Finite Functional Programming

via

Graded Effects & Relevance Types

Michael Arntzenius

UC Berkeley

HOPE 2025

but NOT dependent!

use FANCY TYPES

to deconstruct LOGIC PROGRAMMING

into FUNCTIONAL PROGRAMMING

MOTIVATION

FP ♡ RELATIONAL MODEL

FP:

- higher order abstraction
- modular programming

RELATIONAL MODEL:

- declarative

  separate WHAT am I doing to my data
  from HOW am I representing it

- efficient

toward "DATALOG with HIGHER ORDER ABSTRACTION and MODULES"

cf RelationalAI's "Rel" language

LOGIC

```
follows(ada, bob).
follows(ada, charlie).
follows(bob, ada).
...

mutuals(X, Y) ←
    follows(X, Y),
    follows(Y, X).
```

LOGIC

```
follows(ada, bob).
follows(ada, charlie).
follows(bob, ada).
...

mutuals(X, Y) ←
    follows(X, Y),
    follows(Y, X).
```

ENUMERATES INPUTS THAT YIELD TRUE

$\stackrel{?}{=}$

FUNCTIONAL

```
follows : person → person → bool
follows x y =
    (x == "ada" and y == "bob")
 or (x == "ada" and y == "charlie")
 or (x == "bob" and y == "ada")
 or ...

mutuals : person → person → bool
mutuals x y =
        follows x y
    and follows y x
```

INPUT ⟶ OUTPUT BLACK BOX

LOGIC

```
directedActor(Dir, Actor) ←
    directedFilm(Dir, Film),
    actedIn(Actor, Film).
```

LOGIC

```
directedActor(Dir, Actor) ←
    directedFilm(Dir, Film),
    actedIn(Actor, Film).
```

FUNCTIONAL

```
directedFilm : person → film → bool
actedIn      : person → film → bool

directedActor : person → person → bool
directedActor dir actor =
```

??

LOGIC	FUNCTIONAL
	directedFilm : person → film → bool actedIn : person → film → bool
directedActor(Dir, Actor) ← directedFilm(Dir, Film), actedIn(Actor, Film).	directedActor : person → person → bool directedActor dir actor = exists (λfilm. directedFilm dir film and actedIn actor film)

exists : (film → bool) → bool ??

enumerate inputs that yield true?

we need FUNCTIONS

that can ENUMERATE THEIR SUPPORT !

IN THIS TALK: FINITELY SUPPORTED functions

IMPLEMENTED AS TABLES,

$input_1$	$output_1$
$input_2$	$output_2$
⋮	⋮

What is SUPPORT?

sets A, B

pointed sets P, Q are sets equipped with a "default value",

$nil_P \in P$

eg. $nil_{bool} = false$

$$\mathrm{support}\,(f : A \to P) \stackrel{def}{=} \{ a \in A : f(a) \neq nil_P \}$$

$$A \xrightarrow{fs} P \stackrel{def}{=} \{ f \in A \to P : \mathrm{support}(f) \text{ is finite} \}$$

$$nil_{A \xrightarrow{fs} P} \stackrel{def}{=} \lambda x.\, nil_P$$

WHY BOTHER WITH POINTED SETS?

(isn't bool enough to do logic programming?)

1. TYPES for OR and AND

2. CURRYING, $A \xrightarrow{fs} B \xrightarrow{fs} P$

↑ IN THIS TALK

↓ NOT IN THIS TALK

3. AGGREGATIONS

4. WEIGHTED LOGIC PROGRAMMING
   (eg. semirings — lots of related work here in past 10 yrs)

THE GOAL

a language that LOOKS functional
& supports "procedural" functions (incl. higher-order)
BUT ALSO has a type of "FINITE FUNCTIONS"
which are

1. GUARANTEED finitely supported
(using types)

2. REPRESENTED as TABLES

3. COMPUTED like DB QUERIES
(with join algorithms)

EXAMPLES

```
costarred : person ->fs person ->fs bool
costarred x y =
    exists (λfilm. starredIn x film and starredIn y film)

-- # of films an actor has starred in
filmography : person ->fs ℕ
filmography actor = sum (λfilm. when (actedIn actor film) 1)

-- STRETCH GOAL: recursion!
path : person ->fs person ->fs bool
path x y =
     costarred x y
  or exists (λz. costarred x z and path z y)
```

INSIGHT

$T_A P \overset{\text{def}}{=} A \xrightarrow{fs} P$ is a graded co/monad!

monad $\longrightarrow$

pure / extract : $P \cong T_1 P$ ie. $P \cong 1 \xrightarrow[fs]{} P$

return / dup : $T_A T_B P \cong T_{A \times B} P$ ie. $A \xrightarrow[fs]{} B \xrightarrow[fs]{} P \cong A \times B \xrightarrow[fs]{} P$

$\longleftarrow$ comonad

but what is map?

✗ $\mathrm{map} : (P \to Q) \to (A \xrightarrow[fs]{} P) \to (A \xrightarrow[fs]{} Q)$

eg. $\mathrm{map}\,(\lambda\_.\,\mathrm{true})\ \tau = \underline{\lambda\_.\,\mathrm{true}}$

NOT FINITELY SUPPORTED

IN $\mathrm{map}\ f\ \tau$,

f MUST PRESERVE nil!

$$P \multimap Q \stackrel{\mathrm{def}}{=} \{ f \in P \to Q : f(\mathrm{nil}_P) = \mathrm{nil}_Q \}$$

$$\mathrm{nil}_{P \multimap Q} \stackrel{\mathrm{def}}{=} \mathrm{const}\ \mathrm{nil}_Q$$

✓ $\mathrm{map} : (P \multimap Q) \multimap (A \xrightarrow[fs]{} P) \multimap (A \xrightarrow[fs]{} Q)$

A, B sets $\quad P, Q$ pointed sets $\quad nil_P \in P$

$A \to B$ functions

$P \multimap Q$ nil-preserving fns

} procedures

$A \xrightarrow[fs]{} P$ finitely supported maps — tables

✓ $\lambda x.x \; : \; P \multimap P$

✗ $\lambda x.\, \text{true} \; : \; P \multimap \text{bool}$

✓ $\lambda x.\, \text{foo } x \; \underline{\text{and}} \; \text{bar } x \; : \; P \multimap \text{bool}$

with $\text{foo}, \text{bar} : P \multimap \text{bool}$

~~LINEAR~~ exactly once $\quad$ ~~AFFINE~~ at most once

RELEVANT at <u>least</u> once

$$P \,\&\, Q \overset{def}{=} P \times Q$$

$$nil_{P \& Q} \overset{def}{=} (nil_P, nil_Q)$$

$$or : bool \,\&\, bool \multimap bool$$

BC $or(false, false) = false$

IE

$$or(nil_{bool}, nil_{bool}) = nil_{bool}$$

"DIRECT PRODUCT"

$$P \otimes Q \overset{def}{=} P \times Q \text{ modulo}$$

$$nil_{P \otimes Q} \overset{def}{=} (nil_P, y) = (x, nil_Q)$$

$$and : bool \otimes bool \multimap bool$$

BC
$and(false, \_) = false$
$and(\_, false) = false$

IE

$and(nil, \_) = nil$
$and(\_, nil) = nil$

"SMASH PRODUCT"
or
TENSOR PRODUCT

$\Gamma ::= \cdot \mid \Gamma, x : A$ set context

$\Delta ::= \cdot \mid \Delta, x : P$ pointed (nil-preserving) context

$\Gamma / \Delta \vdash t : P$ glosses as $\Gamma \to \Delta \multimap P$

$$\frac{\Gamma / \Delta \vdash t : P \qquad \Gamma / \Delta \vdash u : Q}{\Gamma / \Delta \vdash (t, u) : P \mathbin{\&} Q}$$

nil if BOTH t,u are

$$\frac{\Gamma / \Delta_1 \vdash t : P \qquad \Gamma / \Delta_2 \vdash u : Q}{\Gamma / \Delta_1, \cup \Delta_2 \vdash (t, u) : P \otimes Q}$$

AT LEAST ONCE!

nil if EITHER t,u is

$\Gamma ::= \cdot \mid \Gamma, x : A$ set context

$\Delta ::= \cdot \mid \Delta, x : P$ pointed (nil-preserving) context

$\Omega ::= \cdot \mid \Omega, x : A$ finitely supported context

~~$\Gamma / \Delta \vdash t : P$ glosses as $\Gamma \to \Delta \multimap P$~~

$\Gamma / \Delta / \Omega \vdash t : P$ glosses as $\Gamma \to \Delta \multimap \Omega \xrightarrow[fs]{} P$

$\gamma \in [\![\Gamma]\!] \quad \delta \in [\![\Delta]\!]$

$$\frac{\Gamma / \Delta / \Omega \vdash t : P \qquad \Gamma / \Delta / \Omega \vdash u : Q}{\Gamma / \Delta / \Omega \vdash (t, u) : P \,\&\, Q}$$

$$\frac{\mathrm{support}([\![t]\!](\gamma)(\delta)) = \sigma_1 \qquad \mathrm{support}([\![u]\!](\gamma)(\delta)) = \sigma_2}{\mathrm{support}([\![(t,u)]\!](\gamma)(\delta)) = \sigma_1 \cup \sigma_2}$$

nil if BOTH t, u are $\longleftrightarrow$ non-nil if EITHER t, u is

MONOIDAL: $T_A P \,\&\, T_A Q \cong T_A (P \,\&\, Q)$

GIVEN $f, g : \mathbb{N} \xrightarrow[fs]{} \text{bool}$, WHICH ARE WELL-TYPED?

$\lambda x. \lambda y.\ f\,x \ \underline{\text{and}}\ g\,y$:
$\mathbb{N} \xrightarrow[fs]{} \mathbb{N} \xrightarrow[fs]{} \text{bool}$

$\lambda x.\ f\,x \ \underline{\text{and}}\ g\,x$:
$\mathbb{N} \xrightarrow[fs]{} \text{bool}$

$\lambda x.\ f\,x \ \underline{\text{and}}\ x > 17$:
$\mathbb{N} \xrightarrow[fs]{} \text{bool}$

GIVEN $f, g : \mathbb{N} \xrightarrow[fs]{} \mathrm{bool}$, WHICH ARE WELL-TYPED?

$\lambda x. \lambda y. f\,x \ \underline{\text{and}}\ g\,y$: $\mathbb{N} \xrightarrow[fs]{} \mathbb{N} \xrightarrow[fs]{} \mathrm{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1, \Omega_2 \vdash (t, u) : P \otimes Q}$$

$\lambda x. f\,x \ \underline{\text{and}}\ g\,x$: $\mathbb{N} \xrightarrow[fs]{} \mathrm{bool}$

$$T_{\Omega_1} P \otimes T_{\Omega_2} Q \multimap T_{\Omega_1, \Omega_2} (P \otimes Q)$$

"graded monoidal" ???

support = CROSS PRODUCT of support of t, u

$\lambda x. f\,x \ \underline{\text{and}}\ x > 17$: $\mathbb{N} \xrightarrow[fs]{} \mathrm{bool}$

GIVEN $f, g : \mathbb{N} \xrightarrow[fs]{} \text{bool}$, WHICH ARE WELL-TYPED?

$\lambda x. \lambda y.\ f\,x\ \underline{\text{and}}\ g\,y :$
$\mathbb{N} \xrightarrow[fs]{} \mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1, \Omega_2 \vdash (t, u) : P \otimes Q}$$

$\lambda x.\ f\,x\ \underline{\text{and}}\ g\,x :$
$\mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1 \cup \Omega_2 \vdash (t, u) : P \otimes Q}$$

$\lambda x.\ f\,x\ \underline{\text{and}}\ x > 17 :$
$\mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$T_A P \otimes T_A Q \multimap T_A (P \otimes Q)$$

T_A lax monoidal

Support is INTERSECTION of support of t, u

GIVEN $f, g : \mathbb{N} \xrightarrow[fs]{} \text{bool}$, WHICH ARE WELL-TYPED?

$\lambda x. \lambda y. f\, x \;\underline{\text{and}}\; g\, y : \mathbb{N} \xrightarrow[fs]{} \mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1, \Omega_2 \vdash (t, u) : P \otimes Q}$$

$\lambda x. f\, x \;\underline{\text{and}}\; g\, x : \mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1 \cup \Omega_2 \vdash (t, u) : P \otimes Q}$$

$\lambda x. f\, x \;\underline{\text{and}}\; x > 17 : \mathbb{N} \xrightarrow[fs]{} \text{bool}$

$$\frac{\Gamma / \Delta_1 / \Omega_1 \vdash t : P \qquad \Gamma, \Omega_1 / \Delta_2 / \Omega_2 \vdash u : Q}{\Gamma / \Delta_1 \cup \Delta_2 / \Omega_1, \Omega_2 \vdash (t, u) : P \otimes Q}$$

WEIRD but CORRECT

ARE F·S VARS JUST FUNKY RELEVANCE TYPES?

FUNCTION	PRESERVES NIL?	FINITE SUPPORT?
$\lambda x. x$	✓	
$\lambda x.\ \text{true}$	✗	
$\lambda x.\ \text{foo}\ x\ \underline{\text{and}}\ \text{bar}\ x$	✓	

(assume foo, bar ARE nil-preserving / fin. supp, respectively)

ARE F·S VARS JUST FUNKY RELEVANCE TYPES?

FUNCTION	PRESERVES NIL?	FINITE SUPPORT?
$\lambda x. x$	✓	✗ *
$\lambda x.$ true	✗	✗
$\lambda x.$ foo x <u>and</u> bar x	✓	✓

(assume foo, bar ARE nil-preserving / fin. supp, respectively)

*NOT EVEN WELL KINDED!
F·S MAPS ARE $A \xrightarrow[fs]{} P$ NOT $P \xrightarrow[fs]{} Q$!

HOW TO MAKE & USE F·S VARS?

$$\frac{\Gamma / \Delta / \Omega, x : A \vdash t : P}{\Gamma / \Delta / \Omega \vdash \lambda x . t : A \xrightarrow[fs]{} P} \xrightarrow[fs]{} \mathrm{I}$$

$$\frac{\Gamma / \Delta / \Omega \vdash t : A \xrightarrow[fs]{} P}{\Gamma / \Delta / \Omega, x : A \vdash t\, x : P} \xrightarrow[fs]{} \mathrm{E}$$

ALSO THE VARIABLE USE RULE!

what about $(t \; x \; x)$

where $t : A \xrightarrow[fs]{} A \xrightarrow[fs]{} P$?

HOW TO MAKE & USE F·S VARS?

$$\frac{\Gamma / \Delta / \Omega, x : A \vdash t : P}{\Gamma / \Delta / \Omega \vdash \lambda x . t : A \xrightarrow[fs]{} P} \xrightarrow[fs]{} I$$

$$\frac{\Gamma / \Delta / \Omega \vdash t : A \xrightarrow[fs]{} P}{\Gamma / \Delta / \Omega \cup (x : A) \vdash t\ x : P} \xrightarrow[fs]{} E$$ — ALSO THE VARIABLE USE RULE!

what about $(t \times x)$

where $t : A \xrightarrow[fs]{} A \xrightarrow[fs]{} P$? ✓

HOW TO MAKE & USE F.S MAPS?

$\text{eq} : A \to A \xrightarrow{fs} \text{bool}$

$\text{or} : \text{bool} \,\&\, \text{bool} \multimap \text{bool}$

$\text{and} : \text{bool} \otimes \text{bool} \multimap \text{bool}$

$\text{single} : A \to P \multimap A \xrightarrow{fs} P$

$\text{when} : \text{bool} \otimes P \multimap P$

$\text{exists} : (A \xrightarrow[fs]{} \text{bool}) \multimap \text{bool}$

$\text{sum} : (A \xrightarrow[fs]{} \mathbb{N}_0) \multimap \mathbb{N}_0$

$\text{min} : (A \xrightarrow[fs]{} \mathbb{N}_\infty) \multimap \mathbb{N}_\infty$

TYPECLASS!

```
class Additive P where
    add : P & P ⊸ P          -- LAWS { add(x, nil) = x
                                     + COMMUTES
                                     + ASSOCIATES
    sum : (A →fs P) ⊸ P      -- derivable from `add`
                                BUT NOT DEFINABLY ☹
```

λ: the ULTIMATE RELATION!

→ RELATIONS = FUNCTIONS + SUPPORT

→ FINITE support = DATALOG.
ENUMERABLE " = PROLOG ??

→ naturally handles { WEIGHTED logic programming
AGGREGATIONS

→ type system is UGLY but I love it ♡

GRADED (CO) MONAD
+ ADJUNCTION
+ RELEVANT TYPES
+ HAX

AND MORE...

the Maybe COMONAD

recursion ??

FUTURE WORK

the DECIDABLE typeclass
& user-defined functions

user-defined
pointed types?

IMPLEMENTATION ?!
(come to my miniKanren
workshop talk!
FRIDAY 14:37

non-graded return/join
=
Prolog/unrestricted
logic prog ?