be `foldl (+) 0` when writing `sum :: [Integer] -> Integer` or `foldl1 max` for `maximum :: [String] -> String`.

lambda DAYS

Refinement Substitutions

- A refinement substitution is a valid substitution that has one or more holes in it.
- Will be (probably) be available in `GHC` version `8.6.1`
- When searching for e.g. something of type `[Integer] -> Integer`, a refinement substitution might be `foldl1 _` or `foldl _ _`.
- Using refinement substitutions, we can get progressively closer to the definition we want.

lambda DAλS

Refinement Substitutions

- Let's look at the following code:

```
f :: [Integer] -> Integer
f = _ 0
```

- Compiling this with `-frefinement-level-substitutions=1` the compiler will tell us:

```
Valid refinement substitutions include
  foldr _ ::  Foldable t =>
              (a -> b -> b) -> b -> t a -> b
  foldl _ ::  Foldable t =>
              (b -> a -> b) -> b -> t a -> b
```

lambda DAYS

DEMO

How?

- Our extension is based on using the built-in machinery in `GHC` for checking whether one type is a subtype of another.
- By using the already built-in machinery, we can handle large libraries like `lens`, and advance type system features like type families.

DEMO

DEMO

Conclusion

- Valid substitution suggestions for typed holes are useful in many scenarios, for both advanced Haskellers and beginners alike.
- They can help with understanding and learning libraries like the `Prelude` and `lens`.
- They can even help you write secure code!
- Valid substitutions will be available in `8.4.1` (coming soon), while sorting and refinement substitutions are available on `GHC HEAD` and will be released in `8.6.1` later this year.

lambda DAYS

Questions?

Thank you!

lambda DAλS

Details

- To find these substitutions, we first construct the type of the hole, including any constraints.
- Then, for each identifier in scope, we check whether it can fit the hole.
- We do this by emitting a subtype constraint to the constraint solver, to check that the type of the identifier is a subtype of the hole.
- Since the constraint solver works by doing unification via side-effects, we have to take care to clone any type variables involved, so that they don't get effected the checker.
- We then run the constraints checker, and see if the subtype constraint and any relevant constraints get solved.

lambda DAλS

Future Work

- Still buggy! Especially refinement substitutions.
- A search algorithm more akin to what Hoogle does would be nice where they allow some tweaks to the type like changing the number or order of the inputs.
- Consider `f x = (_+x)/5`. Here, `pi :: Floating a => a` is a valid substitution. But since GHC infers that `f` has type of `Fractional a => a -> a`, `pi` gets rejected for not being general enough. Suggesting a tightening (i.e. to a subclass) of inferred constraints when that would give more suggestions would be nice.

Typed Holes Demo Output

Demo1a.hs

```
1  • Found hole: _ :: [Char] -> [String]
2  • In the expression: _
3    In the expression: _ "hello, world"
4    In an equation for ‘f’: f = _ "hello, world"
5  • Relevant bindings include
6      f :: [String] (bound at Demo1.hs:6:1)
```

lambda DAYS

Typed Holes Demo Output

Demo1b.hs

```
1  • Found hole: _a :: (Char -> Bool) -> [Char] -> String
2    Or perhaps ‘_a’ is mis-spelled, or not in scope
3  • In the expression: _a
4    In the expression:
5      _a (_b :: Char -> Bool) "hello, world"
6    In an equation for ‘g’:
7      g = _a (_b :: Char -> Bool) "hello, world"
8  • Relevant bindings include
9      g :: String (bound at TypedHolesDemo/Demo1b.hs:7:1)
10
11 • Found hole: _b :: Char -> Bool
12   Or perhaps ‘_b’ is mis-spelled, or not in scope
13 • In the first argument of ‘_a’,
14     namely ‘(_b :: Char -> Bool)’
15   In the expression:
16     _a (_b :: Char -> Bool) "hello, world"
17   In an equation for ‘g’:
18     g = _a (_b :: Char -> Bool) "hello, world"
19 • Relevant bindings include
20     g :: String (bound at TypedHolesDemo/Demo1b.hs:7:1)
```

lambda DAYS

Typed Holes Demo Output

Demo1c.hs

```
1   • Found hole: _ :: a0
2     Where: ‘a0’ is an ambiguous type variable
3   • In the first argument of ‘show’, namely ‘_’
4     In the expression: show _
5     In an equation for ‘h’: h = show _
6   • Relevant bindings include
7       h :: String (bound at TypedHolesDemo/Demo1c.hs:4:1)
```

lambda DAYS

Valid Substitutions Demo Output

Demo1a.hs

```
1   • Found hole: _ :: [Char] -> [String]
2   • In the expression: _
3     In the expression: _ "hello, world"
4     In an equation for ‘f’: f = _ "hello, world"
5   • Relevant bindings include
6       f :: [String] (bound at Demo1.hs:6:1)
7     Valid substitutions include
8       lines :: String -> [String]
9       words :: String -> [String]
10      group :: forall a. Eq a => [a] -> [[a]]
11      inits :: forall a. [a] -> [[a]]
12      permutations :: forall a. [a] -> [[a]]
13      subsequences :: forall a. [a] -> [[a]]
14      (Some substitutions suppressed;
15        use -fmax-valid-substitutions=N
16        or -fno-max-valid-substitutions)
```

lambda DAYS

Valid Substitutions Demo Output

Demo1b.hs: First Hole

```
1   • Found hole: _a :: (Char -> Bool) -> [Char] -> String
2     Or perhaps ‘_a’ is mis-spelled, or not in scope
3   • In the expression: _a
4     In the expression:
5       _a (_b :: Char -> Bool) "hello, world"
6     In an equation for ‘g’:
7       g = _a (_b :: Char -> Bool) "hello, world"
8   • Relevant bindings include
9       g :: String (bound at TypedHolesDemo/Demo1b.hs:7:1)
10    Valid substitutions include
11      filter :: forall a. (a -> Bool) -> [a] -> [a]
12      dropWhile :: forall a. (a -> Bool) -> [a] -> [a]
13      takeWhile :: forall a. (a -> Bool) -> [a] -> [a]
14      dropWhileEnd :: forall a. (a -> Bool) -> [a] -> [a]
15      sortOn :: forall b a. Ord b => (a -> b) -> [a] -> [a]
16      mempty :: forall a. Monoid a => a
17      (Some substitutions suppressed;
18          use -fmax-valid-substitutions=N
19          or -fno-max-valid-substitutions)
```

lambda DAYS

Valid Substitutions Demo Output

Demo1b.hs: Second Hole

```
• Found hole: _b :: Char -> Bool
  Or perhaps ‘_b’ is mis-spelled, or not in scope
• In the first argument of ‘_a’,
    namely ‘(_b :: Char -> Bool)’
  In the expression:
    _a (_b :: Char -> Bool) "hello, world"
  In an equation for ‘g’:
    g = _a (_b :: Char -> Bool) "hello, world"
• Relevant bindings include
    g :: String (bound at TypedHolesDemo/Demo1b.hs:7:1)
  Valid substitutions include
    isLetter :: Char -> Bool
    isMark :: Char -> Bool
    isNumber :: Char -> Bool
    isSeparator :: Char -> Bool
    isAlpha :: Char -> Bool
    isAlphaNum :: Char -> Bool
    (Some substitutions suppressed;
        use -fmax-valid-substitutions=N
        or -fno-max-valid-substitutions)
```

lambda DAYS

Valid Substitutions Demo Output

Demo1c.hs

```
1   • Found hole: _ :: a0
2     Where: ‘a0’ is an ambiguous type variable
3   • In the first argument of ‘show’, namely ‘_’
4     In the expression: show _
5     In an equation for ‘h’: h = show _
6   • Relevant bindings include
7       h :: String (bound at TypedHolesDemo/Demo1c.hs:4:1)
8     Valid substitutions include
9       h :: String
10      EQ :: Ordering
11      LT :: Ordering
12      GT :: Ordering
13      pi :: forall a. Floating a => a
14      otherwise :: Bool
15      (Some substitutions suppressed;
16          use -fmax-valid-substitutions=N
17          or -fno-max-valid-substitutions)
```

lambda DAYS

Lens Demo Output

LensDemo/sr/test.hs

```
1   • Found hole:
2       _a :: ((Integer -> f0 Integer) -> Test -> f0 Test)
3             -> (Integer -> Integer) -> State Test a0
4     Where: ‘f0’ is an ambiguous type variable
5            ‘a0’ is an ambiguous type variable
6   • In the expression: ...
7   • Relevant bindings include ...
8     Valid substitutions include
9       (%=) :: forall s (m :: * -> *) a b.
10                MonadState s m => ASetter s s a b -> (a -> b) -> m ()
11      modifying :: forall s (m :: * -> *) a b.
12                   MonadState s m => ASetter s s a b -> (a -> b) -> m ()
13      (<%=) :: forall s (m :: * -> *) b a.
14               MonadState s m => LensLike ((,) b) s s a b -> (a -> b) -> m b
15      (<#%=) :: forall s (m :: * -> *) a b.
16                MonadState s m => ALens s s a b -> (a -> b) -> m b
17      (#%=) :: forall s (m :: * -> *) a b.
18               MonadState s m => ALens s s a b -> (a -> b) -> m ()
19      uses :: forall s (m :: * -> *) r a.
20              MonadState s m => LensLike' (Const r) s a -> (a -> r) -> m r
21      (Some substitutions suppressed;
22          use -fmax-valid-substitutions=N
23          or -fno-max-valid-substitutions)
```

lambda DAYS

Lens Demo Output

LensDemo/src/test.hs

```
• Found hole:
    _b :: ((Integer -> f0 Integer) -> Test -> f1 Test)
          -> Integer -> State Test a1
  Where: ‘f1’ is an ambiguous type variable
         ‘a1’ is an ambiguous type variable
• In the expression: ...
• Relevant bindings include ...
  Valid substitutions include
    (^=) :: forall s (m :: * -> *) a e.
            (MonadState s m, Num a, Integral e) => ASetter' s a -> e -> m ()
    (<.=) :: forall s (m :: * -> *) a b.
             MonadState s m => ASetter s s a b -> b -> m b
    (*=) :: forall s (m :: * -> *) a.
            (MonadState s m, Num a) => ASetter' s a -> a -> m ()
    (+=) :: forall s (m :: * -> *) a.
            (MonadState s m, Num a) => ASetter' s a -> a -> m ()
    (-=) :: forall s (m :: * -> *) a.
            (MonadState s m, Num a) => ASetter' s a -> a -> m ()
    (.=) :: forall s (m :: * -> *) a b.
            MonadState s m => ASetter s s a b -> b -> m ()
    (Some substitutions suppressed;
        use -fmax-valid-substitutions=N
        or -fno-max-valid-substitutions)
```

lambda DAYS

Constraints Demo Output

DCC/Example.hs

```
1   • Found hole: _ :: T 'H User -> T 'L a0
2     Where: ‘a0’ is an ambiguous type variable
3   • In the expression: _
4     In the first argument of ‘pure’, namely ‘(_ user)’
5     In a stmt of a 'do' block: info <- pure (_ user)
6   • Relevant bindings include
7       user :: T 'H User (bound at Example.hs:4:11)
8       main :: IO () (bound at Example.hs:4:1)
9     Valid substitutions include
10      isInGothenburg :: T 'H User -> T 'L Bool
11      isAllowedToDrink :: T 'H User -> T 'L Bool
12      bestNearbyRestaurant :: T 'H User -> T 'L (Maybe Restaurant)
```

lambda DAYS

Sorting Demo Output

Demo4a.hs

```
1   • Found hole: _ :: [Char] -> [String]
2   • In the expression: _
3     In the expression: _ "hello, world"
4     In an equation for ‘f’: f = _ "hello, world"
5   • Relevant bindings include
6       f :: [String] (bound at TypedHolesDemo/Demo4a.hs:7:1)
7     Valid substitutions include
8       inits :: forall a. [a] -> [[a]]
9       fail :: forall (m :: * -> *). Monad m => forall a. String -> m a
10      mempty :: forall a. Monoid a => a
11      pure :: forall (f :: * -> *). Applicative f => forall a. a -> f a
12      return :: forall (m :: * -> *). Monad m => forall a. a -> m a
13      read :: forall a. Read a => String -> a
14      lines :: String -> [String]
15      words :: String -> [String]
16      error :: forall (a :: TYPE r). HasCallStack => [Char] -> a
17      errorWithoutStackTrace :: forall (a :: TYPE r). [Char] -> a
18      undefined :: forall (a :: TYPE r). HasCallStack => a
19      repeat :: forall a. a -> [a]
```

lambda DAYS

Sorting Demo Output

Demo4b.hs

```
1   • Found hole: _ :: [Char] -> [String]
2   • In the expression: _
3     In the expression: _ "hello, world"
4     In an equation for ‘f’: f = _ "hello, world"
5   • Relevant bindings include
6       f :: [String] (bound at TypedHolesDemo/Demo4b.hs:7:1)
7     Valid substitutions include
8       lines :: String -> [String]
9       words :: String -> [String]
10      inits :: forall a. [a] -> [[a]]
11      read :: forall a. Read a => String -> a
12      repeat :: forall a. a -> [a]
13      mempty :: forall a. Monoid a => a
14      return :: forall (m :: * -> *). Monad m => forall a. a -> m a
15      pure :: forall (f :: * -> *). Applicative f => forall a. a -> f a
16      fail :: forall (m :: * -> *). Monad m => forall a. String -> m a
17      error :: forall (a :: TYPE r). HasCallStack => [Char] -> a
18      errorWithoutStackTrace :: forall (a :: TYPE r). [Char] -> a
19      undefined :: forall (a :: TYPE r). HasCallStack => a
```

lambda DAYS

Refinement Demo Output

Demo0a.hs

```
1   • Found hole: _ :: Integer -> [Integer] -> Integer
2   • In the expression: _
3     In the expression: _ 0
4     In an equation for ‘f’: f = _ 0
5   • Relevant bindings include
6       f :: [Integer] -> Integer (bound at TypedHolesDemo/Demo0a.hs:4:1)
7     Valid substitutions include
8       const :: forall a b. a -> b -> a
9       undefined :: forall (a :: TYPE r). HasCallStack => a
10    Valid refinement substitutions include
11      foldr _ :: forall (t :: * -> *).
12                 Foldable t =>
13                 forall a b. (a -> b -> b) -> b -> t a -> b
14      foldl _ :: forall (t :: * -> *).
15                 Foldable t =>
16                 forall b a. (b -> a -> b) -> b -> t a -> b
17      head _ :: forall a. [a] -> a
18      last _ :: forall a. [a] -> a
19      error _ :: forall (a :: TYPE r). HasCallStack => [Char] -> a
20      errorWithoutStackTrace _ :: forall (a :: TYPE r). [Char] -> a
21      (Some refinement substitutions suppressed;
22          use -fmax-refinement-substitutions=N
23          or -fno-max-refinement-substitutions)
```

lambda DAYS

Refinement Demo Output

Demo0b.hs

```
1   • Found hole: _ :: [Integer] -> Integer
2   • In the expression: _
3     In an equation for ‘f’: f = _
4   • Relevant bindings include
5       f :: [Integer] -> Integer (bound at TypedHolesDemo/Demo0b.hs:4:1)
6     Valid substitutions include
7       f :: [Integer] -> Integer
8       product :: forall (t :: * -> *). Foldable t =>
9                  forall a. Num a => t a -> a
10      sum :: forall (t :: * -> *). Foldable t =>
11             forall a. Num a => t a -> a
12      maximum :: forall (t :: * -> *). Foldable t =>
13                 forall a. Ord a => t a -> a
14      minimum :: forall (t :: * -> *). Foldable t =>
15                 forall a. Ord a => t a -> a
16      head :: forall a. [a] -> a
17      (Some substitutions suppressed;
18        use -fmax-valid-substitutions=N or -fno-max-valid-substitutions)
19    Valid refinement substitutions include
20      foldr _ _ :: forall (t :: * -> *). Foldable t =>
21                   forall a b. (a -> b -> b) -> b -> t a -> b
22      foldl1 _ :: forall (t :: * -> *). Foldable t =>
23                  forall a. (a -> a -> a) -> t a -> a
24      foldr1 _ :: forall (t :: * -> *). Foldable t =>
25                  forall a. (a -> a -> a) -> t a -> a
26      foldl _ _ :: forall (t :: * -> *). Foldable t =>
27                   forall b a. (b -> a -> b) -> b -> t a -> b
28      head _ :: forall a. [a] -> a
29      last _ :: forall a. [a] -> a
30      (Some refinement substitutions suppressed;
31          use -fmax-refinement-substitutions=N or -fno-max-refinement-substitutions)
```

lambda DAYS

Refinement Demo Output

Demo0c.hs

```
TypedHolesDemo/Demo0c.hs:6:5: error:
    • Found hole: _ :: String -> [String]
    • In the expression: _
      In an equation for ‘f’: f = _
    • Relevant bindings include
        f :: String -> [String] (bound at TypedHolesDemo/Demo0c.hs:6:1)
      Valid substitutions include
        f :: String -> [String]
        lines :: String -> [String]
        words :: String -> [String]
        group :: forall a. Eq a => [a] -> [[a]]
        inits :: forall a. [a] -> [[a]]
        permutations :: forall a. [a] -> [[a]]
        (Some substitutions suppressed; use -fmax-valid-substitutions=N or -fno-max-valid-substitutions)
      Valid refinement substitutions include
        foldl1' _ _ :: forall a. (a -> a -> a) -> [a] -> a
        unfoldr _ :: forall b a. (b -> Maybe (a, b)) -> b -> [a]
        groupBy _ :: forall a. (a -> a -> Bool) -> [a] -> [[a]]
        (<$) _ :: forall (f :: * -> *).
                  Functor f =>
                  forall a b. a -> f b -> f a
        (<*) _ :: forall (f :: * -> *).
                  Applicative f =>
                  forall a b. f a -> f b -> f a
        mapM _ :: forall (t :: * -> *).
                  Traversable t =>
                  forall (m :: * -> *) a b. Monad m => (a -> m b) -> t a -> m (t b)
        (Some refinement substitutions suppressed;
          use -fmax-refinement-substitutions=N
          or -fno-max-refinement-substitutions)
```

lambda DAYS