Jacinda—Implementing an Efficient Functional Stream Processing Language

Vanessa McHale

11 Oct 2024

Outline

Outline

Outline

Outline

Outline

Unix Command-Line

-
  ```
  ~ % ps
    PID   TT  STAT      TIME COMMAND
    ⋮
  22970 s002  S      0:00.07 -zsh
  22989 s002  S+     0:00.21 cabal repl lib:apple
  23031 s002  S+     0:00.03 /Users/vanessa/.ghcup/bin/ca
  23033 s002  S+     0:10.46 /Users/vanessa/.ghcup/ghc/9
  ```

Unix Command-Line

-
  ```
  ~ % ps
    PID   TT  STAT      TIME COMMAND
    ⋮
  22970 s002  S      0:00.07 -zsh
  22989 s002  S+     0:00.21 cabal repl lib:apple
  23031 s002  S+     0:00.03 /Users/vanessa/.ghcup/bin/ca
  23033 s002  S+     0:10.46 /Users/vanessa/.ghcup/ghc/9
  ```

-
  ```
  ~ % ps | rg cabal | rg -v rg
  23400 s002  S+     0:00.16 cabal repl lib:apple
  23401 s002  S+     0:00.04 /Users/vanessa/.ghcup/bin/ca
  23403 s002  S+     0:11.17 /Users/vanessa/.ghcup/ghc/9
  ```

Unix Command-Line

-
  ```
  ~ % ps | rg 'cabal' | rg -v 'rg' | cut -d' ' -f1
  24144
  24196
  24198
  ```

Unix Command-Line

-
  ```
  ~ % ps | rg 'cabal' | rg -v 'rg' | cut -d' ' -f1
  24144
  24196
  24198
  ```

-
  ```
  ~ % ps | rg cabal | rg -v rg | \
        cut -d' ' -f1 | xargs kill
  ```

Structured Text in Unix

-
  ```
  % otool -l $(locate librure.dylib)
  ⋮
  Load command 12
            cmd LC_LOAD_DYLIB
        cmdsize 56
           name /usr/lib/libiconv.2.dylib (offset 24)
     time stamp 2 Wed Dec 31 19:00:02 1969
        current version 7.0.0
  compatibility version 7.0.0
  Load command 13
            cmd LC_LOAD_DYLIB
        cmdsize 56
           name /usr/lib/libSystem.B.dylib (offset 24)
     time stamp 2 Wed Dec 31 19:00:02 1969
        current version 1351.0.0
  ⋮
  ```

Structured Text in Unix

-
  ```
   % otool -l $(locate librure.dylib)
  ⋮
  Load command 12
            cmd LC_LOAD_DYLIB
        cmdsize 56
           name /usr/lib/libiconv.2.dylib (offset 24)
     time stamp 2 Wed Dec 31 19:00:02 1969
        current version 7.0.0
  compatibility version 7.0.0
  Load command 13
            cmd LC_LOAD_DYLIB
        cmdsize 56
           name /usr/lib/libSystem.B.dylib (offset 24)
     time stamp 2 Wed Dec 31 19:00:02 1969
        current version 1351.0.0
  ⋮
  ```
- $\underbrace{\texttt{\$(locate librure.dylib)}}_{\text{Command substitution}}$

Structured Text in Unix

```
 % otool -l $(locate librure.dylib)
⋮
Load command 12
          cmd LC_LOAD_DYLIB
      cmdsize 56
         name /usr/lib/libiconv.2.dylib (offset 24)
   time stamp 2 Wed Dec 31 19:00:02 1969
      current version 7.0.0
compatibility version 7.0.0
Load command 13
          cmd LC_LOAD_DYLIB
      cmdsize 56
         name /usr/lib/libSystem.B.dylib (offset 24)
   time stamp 2 Wed Dec 31 19:00:02 1969
      current version 1351.0.0
⋮
```

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $$\underbrace{\texttt{\$1 \~{} /\^{}name/}}_{\text{Execute when this is true}}$$

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```

- $\underbrace{\texttt{\$1 \textasciitilde{} /\^{}name/}}_{\text{Execute when this is true}}$
- $\underbrace{\texttt{\{print \$2\}}}_{\text{Do this}}$

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{\$1 \~{} /\^{}name/}}_{\text{Execute when this is true}}$
- $\underbrace{\texttt{\{print \$2\}}}_{\text{Do this}}$
- `<PATTERN> { <ACTION> }`

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
        awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{\$1 \textasciitilde{} /\^{}name/}}_{\text{Execute when this is true}}$
- $\underbrace{\texttt{\{print \$2\}}}_{\text{Do this}}$
- `<PATTERN> { <ACTION> }`
- Split, scan for pattern

Structured Text—Patterns

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          awk '$1 ~ /^name/ {print $2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{\$1 ~ /^name/}}_{\text{Execute when this is true}}$
- $\underbrace{\texttt{\{print \$2\}}}_{\text{Do this}}$
- `<PATTERN> { <ACTION> }`
- Split, scan for pattern
- Regular expressions are discovered

Structured Text—Jacinda

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          ja '{`1 ~ /^name/}{`2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```

Structured Text—Jacinda

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
        ja '{`1 ~ /^name/}{`2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{`1 ~ /^name/}}_{\text{Filter stream by this}}$

Structured Text—Jacinda

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          ja '{`1 ~ /^name/}{`2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{`1 \~{} /\^{}name/}}_{\text{Filter stream by this}}$
- $\underbrace{\texttt{\{`2\}}}_{\text{Stream contents}}$

Structured Text—Jacinda

-
  ```
  ~ % otool -l $(locate librure.dylib) | \
          ja '{`1 ~ /^name/}{`2}'

  /usr/local/lib/librure.dylib
  /usr/lib/libiconv.2.dylib
  /usr/lib/libSystem.B.dylib
  ```
- $\underbrace{\texttt{`1 ~ /^name/}}_{\text{Filter stream by this}}$
- $\underbrace{\texttt{\{`2\}}}_{\text{Stream contents}}$
- `{ <PROPOSITION> }{ <EXPR> }`

Structured Text—Jacinda III

-
  ```
  ~ % echo $PATH
  /Users/vanessa/.ghcup/bin:/Library/Frameworks/Python.fr

  ~ % echo $PATH | ja -R: "fold1 (\x.\y. x+'\n'+y) \$0"
  /Users/vanessa/.ghcup/bin
  /Library/Frameworks/Python.framework/Versions/3.13/bin
  /usr/local/bin
  /System/Cryptexes/App/usr/bin
  /usr/bin
  /bin
  /usr/sbin
  /sbin
  ⋮
  ```

Tour

- Dump every assembly instruction (Pepijn de Vos)

Tour

- Dump every assembly instruction (Pepijn de Vos)

-
  ```
  objdump -d /usr/bin/* | cut -f3 | \
        ja 'dedup (filter (~ /^[a-z]+/) $0)'
  ```

Tour

- Dump every assembly instruction (Pepijn de Vos)

-
  ```
  objdump -d /usr/bin/* | cut -f3 | \
        ja 'dedup (filter (~ /^[a-z]+/) $0)'
  ```

-
  ```
  ⋮
  dup.2s
  add.2s
  xtn.2s
  movi.8b
  cmhs.2d
  umull.8h
  umlal2.8h
  uaddlv.16b
  uaddlv.8h
  zip1.2s
  ⋮
  ```

Tour
Functional Approach

- `dedup : Ord a :=> Stream a -> Stream a`

Tour

Functional Approach

- `dedup : Ord a :=> Stream a -> Stream a`
- `filter : (a -> Bool) -> Stream a -> Stream a`

Tour

Functional Approach

- `dedup : Ord a :=> Stream a -> Stream a`
- `filter : (a -> Bool) -> Stream a -> Stream a`
- `filter` $\underbrace{\texttt{(\textasciitilde{} /\textasciicircum{}[a-z]+/)}}_{\text{Curried}}$

Tour

Functional Approach

- `dedup : Ord a :=> Stream a -> Stream a`
- `filter : (a -> Bool) -> Stream a -> Stream a`
- `filter` $\underbrace{\texttt{(\textasciitilde{} /\textasciicircum{}[a-z]+/)}}_{\text{Curried}}$
- Type signatures in manpage

Tour II

-
  ```
  ~ % readelf -p '.debug-ghc-link-info' $(which pandoc)

  String dump of section '.debug-ghc-link-info':
    [     5]  N
    [     c]  GHC link info
    [    1c]  (([\"-lHSpandoc-lua-engine-0.2.1.3-52609aae
  ```

Tour II

-
  ```
  ~ % readelf -p '.debug-ghc-link-info' $(which pandoc)

  String dump of section '.debug-ghc-link-info':
    [     5]  N
    [     c]  GHC link info
    [    1c]  (([“-lHSpandoc-lua-engine-0.2.1.3-52609aae
  ```

-
  ```
  $ readelf -p '.debug-ghc-link-info' $(which pandoc) \
      | ja -R, 'catMaybes {|`0 ~* 1 /-lHS([A-Za-z][A-
  Za-z0-9\-]*\d+(\.\d+)*)/}'

  pandoc-lua-engine-0.2.1.3
  pandoc-lua-marshal-0.2.5
  pandoc-3.1.12.3
  typst-0.5.0.2
  yaml-0.11.11.2
  ...
  ```

Tour II

Functional Programming

- `-R,` split into records by `,` (not lines)

Tour II

Functional Programming

- `-R,` split into records by `,` (not lines)
- ``catMaybes {|`0 ~* 1 /<REGEX>/}``

Tour II

Functional Programming

- `-R`, split into records by `,` (not lines)
- ``catMaybes {|`0 ~* 1 /<REGEX>/}``
- `~* : Str -> Int -> Regex -> Option Str`

Tour II

Functional Programming

- `-R`, split into records by `,` (not lines)
- ``catMaybes {|`0 ~* 1 /<REGEX>/}``
- `~* : Str -> Int -> Regex -> Option Str`
- `catMaybes : Stream (Option a) -> Stream a`

Tour II

Functional Programming

- `-R`, split into records by `,` (not lines)
- ``catMaybes {|`0 ~* 1 /<REGEX>/}``
- `~* : Str -> Int -> Regex -> Option Str`
- `catMaybes : Stream (Option a) -> Stream a`
- Capture groups (AWK not fluent)

Tour III

Capture Groups

- Get latest `less` version

Tour III

Capture Groups

- Get latest `less` version
-
```
~ % curl -s 'https://www.greenwoodsoftware.com/less/dow
        rg 'less-(\d+)\.tar\.gz' -r '$1' -o | \
        head -n1
661
```

Tour III

Capture Groups

- Get latest `less` version

-
  ```
  ~ % curl -s 'https://www.greenwoodsoftware.com/less/dow
          rg 'less-(\d+)\.tar\.gz' -r '$1' -o | \
          head -n1
  661
  ```

-
  ```
  fold1 (\x.\y. x)
     (catMaybes {|`0 ~* 1 /less-(\d+)\.tar\.gz/})
  ```

Tour III

Capture Groups

- Get latest `less` version

-
  ```
  ~ % curl -s 'https://www.greenwoodsoftware.com/less/dow
        rg 'less-(\d+)\.tar\.gz' -r '$1' -o | \
        head -n1
  661
  ```

-
  ```
  fold1 (\x.\y. x)
    (catMaybes {|`0 ~* 1 /less-(\d+)\.tar\.gz/})
  ```
- `catMaybes` makes sense

Tour III

Capture Groups

- Get latest `less` version

-
  ```
  ~ % curl -s 'https://www.greenwoodsoftware.com/less/dow
          rg 'less-(\d+)\.tar\.gz' -r '$1' -o | \
          head -n1
  661
  ```

-
  ```
  fold1 (\x.\y. x)
    (catMaybes {|`0 ~* 1 /less-(\d+)\.tar\.gz/})
  ```
- `catMaybes` makes sense
- `fold1 (\x.\.y x)` for last in stream

Tour IV

- Libraries used to build but not present in final artifact

```
diff \
    <(readelf ... $(which ja) | ja -R, 'catMaybes {|`0
    <(nm $(which ja) | sed ... | ja 'dedup (catMaybes -
⋮
< microlens-0.4.13.1
< microlens-mtl-0.2.0.3
23d12
< pretty-1.1.3.6
30d18
⋮
```

Tour IV

- Libraries used to build but not present in final artifact

```
diff \
    <(readelf ... $(which ja) | ja -R, 'catMaybes {|`0
    <(nm $(which ja) | sed ... | ja 'dedup (catMaybes -
⋮
< microlens-0.4.13.1
< microlens-mtl-0.2.0.3
23d12
< pretty-1.1.3.6
30d18
⋮
```

- `<( ...)` process substitution

Implementation

Crash Course

Polymorphic Syntax Trees

-
  ```
  data Expr a = RealLit a !Double
              | Var a !Name
              | Lam a Name (Expr a)
              ⋮
  ```

Crash Course

Polymorphic Syntax Trees

-
  ```haskell
  data Expr a = RealLit a !Double
              | Var a !Name
              | Lam a Name (Expr a)
              ⋮
  ```
- `parse :: String -> Expr Loc`

Crash Course

Polymorphic Syntax Trees

-
  ```
  data Expr a = RealLit a !Double
              | Var a !Name
              | Lam a Name (Expr a)
              ⋮
  ```
- `parse :: String -> Expr Loc`
- Functorial!

Crash Course

Polymorphic Syntax Trees

-
  ```
  data Expr a = RealLit a !Double
              | Var a !Name
              | Lam a Name (Expr a)
              ⋮
  ```
- `parse :: String -> Expr Loc`
- Functorial!
  - `void :: Expr Loc -> Expr ()`

Crash Course

Polymorphic Syntax Trees

-
  ```
  data Expr a = RealLit a !Double
              | Var a !Name
              | Lam a Name (Expr a)
              ⋮
  ```
- `parse :: String -> Expr Loc`
- Functorial!
  - `void :: Expr Loc -> Expr ()`
  - `DeriveFunctor`

Crash Course

No Symbol Table

- Annotate AST with types

```
tyOf :: Expr Loc -> Either TyErr (Expr Type)
```

Crash Course

No Symbol Table

- Annotate AST with types

  ```
  tyOf :: Expr Loc -> Either TyErr (Expr Type)
  ```

-
  ```
  applySubstE :: Subst -> Expr Type -> Expr Type
  applySubstE s = fmap (applySubst s)
  ```

Crash Course

No Symbol Table

- Annotate AST with types

  ```
  tyOf :: Expr Loc -> Either TyErr (Expr Type)
  ```

-
  ```
  applySubstE :: Subst -> Expr Type -> Expr Type
  applySubstE s = fmap (applySubst s)
  ```

- GHC approach

Crash Course

No Symbol Table

- Annotate AST with types

  ```
  tyOf :: Expr Loc -> Either TyErr (Expr Type)
  ```

-
  ```
  applySubstE :: Subst -> Expr Type -> Expr Type
  applySubstE s = fmap (applySubst s)
  ```

- GHC approach
- Pattern match on type

Crash Course

No Symbol Table

- Annotate AST with types

  ```
  tyOf :: Expr Loc -> Either TyErr (Expr Type)
  ```

-
  ```
  applySubstE :: Subst -> Expr Type -> Expr Type
  applySubstE s = fmap (applySubst s)
  ```

- GHC approach
- Pattern match on type
  - `(Dedup (Int :~> _) _)`

Motivation

-
  ```
  fn count(x) :=
    fold (+) 0 ([:1"x);

  fn isEven() :=
    (~ /(0|2|4|6|8)$/);

  fn isOdd() :=
    (~ /(1|3|5|7|9)$/);

  let
    val even := count (filter isEven $0)
    val odd := count (filter isOdd $0)
    val total := odd + even
  in (total . even . odd) end
  ```

Motivation

-
  ```
  fn count(x) :=
    fold (+) 0 ([:1"x);

  fn isEven() :=
    (~ /(0|2|4|6|8)$/);

  fn isOdd() :=
    (~ /(1|3|5|7|9)$/);

  let
    val even := count (filter isEven $0)
    val odd := count (filter isOdd $0)
    val total := odd + even
  in (total . even . odd) end
  ```

-
  ```
  ~ % seq 1000 | ja run evenOdd.jac
  (1000 . 500 . 500)
  ```

Motivation

- Passes over input multiple times!

Motivation

- Passes over input multiple times!
- AWK does better:

```
/(0|2|4|6|8)$/ { even += 1 }

/(1|3|5|7|9)$/ { odd += 1 }

END { print even, odd, even + odd }
```

Motivation

Problem

- Less efficient

Motivation

Problem

- Less efficient
- ...but folds are nicer

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end
  - Two folds at once: `(.)`

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end
  - Two folds at once: `(.)`
  - No bytecode!

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end
  - Two folds at once: `(.)`
  - No bytecode!
- `type Env = Map Temp (Maybe Expr)`

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end
  - Two folds at once: `(.)`
  - No bytecode!
- `type Env = Map Temp (Maybe Expr)`
- Redefine the `Maybe Expr`s for each line

Motivation

Problem

- Less efficient
- ...but folds are nicer
- Rewrite many folds into something better
  - `Env -> Env`
  - Read from `Env` at end
  - Two folds at once: `(.)`
  - No bytecode!
- `type Env = Map Temp (Maybe Expr)`
- Redefine the `Maybe Exprs` for each line
- `Maybe Expr` for filter

Compiler Machinery

- Consider `fold (+) 0 xs`

Compiler Machinery

- Consider `fold (+) 0 xs`
    1. Place `0` in the Env

Compiler Machinery

- Consider `fold (+) 0 xs`
  1. Place `0` in the Env
  2. Turn `+` into something that updates the Env

Compiler Machinery

- Consider `fold (+) 0 xs`
    1. Place `0` in the Env
    2. Turn + into something that updates the Env
    3. Read from the Env

Compiler Machinery

- Consider `fold (+) 0 xs`
  1. Place `0` in the `Env`
  2. Turn `+` into something that updates the `Env`
  3. Read from the `Env`
  4. Multiple folds: apply all `Env` updates

Compiler Machinery

- Consider `fold (+) 0 xs`
  1. Place `0` in the Env
  2. Turn `+` into something that updates the Env
  3. Read from the Env
  4. Multiple folds: apply all Env updates
- What about `(+)`?

Compiler Machinery

- Consider `fold (+) 0 xs`
    1. Place `0` in the Env
    2. Turn `+` into something that updates the Env
    3. Read from the Env
    4. Multiple folds: apply all Env updates
- What about `(+)`?
    1. η-expand `(+)` to `\x.\y. x+y`

Compiler Machinery

- Consider `fold (+) 0 xs`
  1. Place `0` in the Env
  2. Turn + into something that updates the Env
  3. Read from the Env
  4. Multiple folds: apply all Env updates
- What about `(+)`?
  1. η-expand `(+)` to `\x.\y. x+y`
  2. Associate `x` with some temporary in Env

Compiler Machinery

- Consider `fold (+) 0 xs`
  1. Place `0` in the Env
  2. Turn `+` into something that updates the Env
  3. Read from the Env
  4. Multiple folds: apply all Env updates
- What about `(+)`?
  1. η-expand `(+)` to `\x.\y. x+y`
  2. Associate `x` with some temporary in Env
  3. Replace all `x` in scope with reads from Env at temporary

Compiler Machinery II

- Map—read from Env at given temporary, write to some other temporary

Compiler Machinery II

- Map—read from `Env` at given temporary, write to some other temporary
- Filter—read from `Env` at given temp, write `Nothing` if need be

Compiler Machinery II

- Map—read from `Env` at given temporary, write to some other temporary
- Filter—read from `Env` at given temp, write `Nothing` if need be
- Deduplicate—read at given temp, write to another temp if value hasn't been seen before

Compiler Machinery II

- Map—read from `Env` at given temporary, write to some other temporary
- Filter—read from `Env` at given temp, write `Nothing` if need be
- Deduplicate—read at given temp, write to another temp if value hasn't been seen before
- `$0` write line contents to given temp every time

Compiler Machinery II

- Map—read from `Env` at given temporary, write to some other temporary
- Filter—read from `Env` at given temp, write `Nothing` if need be
- Deduplicate—read at given temp, write to another temp if value hasn't been seen before
- `$0` write line contents to given temp every time
- All stream functions built-in (take unary, binary, ternary)

Compiler Machinery III

Stitching streams

- Consider `filter (='0') $0`

Compiler Machinery III

Stitching streams

- Consider `filter (='0') $0`
- `$0` writes to some `Temp`

Compiler Machinery III

Stitching streams

- Consider `filter (='0') $0`
- `$0` writes to some `Temp`
- `filter (='0')` reads from that `Temp`

Compiler Machinery III

Stitching streams

- Consider `filter (='0') $0`
- `$0` writes to some `Temp`
- `filter (='0')` reads from that `Temp`
- `filter` is an update `Env -> Env`, compose after `$0`

Questions?