Apollo

manager Reza Nayebi
architect Ben Kogan
language Javier Llaca
integrator Souren Papazian
validator Roberto Amorim
But you might know this...

\[ \lambda x . ( +1 ) x \]
...and maybe a bit of this...
What happens if...
...we do this?

\[ \lambda x \cdot (\text{\textmusicalnote})x \]
What is Apollo?

- **Functional** language for music creation
- **Simple** to use and understand
- Intended for **musicians** and **non-musicians**
What is Apollo?

• **Functional** language for music creation

• **Simple** to use and understand

• Intended for **musicians and non-musicians**

Let’s look at Apollo in action
Example

pitches: [Pitch] = [C4, E4, G4, C5, G4, E4, C4]
rhythm: [Duration] = uniform(\4, 7)
arpeggio: [Atom] = zip(pitches, rhythm)
main: Music = [arpeggio]
Data Types

• Int, Pitch, Duration

• Atom

• List

• Music
Data Types

note: Atom = (A4, \4)

chord: Atom = ([A4, C#5, E5], \4)

lead: [Atom] = [note, note]

back: [Atom] = [chord, chord]

song: Music = [lead, back]
Data Types

\( x: \text{Int} = 3 \)

\( a: \text{Pitch} = \text{A4} \) \hfill \text{--- \textit{A in Octave 4 (69)}}

\( q: \text{Duration} = \text{\(\frac{1}{4}\)} \) \hfill \text{--- \textit{Quarter Note (16)}}

\( a\text{Major: [Pitch]} = [\text{A4, C\#4, E4}] \)
Functions

square: (n: Int) -> Int = n * n

fac: (n: Int) -> Int =

  case (n == 0)
    1
  otherwise
    n * fac(n - 1)
Functions

Higher-order

\[ g: (f: (\text{Int}) \rightarrow \text{Int}, x: \text{Int}) \rightarrow \text{Int} = f(f(x)) \]

Typed lambda expressions

\[ \lambda x: \text{Int}, y: \text{Int} \rightarrow \text{Int} = x + y \]
Example Revisited

pitches: [Pitch] = [C4, E4, G4, C5, G4, E4, C4]
rhythm: [Duration] = uniform(\4, 7)
arpeggio: [Atom] = zip(chord, rhythm)
main: Music = [arpeggio]
Example Revisited

main: \textbf{Music} = [zip(

    [C4, E4, G4, C5, G4, E4, C4],

    uniform(\4, 7)])]
Interpreter Architecture

source ➔ Lexer ➔ tokens ➔ Parser ➔ AST ➔ Type-checker

updates expression environment

Evaluator ➔ Export ➔ result

HCodecs library

toAst t src >>> execAst env >>> \result —>

handleExport env ofile "main" >> return result
Extendable Architecture
typecheck :: Env Type -> Expr -> IOThrowsError Type

```haskell

 VInt{} -> return TInt
 VBool{} -> return TBool

 Neg e -> do
 t <- typecheck env e
 if t == TInt
 then return TInt
 else throwError (TypeUMismatch "-" t)

 Head l -> do
 tl <- typecheck env l
 case tl of
  (TList t) -> return t
  _ -> throwError (TypeUMismatch "@" tl)

 BoolOp op a b -> do
 ta <- typecheck env a
 tb <- typecheck env b
 case (ta, tb) of
  (TBool, TBool) -> return TBool
  _ -> throwError (TypeMismatch (show op) ta tb)

 Block body ret -> do
 env' <- clone env
 mapM_ (typecheck env') body
 typecheck env' ret

 FnCall (Name name) args -> do
 TFunc tps tr <- getVar env name
 checkFn env (name, tps, tr) args

 FnCall (VTLam tps tr _ _) args ->
 checkFn env ("<lambda>", tps, tr) args
```

```haskell

```
Def "fac" (Int) -> Int
VLam

["n"]
If

CompOp VInt IntOp

== Name 0 Name FnCall

"n" "n" Name List

"fac" IntOp

"n" - Name VInt

1
Def

"fac" (Int) -> Int

VLam

["n"]

If

CompOp

VInt

IntOp

==

Name

0

Name

FnCall

"fac"

List

IntOp

-

Name

VInt

"n"

1
```plaintext
Def "fac" (Int) -> Int
VLam ["n"]
If CompOp Name "n" VInt 0 IntOp Name "n" FnCall Name "fac" List IntOp "fac" IntOp - Name "n" VInt 1
```
Def "fac" (Int) -> Int

VLam ["n"]

If CompOp VInt IntOp

== Name 0 Name FnCall

"n"  "n" Name List

"fac" IntOp - Int VInt 1
Def "fac" (Int) -> Int
VLam
["n"]
If
 CompOp  VInt  IntOp
== Name  0 Name  FnCall
  "n"  "n" Name  List
    "fac" IntOp
        - Int  Int
Def "fac" (Int) -> Int

VLam ["n"]

If CompOp VInt ["n"]

IntOp CompOp VInt 0

IntOp Name "n"

FnCall Name ["n"]

List "fac" Int
```
Def

"fac" (Int) -> Int
VLam

["n"] If

CompOp VInt IntOp

== Name 0 Name FnCall

"n" "n" (Int) -> Int (Int)
```

```
| "fac" | (Int) -> Int |
| "n"   | Int         |
```
Def

"fac" (Int) -> Int

VLam

["n"] (Int)

<table>
<thead>
<tr>
<th>&quot;fac&quot;</th>
<th>(Int) -&gt; Int</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;n&quot;</td>
<td>Int</td>
</tr>
<tr>
<td>“fac”</td>
<td>(Int) -&gt; Int</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>“n”</td>
<td>Int</td>
</tr>
</tbody>
</table>
An interesting case
An interesting case

case (!a)
  []
otherwise
  f(h@a) :: mapII(f, t@a)
Enabling recursion

• Initialize name before storing its value

• Closures:

    Def name _ (VLam p b)
    => (Function p b env')

• ...and recursive closures!
Software Development Environment

- UNIX
- GitHub
- Haskell
- Cabal
Runtime

• REPL

• Output MIDI

• Prelude
Prelude

concat: (a: [Int], b: [Int]) -> [Int] =

  case (!a)

    b

  otherwise

    h@a :: concat(t@a, b)
filter: (f: (Int) -> Bool, a: [Int]) -> [Int] =

case (!a)

    []

  case (f(h@a))

    h@a :: filter(f, t@a)

otherwise

    filter(f, t@a)
Prelude

sort: (a: [Int]) -> [Int] =

    case (!a)
        []
    otherwise {
        concat(concat(sort(a), [p]), sort(b))
        where
            p: Int = h@a
            a: [Int] = filter(\x: Int -> Bool: x <= p, t@a)
            b: [Int] = filter(\x: Int -> Bool: x > p, t@a)
    }
Project Management
Project Management

• Weekly team meetings

• Git workflow: branch and pull request, wait for validation from other team members before merging

• Travis CI and unit testing helped a lot to catch small errors.
Project Management

- Project came together nicely at the end
- A lot of the hard work in the architecture that allowed us to add all features easily

Git commit history
Testing & Validation
Apollo repl, version 0.0.1.0: https://github.com/apollo-lang/apollo

Commands:
  :browse See all current bindings and their types
  :export <name> Export a name of type Music to `_repl.mid`
  :quit Exit the repl

apollo> []
apollo --repl

Commands:
:browse See all current bindings and their types
:export <name> Export a name of type Music to `_repl.mid`
:quit Exit the repl

apollo> ❯
Testing and Validation

- 20+ code files for testing features, errors
- Bash script tests, validates, and `diff`'s errors
- Test files all at once or line-by-line
- Travis Continuous Integration
~P/apollo master: make test