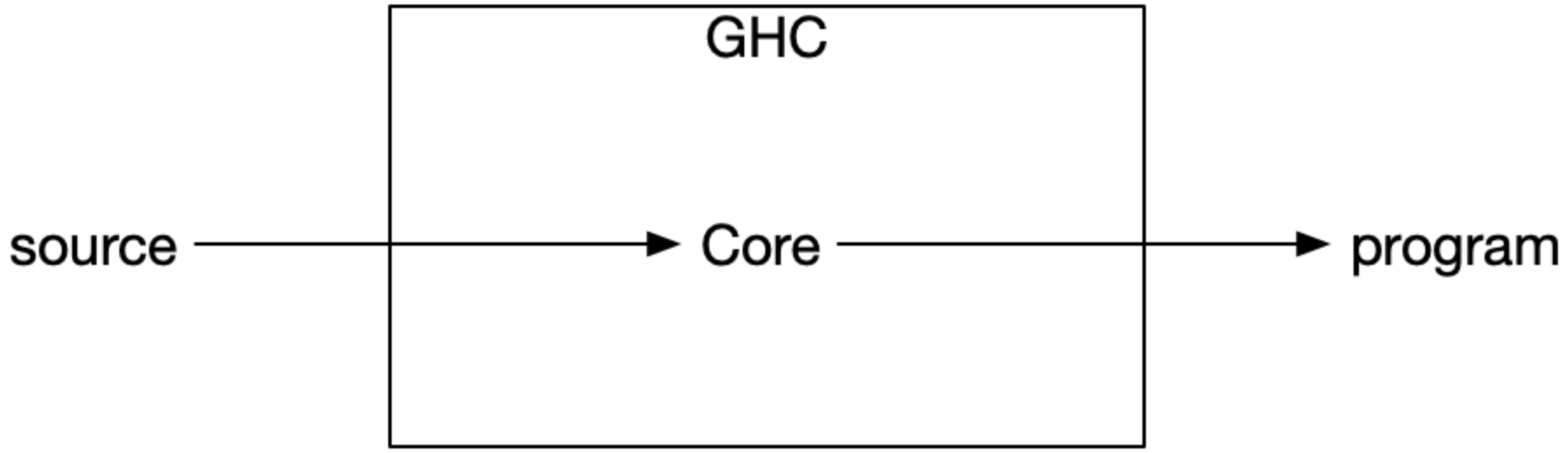
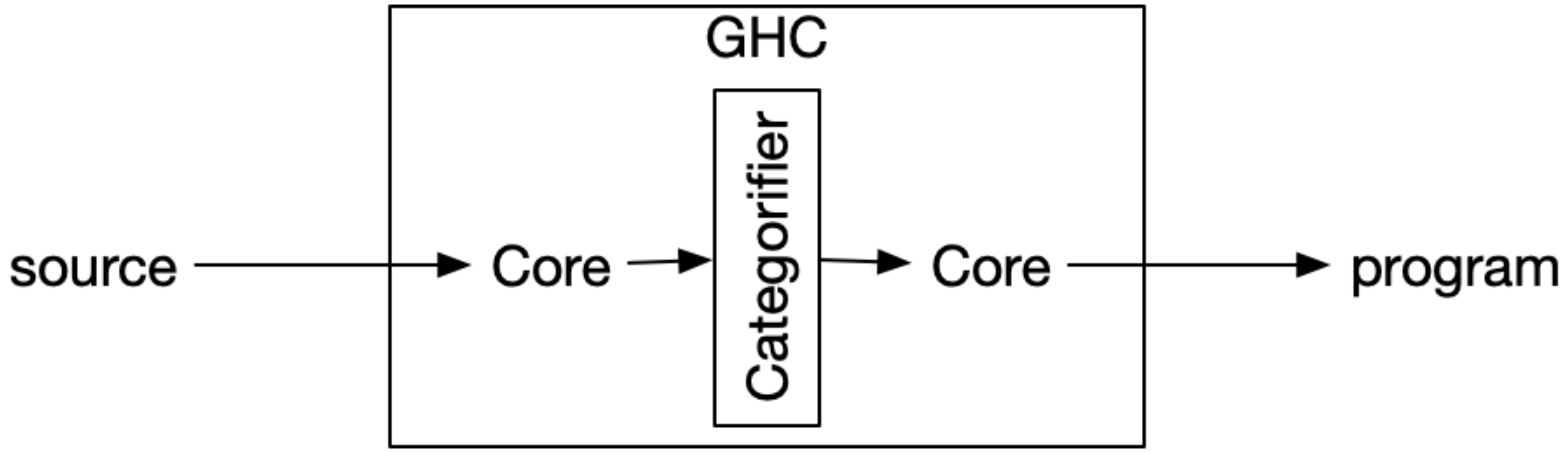
Compiling to Categories

Compiling to Categories

- http://conal.net/papers/compiling-to-categories/





overview

- what we're using this for
- what didn't work
- what we improved
- how to use it
- what's left to do

KITTYHAWK

N239HV

N242HV
N243HV
N223HV
4

N240HV

con-kitty / categorifier-c Public
Unpin
Watch 2
Fork 0
Starred 4
Code
Issues 3
Pull requests 2
Actions
Projects
Wiki
Security
Insights
Settings
master
categorifier-c / README.md
Go to file
sellout Merge pull request #28 from con-kitty/categorify …
Latest commit 0815468 12 days ago
History
2 contributors
138 lines (103 sloc) 6.68 KB
Raw
Blame

C Backend for Categorifier categorifier-c passing

This repo is a backend for the categorifier plugin, with the purpose of compiling Haskell to C. It contains a cartesian closed category, referred to as CCat, as well as code that converts morphisms in this category into C code.

At a high level, a Haskell function is compiled into a C function via the following steps:

- The categorifier plugin (i.e., frontend) maps the Hask morphism (i.e., the Haskell function) into a CCat morphism.
    - Hask is, roughly speaking, a category where objects are Haskell types and morphisms are Haskell functions.
    - CCat is a category where objects are Haskell types, and a morphism from `A` to `B` is a Haskell function from `TargetOb A` to

Compiling to Categories

Anything

Compiling $_\wedge$ to Categories

me (Greg Pfeil)

- Haskell experience: 12+ years
- compiler experience: 15+ years
- at Kittyhawk: 2.5 years
- working remote from Maui, Hawaiʻi

me (Greg Pfeil)

- Haskell experience: 12+ years
- compiler experience: 15+ years
- at Kittyhawk: 2.5 years
- working remote from Maui, Hawai‘i
- breathhold: 4:01
- depth: 41m / 136’
  (about 11 stories)

the team

Greg Horn

Chris McKinlay

Matt Peddie

Ziyang Liu

Ian Kim

Greg Pfeil

con-kitty / categorifier
Public
Unpin
Watch 3
Fork 0
Starred 5
Code
Issues 7
Pull requests 3
Discussions
Actions
Projects
Wiki
Security
master
categorifier / README.md
Go to file
sellout Rename the repo and many modules. …
Latest commit 5a1bb54 18 days ago
History
2 contributors
11 lines (6 sloc) 840 Bytes
Raw
Blame

Categorifier

categorifier passing

Defining novel interpretations of Haskell programs.

You probably want to look at the plugin README.

Contributing

There are compatible direnv and Nix environments in the repo to make it easy to build, test, etc. everything with consistent versions to

the project

- sum types
- recursion
- multiple modules (and third-party dependencies)
- various type class hierarchies
- FFI integration
- references (abstraction in the target category)
- improved performance
- rich error reporting, with suggestions

how to use it

```
newtype Hask a b = Hask {runHask :: a -> b}

instance Category Hask where
  id = Hask id
  Hask g . Hask f = Hask $ g . f

instance Arrow Hask where
  arr = Hask
  Hask f *** Hask g = Hask $ f *** g

wrap_negate :: Num a => a `Hask` a
wrap_negate = Categorify.expression negate

main :: IO ()
main = print $ runHask wrap_negate (5 :: Int)
```

how to use it

```haskell
newtype Hask a b = Hask {runHask :: a -> b}

instance Category Hask where
  id = Hask id
  Hask g . Hask f = Hask $ g . f

instance Arrow Hask where
  arr = Hask
  Hask f *** Hask g = Hask $ f *** g


Categorify.function 'negate [t|Hask|] []

main :: IO ()
main = print $ runHask wrap_negate (5 :: Int)
```

how to use it

```
executable trivial-example
  main-is: NegateFunction.hs
  ghc-options:
    -fplugin Categorifier
  build-depends:
    , base
    , categorifier-plugin
    -- needed for generated code
    , categorifier-category
    , categorifier-client
    , ghc-prim
```

how to use it

```
Categorify.function 'Lens.view [t|Syntactic.Syn|] []

main :: IO ()
main = putStrLn . Syntactic.render $ wrap_view @Int @((->) Int)

-- unsafeCoerce
--   . apply
--   . ( id ***
--       curry ((unsafeCoerce . unsafeCoerce)
--                    . exr)
--     )
--   . dup
```

how to use it

```
executable syntax-example
  main-is: Syntax.hs
  ghc-options:
    -fplugin Categorifier
    -fplugin-opt
    Categorifier:hierarchy:Categorifier.Hierarchy.ConCat.classHierarchy
  build-depends:
    , categorifier-concat-examples
    , categorifier-concat-integration
    , categorifier-plugin
    , lens
    , …
```

how to use it

```haskell
instance Category Syn where
  id  = app0 "id"
  (.) = app2 "."

instance AssociativePCat Syn where
  lassocP = app0 "lassocP"
  rassocP = app0 "rassocP"

instance MonoidalPCat Syn where
  (***)  = app2 "***"
  first  = app1 "first"
  second = app1 "second"
```

```haskell
instance BraidedPCat Syn where
  swapP = app0 "swapP"

instance ProductCat Syn where
  exl = app0 "exl"
  exr = app0 "exr"
  dup = app0 "dup"

instance AssociativeSCat Syn where
  lassocS = app0 "lassocS"
  rassocS = app0 "rassocS"
```

dealing with types

```haskell
import Categorifier.Client

data MyType a b = JustAn a | BothAn a b | Neither

instance HasRep MyType where
  type Rep MyType = Either (Either a (a, b)) ()
  abst = either (either JustAn (uncurry BothAn)) (const Neither)
  repr = \case
    JustAn a -> Left (Left a)
    BothAn a b -> Left (Right a b)
    Neither -> Right ()
```

dealing with types

```
import Categorifier.Client

data MyType a b = JustAn a | BothAn a b | Neither

deriveHasRep ‘’MyType
```

Anything

Compiling $_{\wedge}$ to Categories

(Almost) Anything

Compiling ^ to Categories

NativeCat

```
class NativeCat k (tag :: Symbol) a b where
  nativeK :: a `k` b

instance
  (KRound CExpr a, CExpr a ~ TargetOb a) =>
  NativeCat
    Cat
    “Categorifier.C.KTypes.Round.kRoundDouble”
    (C Double)
    (C a)
  where
  nativeK = cat kRoundDouble
```

automatic interpretation

```
type AutoInterpreter =
  (Plugins.Type -> DictionaryStack Plugins.CoreExpr) ->
  Plugins.Type ->              -- ^ the category
  Plugins.Type ->              -- ^ the original function's type
  Plugins.Id ->                -- ^ the original function
  [Plugins.CoreExpr] ->        -- ^ any arguments applied at the call site
  CategoryStack (Maybe Plugins.CoreExpr)
```

existential types

```haskell
-- won't work
type Rep (Meh b c) = (forall a. (a, b), c)

-- might work
type Rep (Meh b c) = (Exists (Flip (,) b), c)
```

mutual recursion

```
-- won’t work
let a = Foo {bar = b * c, baz = 3 + bar a}

-- works
let bar’ = b * c
    a = Foo {bar = bar’, baz = 3 + bar’}
```

Anything

Compiling ^ to Categories

Thank you! Any questions?

- https://github.com/sellout/compiling-anything-to-categories
- greg@technomadic.org
- @sellout (Twitter)

- http://conal.net/papers/compiling-to-categories
- https://github.com/con-kitty/categorifier