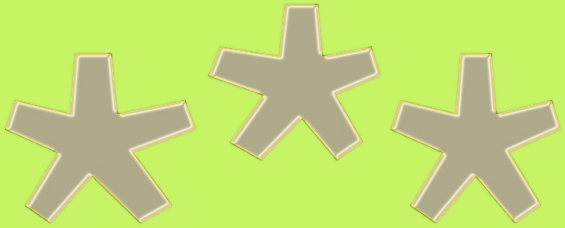


Functional Programming and the Web

Frontend Development in Purescript

Michael Karg

Jürgen Nicklisch-Franken

Hello!

We are Symbolian

- Founded February 2015 in Berlin
- Currently 5 employees + external expertise
- Efficient software factory for high quality code
- Technology-driven innovation

1. Purescript

The language

History and authors

- Phil Freeman
- Gary Burgess – Developer at SlamData
- John A. De Goes – CTO of SlamData
- In development since 2013, current version 0.8 (Feb. 2016)
- Notable companies: SlamData – Visual Analytics for NoSQL (Boulder, CO)

Project structure

- psc – purescript compiler
- psci – interactive REPL
- psc-bundle - “linker” / deployment tool
- psc-docs – documentation generator (markdown format)
- psc-publish – documentation generator for the pursuit API search engine
- psc-hierarchy – graphical doumentation generator for type classes
- Project language: Haskell
- psc target language: JavaScript

Language properties

- purely functional
- strong, static type system
- compiles to human-readable JavaScript
- standalone output – no dedicated JavaScript runtime necessary

Code generation examples

```
increment :: forall f. (Functor f) ⇒ f Int → f Int

increment = map (+1)
```

```
var increment = function (__dict_Functor_0) {
    return Prelude.map(__dict_Functor_0)(function (_0) {
        return _0 + 1 | 0;
    });
};
```

```
main = do
  ref ← newSTRef "Hello"
  readSTRef ref >>= log
```

```
var main = function __do() {
    var _1 = Control_Monad_ST.newSTRef("Hello")();
    return Prelude[">>="]
          (Control_Monad_Eff.bindEff)
          (Control_Monad_ST.readSTRef(_1))
          (Control_Monad_Eff_Console.log)();
};
```

Language properties

- Purescript's design promises a smooth transition for developers from Haskell
- similar type class hierarchy

```
class (Eq a) <= Ord a where

    compare :: a → a → Ordering
```

- similar abstractions and control structures (Monads / Effects, Applicatives, Functors, ...)
- similar syntax (pattern matching, do-notation, modules, sum and product data types, newtypes / type aliases, language keywords)
- similar type annotations

```
liftM1 :: forall m a b. (Monad m) ⇒ (a → b) → m a → m b

liftM1 f a = do

    a' ← a

    return (f a')
```

Differences from Haskell - Syntax

- no tuple syntax
- no cons patterns (less powerful pattern matching)
- row polymorphism in records

```
let showPerson { first: x, last: y } = y ++ ", " ++ x

let person1 =  { first: "Phil", last: "Freeman" }

showPerson person1

"Freeman, Phil"

showPerson { first: "Phil", last: "Freeman", location: "Los Angeles" }

"Freeman, Phil"
```

- record access

```
person1.last
```

Differences from Haskell – Types

- Explicit `forall`
- Named instances
- Extensible Effects

```
main :: forall eff.
      → Eff
         ( canvas :: Canvas,
         , random :: RANDOM
         , err    :: EXCEPTION
         , st     :: ST ConeST
         , dom    :: DOM
         | eff
         )
     Unit
```

Differences from Haskell – The JS world

- evaluated by a JavaScript engine (V8, Spidermonkey...)

  ==> strict evaluation; no concurrency; varying performance characteristics

- package splits are cheap

  ==> minimal Prelude (~ 730 sloc), highly specialized packages (purescript-either, purescript-maybe, ...)

- FFI into the JS world

```
foreign import concatString :: String → String → String
```

```
exports.concatString = function (s1) {
  return function (s2) {
    return s1 + s2;
  };
};
```

2. Purescript

The ecosystem

Basics: building

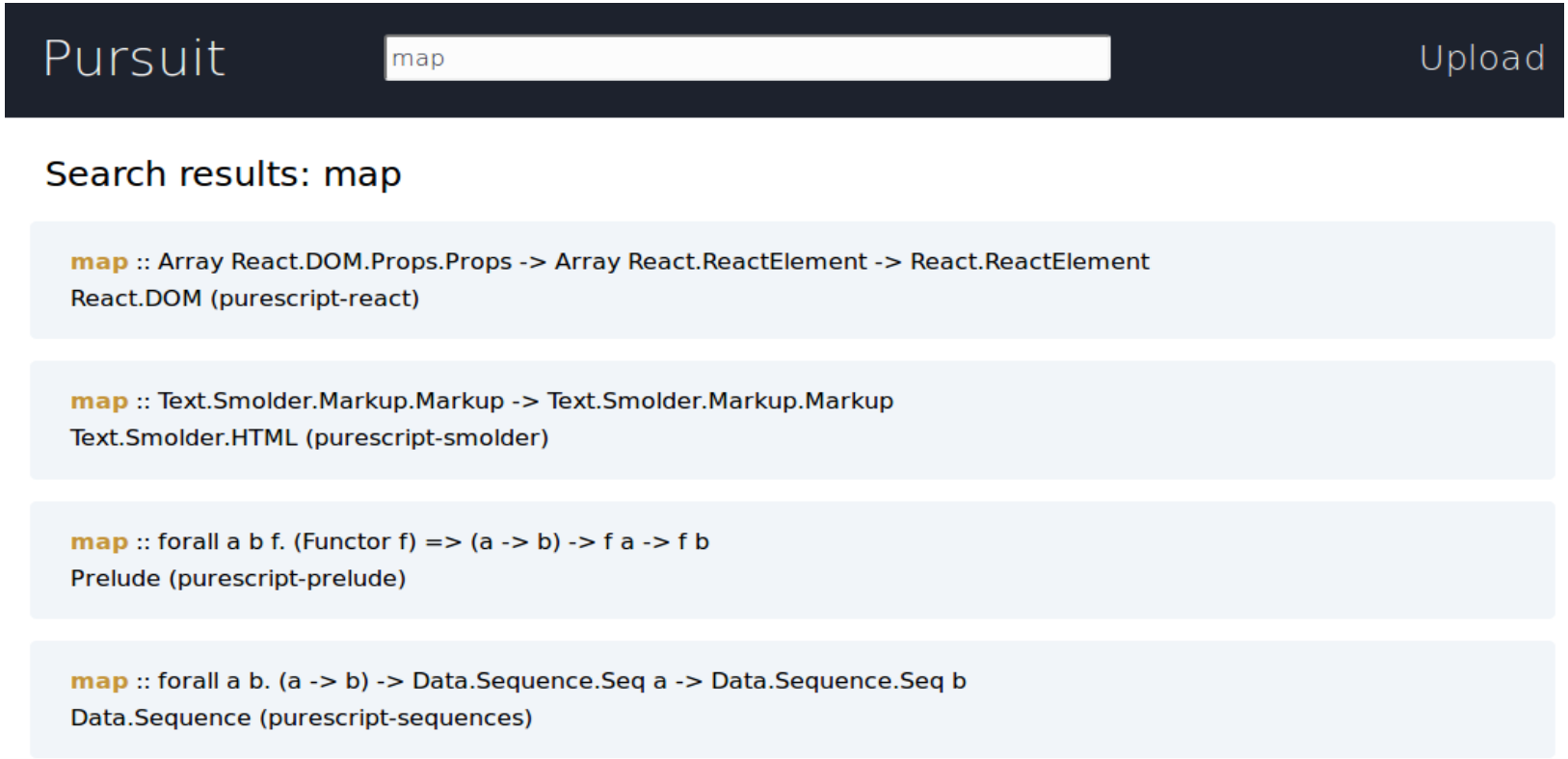
- psc + node / npm, bower – well-known tools of the JS world
- pulp build system

  ==> no difficulties for any frontend developer
- However: very little compiler optimizations (only TCO and DCE)

  ==> without inlining and partial vs. total application in function calls, the equivalent plain JS code performs notably better

Search: pursuit -- https://pursuit.purescript.org

- Search by symbol name, type or package name (just like hoogle or hayoo)



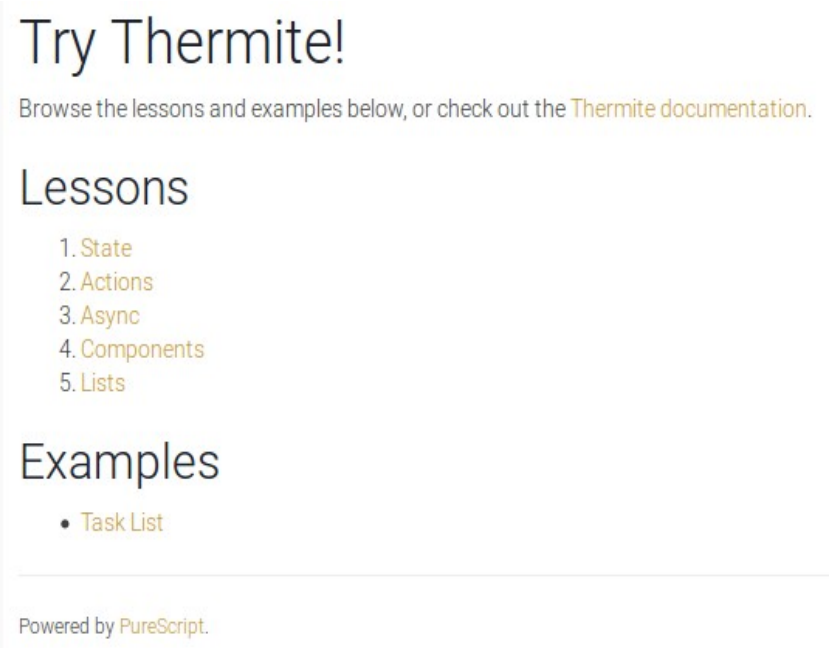
Framework: Thermite

- wrapper for ReactJS with a clean functional API

```
examples :: Array Link
examples =
  [ { title: "Task List"
    , gist: "f5f273e4c5e4161fceff"
    }
  ]

renderLink :: Link -> Array R.ReactElement
renderLink link =
  [ R.a [ RP.href ("?gist=" <> link.gist)
        , RP.target "_top"
        ]
        [ R.text link.title ]
  ]

render :: T.Render _ _ _
render _ _ _ _ =
  [ R.h1' [ R.text "Try Thermite!" ]
  , R.p'  [ R.text "Browse the lessons and examples below, or check out the "
          , R.a [ RP.href "http://pursuit.purescript.org/packages/purescript-thermite/"
                , RP.target "_new"
                ]
                [ R.text "Thermite documentation" ]
          , R.text "."
          ]
  , R.h2' [ R.text "Lessons" ]
  , R.ol' (map (R.li' <<< renderLink) lessons)
  , R.h2' [ R.text "Examples" ]
  , R.ul' (map (R.li' <<< renderLink) examples)
  , R.hr' []
  , R.p'  [ R.small' [ R.text "Powered by "
                     , R.a [ RP.href "http://purescript.org/"
                           , RP.target "_new"
                           ]
                           [ R.text "PureScript" ]
                     , R.text "."
                     ]
          ]
  ]
```



Framework: Halogen

- a type-safe declarative UI library
- native Purescript implementation

```
type State =
  { count :: Int
  }

data Input a
  = Increment a
  | Decrement a

ui :: forall g. (Functor g) ⇒ Component State Input g
ui = component render eval
  where
    render state =
      H.div_
        [ H.button [ E.onClick $ E.input_ Decrement ]
                   [ H.text "-" ]
        , H.p_ [ H.text (show state.count)]
        , H.button [ E.onClick $ E.input_ Increment ]
                   [ H.text "+" ]
        ]

eval :: Eval Input State Input g
eval (Increment next) = do
   modify (\state → state { count = state.count + 1 })
   pure next
eval (Decrement next) = do
   modify (\state → state { count = state.count - 1 })
   pure next
```

Graphics: LambdaCube 3D

- purely functional DSL for programming the GPU
- WebGL rendering backend uses Purescript

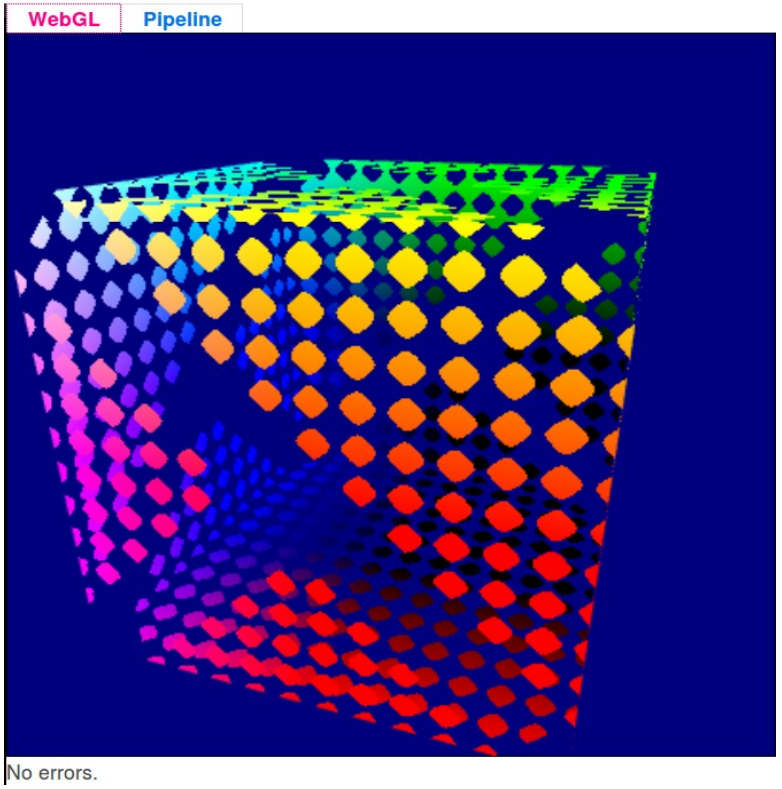
```
len2 v = v%x*v%x + v%y*v%y + v%z*v%z

makeFrame (time :: Float)
          (projmat :: Mat 4 4 Float)
          (vertexstream :: PrimitiveStream Triangle (Vec 4 Float))

    = imageFrame (emptyDepthImage 1, emptyColorImage navy)
  `overlay`
      vertexstream
    & mapPrimitives (\x -> (scale 0.5 (projmat *. x), x))
    & rasterizePrimitives (TriangleCtx CullNone PolygonFill NoOffset LastVertex) Smooth
    & filterFragments ffilter
    & accumulateWith (DepthOp Less True, ColorOp NoBlending (V4 True True True True))
  where
    h = 0.25

    ffilter v =
          (v%y + v%x + v%z > 1 || abs (v%y - v%x + v%z - 1.0) > h)
        && (v%y + v%x - v%z > 1 || abs (v%y + v%x + v%z - 1.0) > h)
        && (v%y - v%x - v%z > 1 || abs (v%y + v%x - v%z - 1.0) > h)
        && (v%y - v%x + v%z > 1 || abs (v%y - v%x - v%z - 1.0) > h)
        && len2 v' > abs (3 * sin (1 * time))
      where
        v' = sin (v *! 20)

main = renderFrame $
   makeFrame (Uniform "Time")
             (Uniform "MVP")
             (fetch_ "stream4" (Attribute "position4"))
```



Conclusion

- wide and diverse ecosystem for such a relatively young language
- fast-growing toolbox for tackling your individual use case
- plenty of interesting approaches and projects to toy around with and get fresh ideas

3. Purescript

The community

Why community?

- Community: Populace of any language's ecosystem
- Community size and structure directly influences the way a language is being used, and can be used
- When choosing a language for commercial development and productive use: get to know the community as well
- Given the demands of a commercial application of any language: will its community be a support, or an obstacle?
- NB. No intention to judge people; just one adaptation more of your development process, trying to avoid false expectations

The purescript community

- purescript compiler: 4 individuals ~ 2/3 of all commits
- same individuals responsible for the vast majority of purescript packages
- ==> tiny community

The purescript community

- tight-knit communication
- communication often in closed circles or one on one
- compared to the Haskell community, seldom open debates or call for feedback
- discussions take place in github's issue tracker or the purescript IRC channel – not the best way to keep up or retrace
- consequence: hasty and premature decisions, lack of prioritization of open issues and TODOs

Examples

- explicit imports – implement, protest, rethink, shrug
- removal of cons patterns... in a language boasting pattern matching
- restructuring of type class hierarchy; (cf. Int / Num)

  NB. w/o inliner, each (+), (-), ... still corresponds to a dictionary lookup of the instance's method at runtime, impacting performance
- orphan instances banned completely
- API changes in and refactoring of the Prelude
- ==> we're not talking about language periphery here
- ==> additionally, all of the above changes in no more than 6 months

The Ugly

- frequent breakage of your code base
- extra expenditure of time (> 1d) just to get it to build again
- due to the character of the community, changes may be pleasing from an aesthetical-theoretical POV, but real world code becomes horrible

The Bad

- due to the size of the community: still no official specs or roadmap for Purescript 1.0
- please see as a suggestion, since it could mitigate a lot of the current Ugly
- Purescript users know what they're getting into – and when!
- Could serve as a base for debate and discussion, enabling feedback
- Would make it possible to prioritize, order and allocate development resources accordingly

The Good

- familiarity
- everyone in the community puts in huge efforts
- development of Purescript advances at a fast pace
- short response times for any issue you might have
- Kudos to the community for the great learning resources (e.g. Purescript book) and the great tooling (e.g. pursuit API search)

4. Purescript
The experience

still doing GUI stuff?

GUI Programming in Haskell

- „GUI is not needed“ (2010)
- „I see why'd you need doing GUI code. Most of the time it's unnecessary, though.“ (2006)
- „wxHaskell and Gtk2Hs are the main two GUI libraries -- they're reasonably comparable in terms of quality.“ (2010)
- Can't wait until we have a functional lib with the same quality of gtk2hs (2006)
- „Javascript is pretty handy for GUI“ (2005)
- „I'm not certain what you mean, do you want a GUI or a web UI“ (2009)

The pretty and good and unsuable or the ugly and evil and usable

GUI and the Browser

ghcjs	Elm	purescript	Idris
Haskell (GHC) to JavaScript compiler	Functional programming in your browser	Haskell like language	A Language with Dependent Types and Javascript backend
Full Haskell Language and Runtime	Designed around high-level front-end development	Has Typeclasses and RankNTypes.	Full dependent types with dependent pattern matching
Lightweight preemptive threading	Time-travelling debugger and Hot-swapping of code	Has row polymorphism and extensible effects.	Dependent records with projection and update
All goodies like MVars, WeakReferences, ...	First class FRP and Reactive DOM	No runtime.	Tactic based theorem proving
+	+++	++	+++

Our contributions to the community

- Wrote purescript-webgl-generator in Haskell to generate purescript to Javascript FFI code from WebGL IDL (Khronos)
- Added higher level binding code to make the use of the API convenient and type safe (purescript-webgl)
- Only magic is in defining „the binding“(exchange to the GPU) in the purescript type system
- Wrote additional needed packages purescript-vector, purescript-matrix and purescript-typedarray
- Wrote purescript-webgl-examples package to test our bindings against the lessons 1-9 from http://learningwebgl.com
- Switched to use a global *webgl* context variable for performance

The good an the bad

Plus

- The core of the typechecker and codegenerator is relatively stable (we found some bugs in both)
- The extensible record system is fantastic (see Extensible records with scoped labels, Daan Leijen), the extensible Effects system (build onto records) is usable.
- The generated code is easy and transparent to follow.
- Error messages have much improved.

Minus

- A lot of breaking changes in releases (e.g. 0.7 meltdown) with some annoying decisions. (E.g. Cons-patterns have been disallowed)
- Tendency in the community to find the one an right way and forbid other (e.g. implicit exports)
- Bad runtime performance:
  - Naive currying
  - No inlinining
  - Naive Type class lookup

Compilation -1-

coneNumLevel

Purescript

```
-- | How many ConeNodes are contained at this level?
coneNumLevel :: forall a.ConeZip a -> Int
coneNumLevel (ConeZip zipper) = L.length zipper.before + 1 + L.length zipper.after
```

Javascript

```
var coneNumLevel = function (v) {
    return (Data_List.length(v.before) + 1 | 0) + Data_List.length(v.after) | 0;
};
```

coneModify

Purescript

```
-- | Modify the current 'ConeNode' with a function.
coneModify :: forall a. (ConeNode a -> ConeNode a) -> ConeZip a -> ConeZip a
coneModify f (ConeZip zipper) = ConeZip zipper {content  = f zipper.content}
```

Javascript

```
var coneModify = function (f) {
    return function (v) {
        var $177 = {};
        for (var $178 in v) {
            if (v.hasOwnProperty($178)) {
                $177[$178] = v[$178];
            };
        };
        $177.content = f(v.content);
        return $177;
    };
};
```

Compilation -2-

coneEnumerate

Purescript

```
coneEnumerate :: forall a eff. (ConeEntry a) ⇒ ConeZip a → ConeEff eff (Array (ConeZip a))
coneEnumerate zipper = do
  condChildren ← coneDown zipper
  let children = case condChildren of
                   Nothing → []
                   Just child → getChildren child
  childrenResults ← traverse coneEnumerate children
  return (zipper : concat childrenResults)
```

Javascript

```
var coneEnumerate = function (dictConeEntry) {
    return function (zipper) {
        return function __do() {
            var v = coneDown(dictConeEntry)(zipper)();
            var children = (function () {
                if (v instanceof Data_Maybe.Nothing) {
                    return [  ];
                };
                if (v instanceof Data_Maybe.Just) {
                    return getChildren(dictConeEntry)(v.value0);
                };
                throw new Error("Failed pattern match at ConeCanvas.ConeZipper line 84, column 7 - line 87, column 3: " + [ v.constructor.name ]);
            })();
            var v1 = Data_Traversable.traverse(Data_Traversable.traversableArray)(Control_Monad_Eff.applicativeEff)(coneEnumerate(dictConeEntry))(children)();
            return Data_Array[":"](zipper)(Data_Array.concat(v1));
        };
    };
};
```

The Ecosystem we use

- We use *Bower* as package system, as it is the recommended solution
- We use *Pulp* as build system, as it is specially developed for this purpose
- We use the Browsers WebDevelopers tools for low level debugging and profiling
- We use *Pursuit* to query purescript packages on the Web
- We use the *Atom editor* with *purescript-langugage* and *purescript-ide* (atom package) and *purescript-ide* (haskell package) as basic IDE.

The Product we build:

The problem we attack with Symbolian:

- How to bring order to data!
- How to resolve the problem of data separation

ConeCanvas is a widget for data classification:

- We use the 3rd dimension
- We reinvent the classical treeview
- We make data intuitively manageable
- We improve the human computer interaction

Thanks!

Any questions?


Symbolian
Knowledge Processing