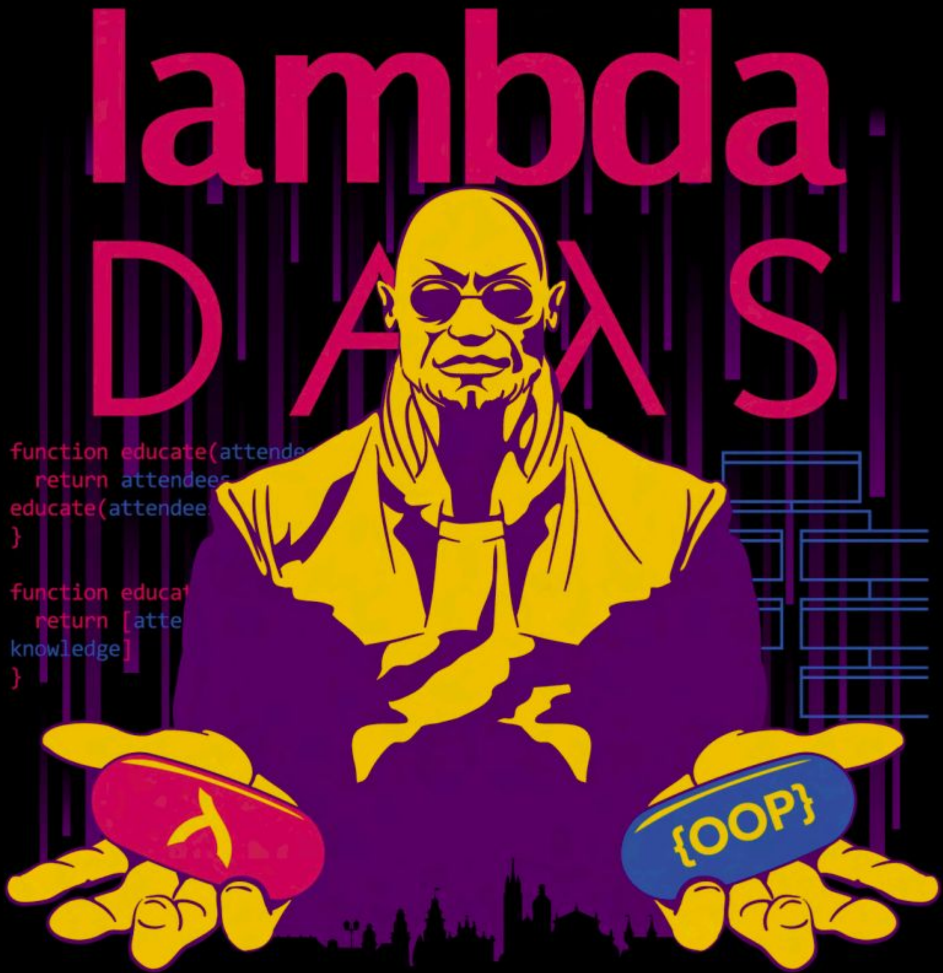


Andor Penzes

lambda
DAλS
27-28 MAY 2024
KRAKÓW | POLAND

lambda
DAλS
27-28 MAY 2024
KRAKÓW | POLAND

Imagine a dependently typed Python

Andor Penzes
jww Thorsten Altenkirch

FP
{OOP}
Did you just take both pills?
imgflip.com

Project status

- Design
- Prototyping
- Discussions
- Call for contributions

github / andorp / DepPy

Dependent Types

- Types and values live in the same space
- Values are part of types
- Types can be inspected like values
- Computation is done at type-checking

[1] typing.readthedocs.io/en/latest/spec (Python)
[3] https://cseweb.ucsd.edu/~rchugh/research/nested/djs.pdf (JS)
[4] https://www.cs.nott.ac.uk/~psztxa/publ/lics01.pdf (Normalization)

Foundations

- We write programs to solve problems
- We organize information as data types
- We transform values of data types via functions
- Sometimes abstractions are lousy and need external help
- Sometimes we want to change the programs

Data representation

- Sum of products, generics, levitation
- Object Oriented Programming (Python)
  - Subclasses are the sums
  - Objects are the products
- FP (Haskell)
  - Data constructors are the sums
  - Fields are the products
- Python
  - Everything is a dict
  - Tagged union of everything

[1] typing.readthedocs.io/en/latest/spec (Python)
[3] https://cseweb.ucsd.edu/~rchugh/research/nested/djs.pdf (JS)
[6] https://personal.cis.strath.ac.uk/conor.mcbride/levitation.pdf (Levitation)

Compile time / Execution time?

- Compile time types ensure consistencies
- Runtime types
    - define representation
    - different interpretations of the same
- Executed tests ensures consistencies
- Assertions ensures consistencies

Nat and Fin

```
class Nat:

class Zero(Nat):

  def __init__(self):
    pass


class Succ(Nat):

  def __init__(self,n:Nat):
    self.n = n
```

```
class Fin:
  # n : Nat


class FZ(Fin):

  def __init__(self,n:Nat):
    self.n = n


class FS(Fin):

  def __init__(self,f:Fin):
    self.n = Succ(f.n)
    self.f = f
```

Vect

```
class Vect:
  # n : Nat
  def append(self,ys:Vect)
      -> Vect [n = self.n + ys.n]:
    pass

class Nil(Vect):
  def __init__(self):
    self.n = Zero()

  def append(self,ys:Vect):
    # ys
    # : Vect [n = ys.n]
    # : Vect [n = 0 + ys.n]
    # : Vect [n = self.n + ys.n]
    return ys
```

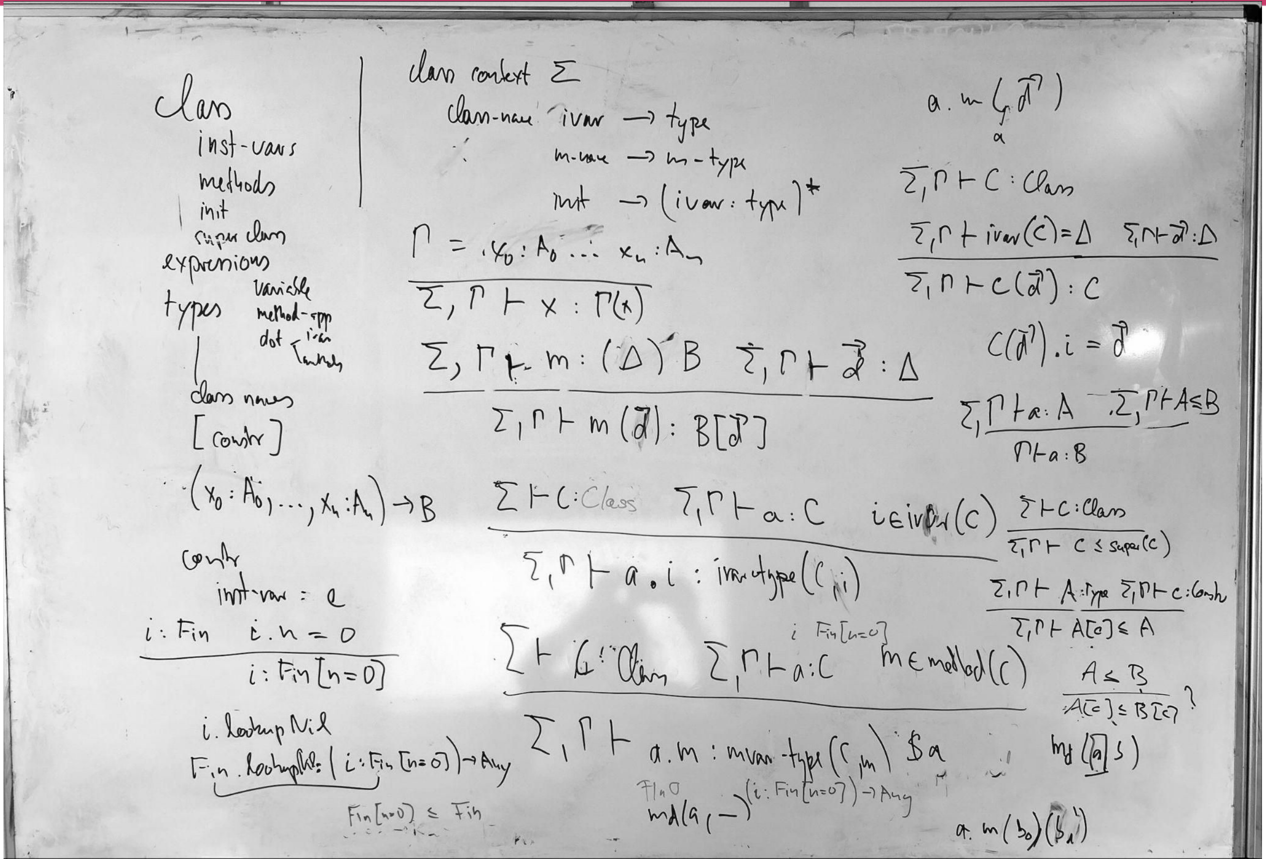
```
class Cons(Vect):

  def __init__(self,x,xs:Vect):
    self.n  = Succ(xs.n)
    self.x  = x
    self.xs = xs

  def append(self,ys:Vect):
    zs = self.xs.append(ys)
    # zs : Vect [n = self.xs.n + ys.n]
    ws = Cons(self.x,zs)
    # ws
    # : Vect [n = Succ (self.xs.n + ys.n)]
    # : Vect [n = Succ (self.xs.n) + ys.n]
    # : Vect [n = self.n + ys.n]
    return ws
```

Type checking by normalization

- Classes, objects and expressions
- Evaluation of closed expressions lead to objects
- Evaluation of open expressions lead to objects with partially applied expressions
- Intermediate form of expressions are good for debugging

[3] https://cseweb.ucsd.edu/~rchugh/research/nested/djs.pdf (JS)
[4] https://www.cs.nott.ac.uk/~psztxa/publ/lics01.pdf (Normalization)

Type System



class
- inst-vars
- methods
- init
- super class

expressions
- variable
- method-app
- dot
  - i-var
  - methods

types
- class names
- [constr]
- $(x_0 : A_0, \dots, x_n : A_n) \to B$

constr
- inst-var = e

$$\frac{i : \mathrm{Fin} \quad i.n = 0}{i : \mathrm{Fin}[n=0]}$$

i.lookup Nil

$\underbrace{\mathrm{Fin}.\mathrm{lookupNil}: (i : \mathrm{Fin}}\,[n=0]) \to \mathrm{Any}$

$\mathrm{Fin}[n=0] \leq \mathrm{Fin}$

class context Σ

class-name ivar → type

m-name → m-type

init → (ivar : type)*

$$\frac{\Gamma = x_0 : A_0 \dots x_n : A_n}{\Sigma, \Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Sigma, \Gamma \vdash m : (\Delta) B \quad \Sigma, \Gamma \vdash \vec{d} : \Delta}{\Sigma, \Gamma \vdash m(\vec{d}) : B[\vec{d}]}$$

$$\frac{\Sigma \vdash C : \mathrm{Class} \quad \Sigma, \Gamma \vdash a : C \quad i \in \mathrm{ivar}(C)}{\Sigma, \Gamma \vdash a.i : \mathrm{ivar\text{-}type}(C, i)}$$

i Fin[n=0]

$$\frac{\Sigma \vdash C : \mathrm{Class} \quad \Sigma, \Gamma \vdash a : C \quad m \in \mathrm{method}(C)}{\Sigma, \Gamma \vdash a.m : \mathrm{mvar\text{-}type}(C, m) \$ a}$$

$\overset{\mathrm{Fin}\,0}{mA(a, -)}^{(i : \mathrm{Fin}[n=0]) \to \mathrm{Any}}$

$a.m(\underset{a}{,}\vec{d})$

$$\frac{\Sigma, \Gamma \vdash C : \mathrm{Class} \quad \Sigma, \Gamma \vdash \mathrm{ivar}(C) = \Delta \quad \Sigma, \Gamma \vdash \vec{d} : \Delta}{\Sigma, \Gamma \vdash C(\vec{d}) : C}$$

$C(\vec{d}).i = \vec{d}$

$$\Sigma, \frac{\Gamma \vdash a : A \quad \Sigma, \Gamma \vdash A \leq B}{\Gamma \vdash a : B}$$

$$\frac{\Sigma \vdash C : \mathrm{Class}}{\Sigma, \Gamma \vdash C \leq \mathrm{super}(C)}$$

$$\frac{\Sigma, \Gamma \vdash A : \mathrm{Type} \quad \Sigma, \Gamma \vdash c : \mathrm{Constr}}{\Sigma, \Gamma \vdash A[c] \leq A}$$

$$\frac{A \leq B}{A[c] \leq B[c]} ?$$

$m_d(\boxed{a}\, s)$

$a.m(b_0)(b_1)$

Side effects

- Don't mention them
- Use Monads / Monad Transformers
- Built-in Effect Handlers

[1] typing.readthedocs.io/en/latest/spec (Python)
[2] www.unison-lang.org/docs/fundamentals/abilities/ (Unison)
[3] https://cseweb.ucsd.edu/~rchugh/research/nested/djs.pdf (JS)
[5] https://arxiv.org/abs/1611.09259v (Frank)

Thank you for your attention!

github / andorp / DepPy

LOOK INTO MY EYES
AND USE DEPENDENT TYPES
imgflip.com

References

[1] typing.readthedocs.io/en/latest/spec (Python)

[2] www.unison-lang.org/docs/fundamentals/abilities/ (Unison)

[3] https://cseweb.ucsd.edu/~rchugh/research/nested/djs.pdf (JS)

[4] https://www.cs.nott.ac.uk/~psztxa/publ/lics01.pdf (Normalization)

[5] https://arxiv.org/abs/1611.09259v (Frank)

[6] https://personal.cis.strath.ac.uk/conor.mcbride/levitation.pdf (Levitation)