Signature Inference for Functional Property Discovery

or: How never to come up with tests manually anymore(*)

Tom Sydney Kerckhove

ETH Zurich
https://cs-syd.eu/
https://github.com/NorfairKing

11 October 2017

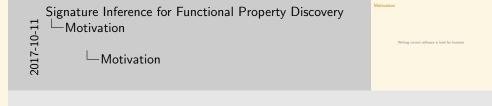
Signature Inference for Functional Property Discovery

2017-10-11

Signature Inference for Functional Property
Discovery
or: How never to come up with tests manually anymore(*)
Tom Sydney Kerduhov
ETH Zalah
Nerge (Franch only March State)
State (Franch only March State)
State (Franch only March State)

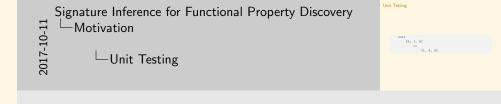
Motivation

Writing correct software is hard for humans.

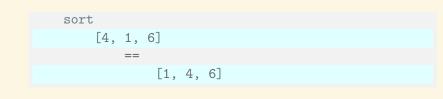


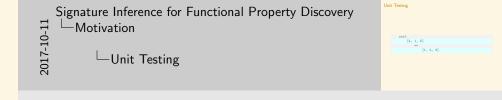
Unit Testing

```
sort
[4, 1, 6]
==
[1, 4, 6]
```



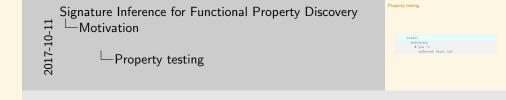
Unit Testing





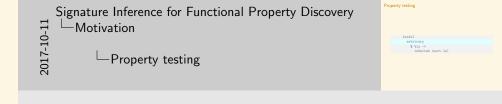
Property testing

```
forAll
arbitrary
$ \ls ->
isSorted (sort ls)
```



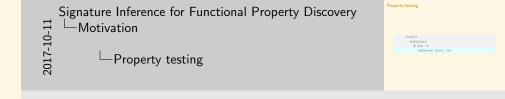
Property testing

```
forAll
arbitrary
$ \ls ->
isSorted (sort ls)
```



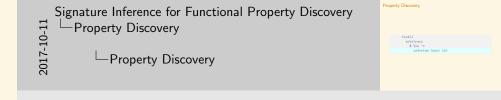
Property testing

```
forAll
arbitrary
$ \ls ->
isSorted (sort ls)
```



Property Discovery

```
forAll
arbitrary
$ \ls ->
isSorted (sort ls)
```



Example Code

```
module MySort where
```

```
mySort :: Ord a => [a] -> [a]
mySort [] = []
mySort (x:xs) = insert (mySort xs)
  where
    insert [] = [x]
    insert (y:ys)
         | x \le y = x : y : ys
         otherwise = y : insert ys
myIsSorted :: Ord a => [a] -> Bool
myIsSorted [] = True
myIsSorted [_] = True
myIsSorted (x:y:ls) = x <= y && myIsSorted (y : ls)</pre>
```

Signature Inference for Functional Property Discovery Property Discovery Example Code Example Code Example Code

myIsSorted [] - True myIsSorted [_] - True

mvIsSorted (x:v:ls) = x <= v && mvIsSorted (v : ls)

Example Code

module MySort where

```
mySort :: Ord a => [a] -> [a]
mySort [] = []
mySort (x:xs) = insert (mySort xs)
  where
    insert [] = [x]
    insert (y:ys)
        | x \le y = x : y : ys
         otherwise = y : insert ys
myIsSorted :: Ord a => [a] -> Bool
myIsSorted [] = True
myIsSorted [_] = True
myIsSorted (x:y:ls) = x <= y && myIsSorted (y : ls)</pre>
```

```
Signature Inference for Functional Property Discovery

Property Discovery

Example Code

Signature Inference for Functional Property Discovery

Functional Property Discovery
```

myIsSorted [_] - True

mvIsSorted (x:v:ls) = x <= v && mvIsSorted (v : ls)

Property Discovery using QuickSpec

```
== Signature ==
    True :: Bool
    (<=) :: Ord a => a -> a -> Bool
    (:) :: a -> [a] -> [a]
    mySort :: Ord a => [a] -> [a]
myIsSorted :: Ord a => [a] -> Bool
```





Property Discovery using QuickSpec

Property Discovery using QuickSpec

```
== Signature ==
    True :: Bool
    (<=) :: Ord a => a -> a -> Bool
    (:) :: a -> [a] -> [a]
    mySort :: Ord a => [a] -> [a]
myIsSorted :: Ord a => [a] -> Bool

== Laws ==
```

```
1. y <= y = True
2. y <= True = True
3. True <= x = x
4. myIsSorted (mySort xs) = True
5. mySort (mySort xs) = mySort xs
6. xs <= mySort xs = myIsSorted xs
7. mySort xs <= xs = True
8. myIsSorted (y : (y : xs)) = myIsSorted (y : xs)
9. mySort (y : mySort xs) = mySort (y : xs)
```


Property Discovery using QuickSpec

```
Property Discovery using QuickSpec

** Eignature **

(**) If the ** on ** on ** on build

(**) If the ** of on build

(**) If the ** of on build

(**) If the ** of on build

** Lines **

I. y or y = True

2. The ** of on build

(**) If the ** of on build

** Lines **

4. synthesized figure un * True

4. synthesized figure un ** of on build

6. so ** of officers un ** optionized as
```

Property Discovery using QuickSpec

```
== Signature ==
     True :: Bool
      (<=) :: Ord a => a -> a -> Bool
      (:) :: a -> [a] -> [a]
    mySort :: Ord a => [a] -> [a]
myIsSorted :: Ord a => [a] -> Bool
== Laws ==
 1. y <= y = True
 2. y <= True = True
 3. True \leq x = x
  4. myIsSorted (mySort xs) = True
  5. mySort (mySort xs) = mySort xs
  6. xs <= mySort xs = myIsSorted xs
  7. mySort xs <= xs = True
  8. myIsSorted (y : (y : xs)) = myIsSorted (y : xs)
  9. mySort (y : mySort xs) = mySort (y : xs)
```

Signature Inference for Functional Property Discovery — Property Discovery

Property Discovery using QuickSpec

QuickSpec Code

```
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE ConstraintKinds #-}
{-# LANGUAGE RankNTypes #-}
{-# LANGUAGE FlexibleContexts #-}
module MySortQuickSpec where
import Control.Monad
import MySort
import QuickSpec
main :: IO ()
main =
    void $
    quickSpec
        signature
        { constants =
              [ constant "True" (True :: Bool)
              , constant "<=" (mkDict (<=) :: Dict (Ord A) -> A -> A -> Bool)
              , constant ":" ((:) :: A -> [A] -> [A])
              , constant "mySort" (mkDict mySort :: Dict (Ord A) -> [A] -> [A])
              , constant
                    "myIsSorted"
                    (mkDict myIsSorted :: Dict (Ord A) -> [A] -> Bool)
mkDict ::
       (c =>
    -> Dict c
    -> a
mkDict x Dict = x
```

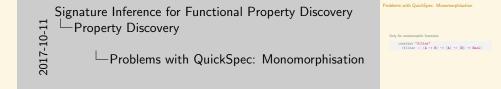
-QuickSpec Code



Problems with QuickSpec: Monomorphisation

Only for monomorphic functions

```
constant "filter"
  (filter :: (A -> B) -> [A] -> [B] -> Bool)
```



Problems with QuickSpec: Code

Programmer has to write code for all functions of interest 15 lines of subject code.

33 lines of QuickSpec code.

Signature Inference for Functional Property Discovery

—Property Discovery

—Problems with QuickSpec: Code

Property Discovery

—Problems with QuickSpec: Code

Problems with QuickSpec: Speed

Dumb version of the QuickSpec approach:

- 1. Generate all possible terms
- 2. Generate all possible equations (tuples) of terms
- 3. Type check them to make sure the equation makes sense
- 4. Check that the input can be generated and the output compared for equality
- 5. Run QuickCheck to see if the equation holds

Signature Inference for Functional Property Discovery Property Discovery

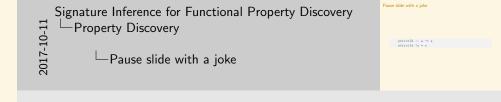
Problems with QuickSpec: Speed

Problems with QuickSpec: Speed

- Dumb version of the QuickSpec approach:
- 3. Type check them to make sure the equation makes sense
- 5. Run QuickCheck to see if the equation holds

Pause slide with a joke

```
strictId :: a -> a
strictId !x = x
```



Signature Inference for Functional Property Discovery

-Property Discovery

Property Discovery with EasySpec

Step 1: Automation

Signatures

```
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE ConstraintKinds #-}
{-# LANGUAGE RankNTypes #-}
{-# LANGUAGE FlexibleContexts #-}
module MySortQuickSpec where
import Control.Monad
import MySort
import QuickSpec
main :: IO ()
main =
    void $
    quickSpec
        signature
        { constants =
              [ constant "True" (True :: Bool)
              , constant "<=" (mkDict (<=) :: Dict (Ord A) -> A -> A -> Bool)
              , constant ":" ((:) :: A -> [A] -> [A])
              , constant "mySort" (mkDict mySort :: Dict (Ord A) -> [A] -> [A])
              , constant
                    "myIsSorted"
                    (mkDict myIsSorted :: Dict (Ord A) -> [A] -> Bool)
mkDict ::
       (c =>
    -> Dict c
    -> a
mkDict x Dict = x
```


Signatures

```
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE ConstraintKinds #-}
{-# LANGUAGE RankNTypes #-}
{-# LANGUAGE FlexibleContexts #-}
module MySortQuickSpec where
import Control.Monad
import MySort
import QuickSpec
main :: IO ()
main =
    void $
    quickSpec
        signature
        { constants =
               [ constant "True" (True :: Bool)
              , constant "<=" (mkDict (<=) :: Dict (Ord A) -> A -> A -> Bool)
              , constant ":" ((:) :: A -> [A] -> [A])
              , constant "mySort" (mkDict mySort :: Dict (Ord A) -> [A] -> [A])
              , constant
                    "myIsSorted"
                    (mkDict myIsSorted :: Dict (Ord A) -> [A] -> Bool)
mkDict ::
       (c =>
    -> Dict c
    -> a
mkDict x Dict = x
```

Signature Inference for Functional Property Discovery Automation Signature Signature Automation

A QuickSpec Signature

```
data Signature =
   Signature {
    functions :: [Function],
    [...]
    properties :: [Prop],
  }

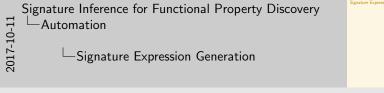
quickSpec :: Signature -> IO Signature
```

Signature Inference for Functional Property Discovery

Automation

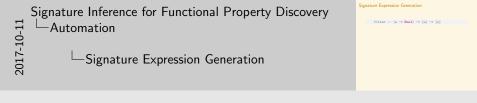
Automation

A QuickSpec Signature



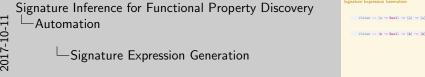
Signature Expression Generation

```
filter :: (a -> Bool) -> [a] -> [a]
```



```
filter :: (a -> Bool) -> [a] -> [a]
```

filter :: (A -> Bool) -> [A] -> [A]

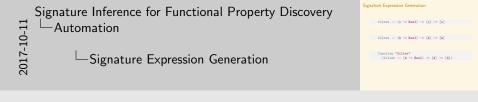




```
filter :: (a -> Bool) -> [a] -> [a]
```

filter :: (A -> Bool) -> [A] -> [A]

function "filter"
 (filter :: (A -> Bool) -> [A] -> [A])



```
filter :: (a -> Bool) -> [a] -> [a]
filter :: (A -> Bool) -> [A] -> [A]
function "filter"
  (filter :: (A -> Bool) -> [A] -> [A])
signature { constants = [...] }
```

```
Signature Inference for Functional Property Discovery

Automation

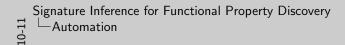
Signature Expression Generation

Signature Expression Generation

Signature Expression Generation
```

Current situation

```
$ cat Reverse.hs
{-# LANGUAGE NoImplicitPrelude #-}
module Reverse where
import Data.List (reverse, sort)
```



Current situation

\$ cat Reverse.hs f-# LANGUAGE NoInmilicitPrelude #-}

(-# LANGUAGE NoImplicitPrelude & module Reverse where import Data.List (reverse, sort)

Current situation

```
$ cat Reverse.hs
{-# LANGUAGE NoImplicitPrelude #-}
module Reverse where
import Data.List (reverse, sort)
$ easyspec discover Reverse.hs
    reverse (reverse xs) = xs
    sort (reverse xs) = sort xs
```

Signature Inference for Functional Property Discovery $\stackrel{\textstyle \sqcup}{}$ Automation

Current situation

are areaucion

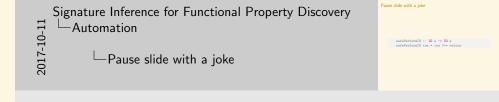
\$ cat Reverse.hs {-# LANGUAGE NoImplicitPrelude 8-} module Reverse where

import Data.List (reverse, sort)

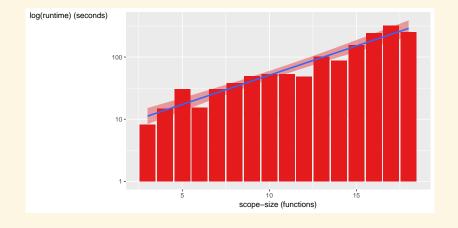
\$ easyspec discover Reverse.hs
reverse (reverse xs) = xs
sort (reverse xs) = sort xs

Pause slide with a joke

```
safePerformIO :: IO a -> IO a
safePerformIO ioa = ioa >>= return
```



Automated, but still slow







Automated, but still slow

—Automated, but still slow

1. Now we have automated QuickSpec, but it still slow

Definitions

Definitions: Property

Example:

```
reverse (reverse ls) = ls
```

Short for:

```
(\label{ls} -> reverse (reverse ls)) = (\ls -> ls)
```

In general:

```
(f :: A -> B) = (g :: A -> B)
for some A and B with
instance Arbitrary A
instance Eq B
```

Signature Inference for Functional Property Discovery

Signature Inference

Definitions: Property

Signature Inference

Compare (respect 10) = (Na = 10)

Definitions: Property

instance Eq B

Why is this slow?

1. Maximum size of the discovered properties

U-Signature Inference
Why is this slow?

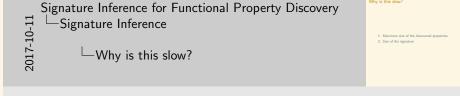
Signature Inference for Functional Property Discovery

1. Maximum size of the discovered properties

Why is this slow?

Why is this slow?

- 1. Maximum size of the discovered properties
- 2. Size of the signature



Why is this slow?

Idea



Signature Inference for Functional Property Discovery

Signature Inference

It-01/2

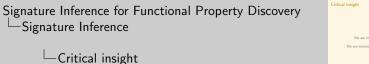
Idea



Critical insight

We are not interested in the entire codebase.

We are interested in a relatively small amount of code.

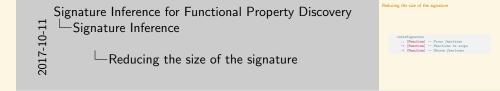




- 1. This means that we have an entirely different goal than QuickSpec.
- 2. Comparisons with QuickSpec are not really fair, but we have nothing else to compare to.

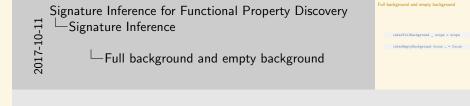
Reducing the size of the signature

```
inferSignature
    :: [Function] -- Focus functions
    -> [Function] -- Functions in scope
    -> [Function] -- Chosen functions
```



Full background and empty background

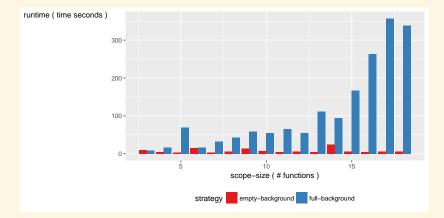
```
inferFullBackground _ scope = scope
inferEmptyBackground focus _ = focus
```



Full background and empty background

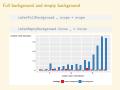
inferFullBackground _ scope = scope

inferEmptyBackground focus _ = focus



Signature Inference for Functional Property Discovery \square Signature Inference

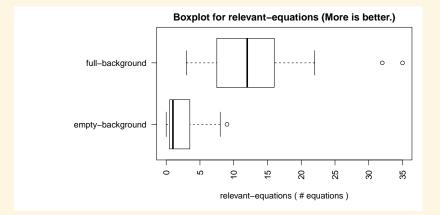
Full background and empty background



Full background and empty background

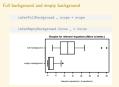
inferFullBackground _ scope = scope

inferEmptyBackground focus _ = focus

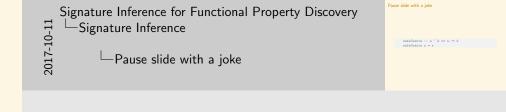


Signature Inference for Functional Property Discovery \square Signature Inference

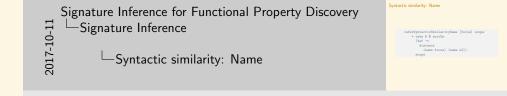
Full background and empty background



Pause slide with a joke

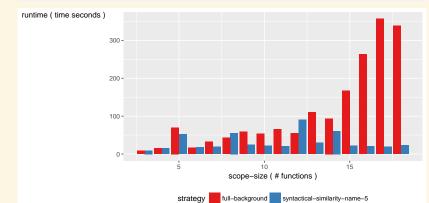


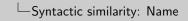
Syntactic similarity: Name

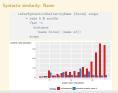


Syntactic similarity: Name

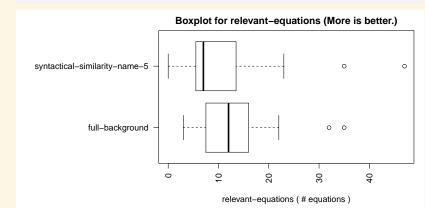
```
inferSyntacticSimilarityName [focus] scope
= take 5 $ sortOn
    (\sf ->
          distance
          (name focus) (name sf))
    scope
```

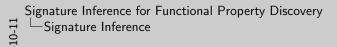




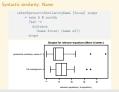


Syntactic similarity: Name





-Syntactic similarity: Name



Syntactic similarity: Implementation

Signature Inference for Functional Property Discovery

Signature Inference

Signature Inference

Syntactic similarity: Implementation

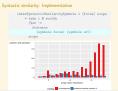
Syntactic similarity: Implementation

Syntactic similarity: Implementation

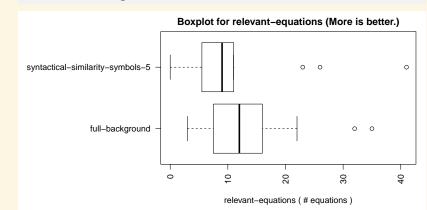
```
inferSyntacticSimilaritySymbols i [focus] scope
           = take i $ sortOn
              (\sf ->
                distance
                    (symbols focus) (symbols sf))
              scope
runtime (time seconds)
                300 -
                200 -
                100 -
                                                           15
                                      scope-size (# functions)
                           strategy full-background syntactical-similarity-symbols-5
```

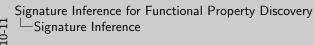
Signature Inference for Functional Property Discovery \square Signature Inference

Syntactic similarity: Implementation

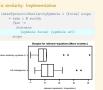


Syntactic similarity: Implementation

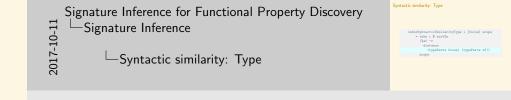








Syntactic similarity: Type



Syntactic similarity: Type

```
inferSyntacticSimilarityType i [focus] scope
           = take i $ sortOn
              (\sf ->
                distance
                   (typeParts focus) (typeParts sf))
              scope
runtime (time seconds)
               300 -
               200 -
               100 -
                                                           15
                                     scope-size (# functions)
                            strategy full-background syntactical-similarity-type-5
```

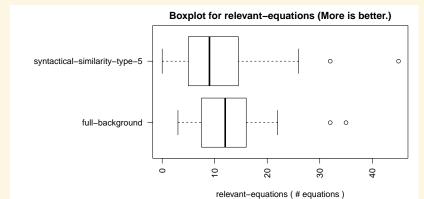
Signature Inference for Functional Property Discovery

Signature Inference

Syntactic similarity: Type



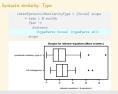
Syntactic similarity: Type



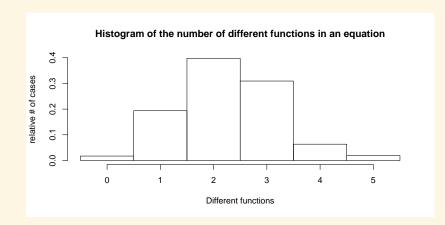
Signature Inference for Functional Property Discovery

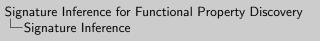
Signature Inference

-Syntactic similarity: Type

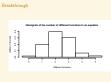


Breakthrough









Idea



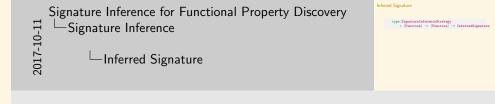
Signature Inference for Functional Property Discovery
Signature Inference
Idea



Signature Inference for Functional Property Discovery

-Signature Inference

We can run QuickSpec more than once!



```
type SignatureInferenceStrategy
= [Function] -> [Function] -> InferredSignature
```

Combine the results of multiple runs:

type InferredSignature = [Signature]

Signature Inference for Functional Property Discovery

Signature Inference

Signature Inference

Inferred Signature

Inferred Signature

Combine the results of multiple runs:

```
type InferredSignature = [Signature]
```

User previous results as background properties:

type InferredSignature = Forest Signature

Signature Inference for Functional Property Discovery

Signature Inference

☐ Inferred Signature

type SignaturaInforenceStrategy

- [Function] >> [Function] >> InforredSignature
bine the results of multiple rans:

type InforredSignature = [Signature]

provided in the results of background properties:

type InforredSignature = Forenties:

type InforredSignature = Forenties

type InforredSignature = Forenties

Inferred Signature

```
type SignatureInferenceStrategy
    = [Function] -> [Function] -> InferredSignature
```

Combine the results of multiple runs:

```
type InferredSignature = [Signature]
```

User previous results as background properties:

```
type InferredSignature = Forest Signature
```

Share previous runs:

type InferredSignature = DAG Signature

Signature Inference for Functional Property Discovery Signature Inference

☐ Inferred Signature

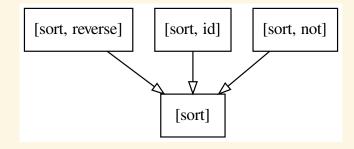
Inferred Signature - [Function] -> [Function] -> InferredSignature type InferredSignature - [Signature] type InferredSignature - Forest Signature

type InferredSignature - DAG Signature

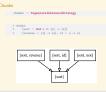
Chunks

chunks :: SignatureInferenceStrategy

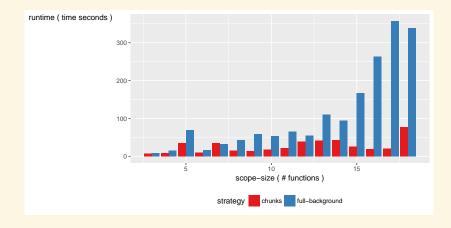
```
> chunks
>     [sort :: Ord a => [a] -> [a]]
>     [reverse :: [a] -> [a], id :: a -> a]
```



└**Chunks**

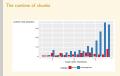


The runtime of chunks



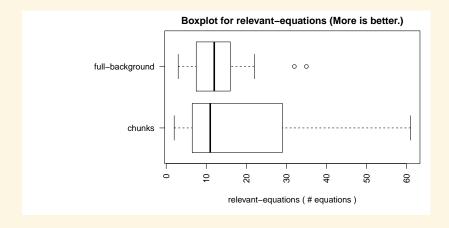
Signature Inference for Functional Property Discovery

—Signature Inference



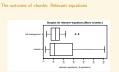
☐The runtime of chunks

The outcome of chunks: Relevant equations



Signature Inference for Functional Property Discovery \square Signature Inference

The outcome of chunks: Relevant equations



```
Signature Inference for Functional Property Discovery

Signature Inference

Signature Inference

Inferred Signature

Inferred Signature

Signat
```

```
type SignatureInferenceStrategy
    = [Function] -> [Function] -> InferM ()
data InferM a where
    InferPure :: a -> InferM a
    InferFmap :: (a -> b) -> InferM a -> InferM b
    InferApp :: InferM (a -> b) -> InferM a -> InferM b
    InferBind :: InferM a -> (a -> InferM b) -> InferM b
    InferFrom
        :: Signature
        -> [OptiToken]
        -> InferM (OptiToken, [Equation])
```

```
Signature Inference for Functional Property Discovery

Signature Inference

Signature Inference

Inference

Inference

Inference

Inference Signature

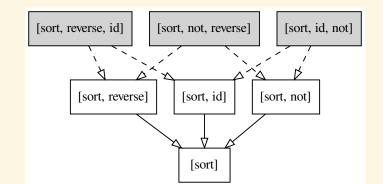
Inference Signature

Inference Signature
```

Chunks Plus

chunksPlus :: SignatureInferenceStrategy

```
> chunksPlus
> [sort :: Ord a => [a] -> [a]]
> [reverse :: [a] -> [a], id :: a -> a]
```



Signature Inference for Functional Property Discovery

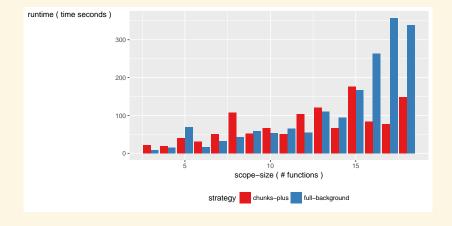
Signature Inference

Charles Plus

Charles Plus

Charles Plus

The runtime of chunks plus



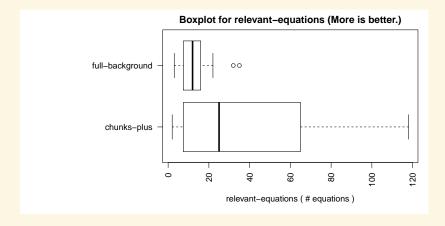
Signature Inference for Functional Property Discovery

Signature Inference

The runtime of chunks plus

☐ The runtime of chunks plus

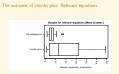
The outcome of chunks plus: Relevant equations



Signature Inference for Functional Property Discovery

—Signature Inference





Neat

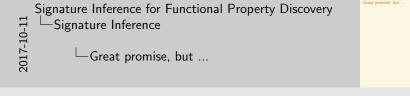
Signature Inference for Functional Property Discovery

Signature Inference

Signature Inference

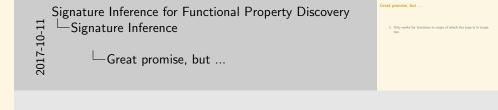
Neat

Neat

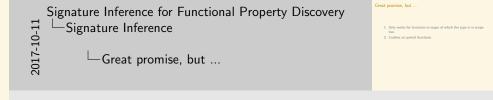


Great promise, but ...

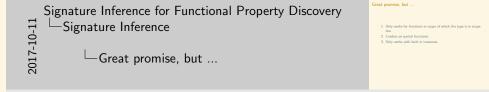
1. Only works for functions in scope of which the type is in scope too.



- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.



- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.
- 3. Only works with built in instances.



- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.
- 3. Only works with built in instances.
- 4. Data has to have an Arbitrary instance in scope.

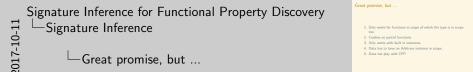
Signature Inference for Functional Property Discovery

Signature Inference

Cody works for function is support of which the type is in scope of which the type as in scope of which the type as in scope of which the type is in scope.

Great promise, but ...

- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.
- 3. Only works with built in instances.
- 4. Data has to have an Arbitrary instance in scope.
- 5. Does not play with CPP.



- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.
- 3. Only works with built in instances.
- 4. Data has to have an Arbitrary instance in scope.
- 5. Does not play with CPP.
- 6. Does not play well with higher kinded type variables.

Signature Inference for Functional Property Discovery

Signature Inference

Creat promise, but ...

1. Only works for functions in stope of which the type is in scope of which the type as in scope of which the transformer.

2. Content on print functions.

3. Only works and hash in instruction.

4. Data has to have an Anthrop states in scope.

5. Data on all part and CPP.

6. Data on all part and CPP.

- 1. Only works for functions in scope of which the type is in scope too.
- 2. Crashes on partial functions.
- 3. Only works with built in instances.
- 4. Data has to have an Arbitrary instance in scope.
- 5. Does not play with CPP.
- 6. Does not play well with higher kinded type variables.

All technical problems, not theoretical problems!

Signature Inference for Functional Property Discovery Signature Inference 1. Only work for functions is some of which the type is in scape of some of the control of the type is in scape of some of the control of th

1. Can we go faster?

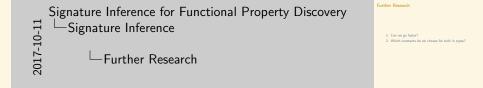
Further Research

-Signature Inference

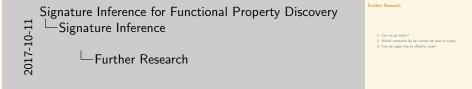
Signature Inference for Functional Property Discovery

1. Can we go faster?

- 1. Can we go faster?
- 2. Which constants do we choose for built in types?



- 1. Can we go faster?
- 2. Which constants do we choose for built in types?
- 3. Can we apply this to effectful code?



- 1. Can we go faster?
- 2. Which constants do we choose for built in types?
- 3. Can we apply this to effectful code?
- 4. Relative importance of equations

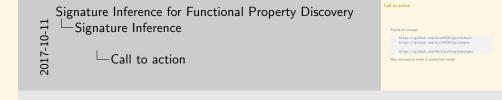


Call to action

Proofs of concept:

```
https://github.com/nick8325/quickcheck
https://github.com/nick8325/quickspec
https://github.com/NorfairKing/easyspec
```

Now we need to make it production ready!



About Me

This was my master thesis Wrote Haskell in open source Taught Haskell at ETH Haskell and DevOps in industry

https://cs-syd.eu/
https://cs-syd.eu/cv
https://github.com/NorfairKing

Signature Inference for Functional Property Discovery

Signature Inference

This was my maker theat Winds Marked in community Winds and Community Wind