Frex: staged-optimisation and equational-proof-synthesis using universal algebra

Ohad Kammar
University of Edinburgh

Working Group 2.11: Program Generation
International Federation for Information Processing
Technical Committee 2: Software: Theory and Practice

21^{st} Meeting
15–18 August, 2022
University of Southern Denmark, Odense


supported by:
THE ROYAL SOCIETY
The Alan Turing Institute
Facebook Research
NCSC

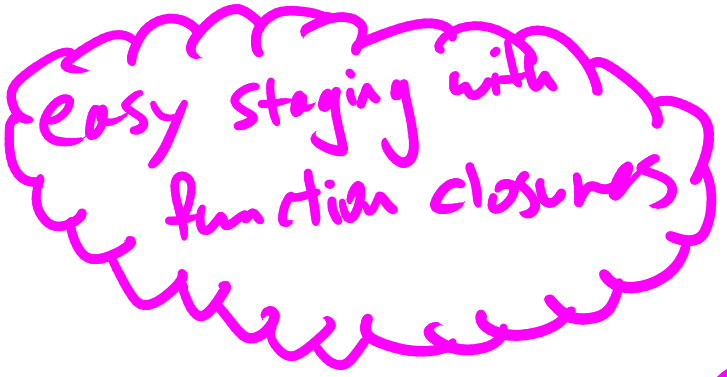
Open-Terms Modulo equations

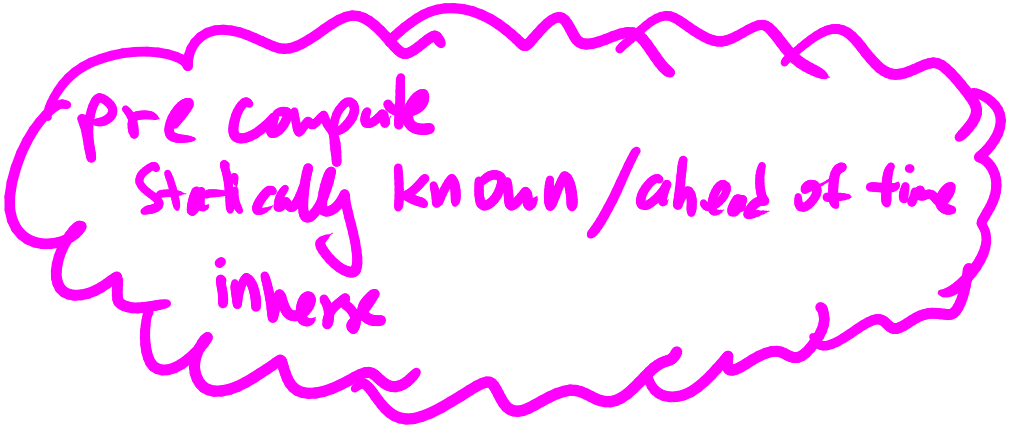
```
A : Matrix 4 4 Double  --NB: Matrix row col a
A = MkMatrix           --       = Vect col (Vect row a)
  [[ 3, 2, 0, 3]
  ,[ 4, 3, 0, 0]
  ,[ 0, 0, 6, 5]
  ,[ 0, 0, 7, 6]]
```

spec — easy staging — motivate this talk

```
compute, thoughtful, clever
  : (dataset : Matrix 3 300 Double) -> Vect 4 Double
compute
  = foldl
      (\acc,[x,y,z] =>
          head $ --NB: result : Matrix 4 1 Double
          [acc] |+| A .inv * [[x, 2 + y, -x, z + y]])
      [0,0,0,0]
```

$$\sum_{v \in dataset} A^{-1} \cdot \begin{pmatrix} v_x \\ 2+v_y \\ -v_x \\ v_z+v_y \end{pmatrix}$$






```
thoughtful =
  let B := A .inv
  in foldl
       (\acc,[x,y,z] =>
           head $
           [acc] |+| B * [[x, 2 + y, -x, z + y]])
       [0,0,0,0]
```

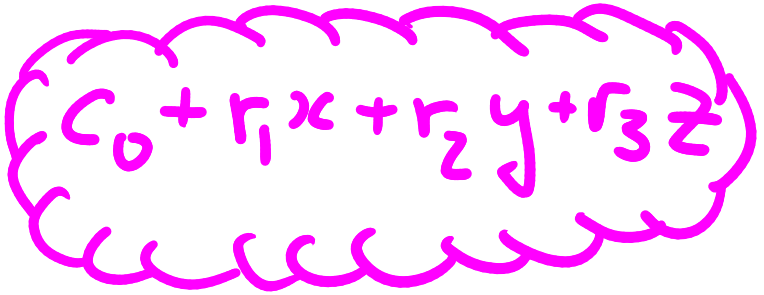
NB: very thoughtful: use more linear algebra (tangential)

Partially Static Representation

```
Frex a n = (a, Vect n Double)


  (c, [r1, r2, r3]) : Frex a 3


clever =
  let B := A .inv
      precalculate := head $ B * [[x, 2 + y, -x, z + y]]
  in foldl
      (\acc,[x,y,z] =>
          acc |+| eval precalculate [x,y,z])
      [0,0,0,0]
```



Why does it work?

Operations

$0, (+), (r \cdot)$ $(r \in \mathbb{R})$ $\Rightarrow$ algebraic signature Σ

Semantic equations

$r \cdot (s \cdot x) = (r \cdot s) \cdot x$ $x + y = y + x$

$\Rightarrow$ algebraic presentation Ax

Thm: $(\mathbb{R}, \mathrm{Vect}\ n\ \mathbb{R}) \cong \mathrm{Term}_{\Sigma}(\mathrm{Fin}\ n \uplus \mathbb{R}) / +\ Ax\ Eval$

variables / dynamic values

static values

semtic axioms

$\frac{1}{2} \cdot 4 = 2$

This Construction

$$\mathrm{Term}_{\sum}(\mathrm{Fin}\, n \uplus R)/_{+}^{\mathrm{Ax}_{\mathrm{Eval}}}$$

has a name:

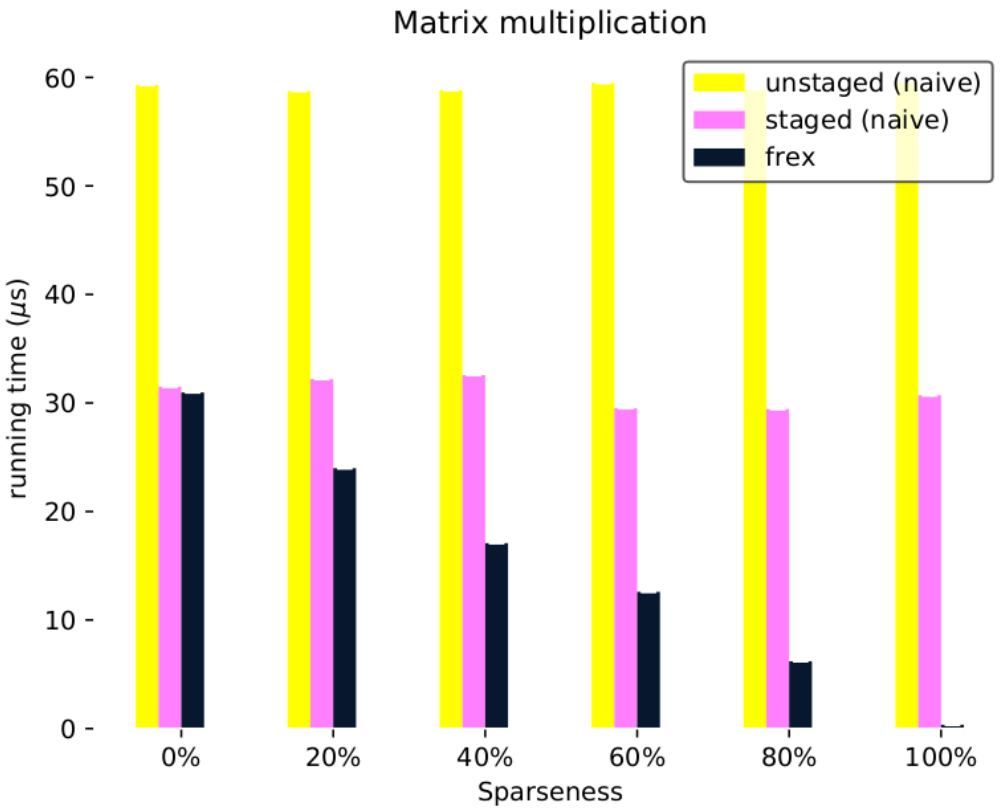
Free Extention

(Free)

Staging with Frex

[Yallop, von Glehn, Kammar '18]

- OCaml & Haskell libs implementing frexlets for:
  (commutative) monoids, rings, distributive lattices
  Abelian groups
- use frexlets for
  Staged Optimization



This talk: ongoing work

joint with:

- [x] Introduction
- [ ] Frex tutorial
- [ ] equational-proof synthesis
- [ ] modularity
- [ ] Normalisation by Evaluation (NbE)
- [ ] Generalised algebraic theories

Yallop

van Glabbeek

Allais

Brady

Corbyn

Lindley

Valliappan

Algebras

$(\mathbb{M}_{n\times n}(\mathbb{N}), I_n, (\cdot)) \models$ monoids

$(\mathbb{N}, 0, (+)), (\mathbb{N}, 1, (+)) \models$ commutative monoids

$(\mathbb{N}, \max, 0) \models$ semi-lattices

$\frac{\bar{\bar{x}} = x}{\bar{x}\bar{y} = \bar{y}\bar{x}}$

$(\text{String}, "", (+), \text{reverse}) \models$ involutive monoids

$(\mathbb{R}, 0, (+), (r\cdot)\ (r \in \mathbb{R})) \models$ distributive $\mathbb{R}$-action on a commutative monoid

lower the barrier for bespoke algebraic structures

Universal algebra vocabulary

- Signatures $\Sigma \ni (op : n)$ (arity n, operation symbol op)

  e.g.: $0, 1 : 0$, $(+), (\cdot), \max, (\#) : 2$, $(-)^{-1}, \overline{(-)} : 1$

  $\rightsquigarrow \mathrm{Term}_\Sigma X \ni X \vdash t$

  $\{x, y\} \vdash x + y$, $\overline{x \cdot y}$, $(x \cdot x)^{-1}$

- Presentations: (Σ, Ax)

  where $Ax \subseteq \sum_{X \in \mathbb{N}} (\mathrm{Term}_\Sigma X)^2$ (axioms)

  e.g.: $x \cdot 1 = x$, $x + y = y + x$

- Algebra $A = (\underline{A}, [\![op : n]\!] : \underline{A}^n \to \underline{A})$ (carrier, operation)

  $A \models Ax$ e.g.: previous slide

Free algebra over $X \in \mathbf{Set}$:

$$F_{Ax}\,X := \mathrm{Term}_{\Sigma}\,X \,/\, Ax^{\vdash}$$

+ operations

Representation Theorem:

$$F_{\mathrm{monoid}}\,X \cong (\mathrm{List}\,X, [\,], (+\!+))$$

Need:
homomorphisms/categories
& universal properties/adjunctions

not today.
come talk to me!

Free extension of $A \models Ax$ by $X \in \mathrm{Set}$:

$$A[X] := \mathrm{Term}_{\Sigma}(\underline{A} \uplus X) / Ax + \mathrm{Eval}$$

Eval: $\mathrm{Left}(a_1) + \mathrm{Left}(a_2) = \mathrm{Left}(a_1 +_A a_2)$

(operation symbol; operation in A)

Repr. Thm:

$\forall A \models$ com. monoids

$$A[\mathrm{Fin}\, n] \cong ((\underline{A}, \mathrm{Vec}\, n\, \mathbb{N}), \text{pointwise operations})$$

This talk: ongoing work

- [x] Introduction
- [x] Frex tutorial
- [ ] equational-proof synthesis
- [ ] modularity
- [ ] Normalisation by Evaluation (NbE)
- [ ] Generalised algebraic theories

Dependently typed Programming

Beauty:

guarantee length is correct

```
(++) : Vect n a -> Vect m a -> Vect (n+m) a
[]        ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

identical code to simply-typed list concatenation.

Beast

```
mergeBy : (isLT: a -> a -> Bool) ->
  Vect n a -> Vect m a -> Vect (n + m) a
mergeBy     _ [] ys = ys
mergeBy {n} _ xs [] = rewrite plusZeroRightNeutral n in
                      xs
  where
    plusZeroRightNeutral : (n : Nat) ->
      n + 0 = n
mergeBy {n = S n} {m = S m}
        isLT xs'@(x :: xs) ys'@(y :: ys) =
  if isLT x y
  then x :: mergeBy isLT xs ys'
  else rewrite sym (plusSuccRightSucc n m) in
       y :: mergeBy isLT xs' ys
  where
    plusSuccRightSucc : (n,m : Nat) ->
      S (n + m) = n + S m
```

innocuous looking type

necessary rewriting steps

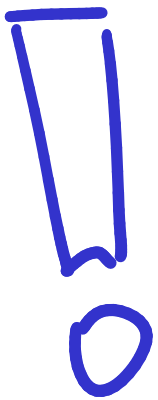
explicit implicit arg

prove auxiliary lemmata

Need equational proofs to type-check

⇝ rote simplification steps:

$$x + (1 + y) = (1 + x) + y$$



Same problem

but type-checker

needs to see proofs

Formalize:

congruence wrt each $[\![\mathrm{OP}]\!]$

$(\underline{A}, [\![-]\!], \equiv) \models Ax$

- UP's of $F_{Ax} X$, $A[X]$ in Agda + Idris w.r.t. Setoids
- Representation Thms for Concrete presentations

Example: binary representations indexed by their semantics

[Brady, Hammond, McKinna '07]

```
|~ ((b_s + b_s) + c_s) + (val_a + val_a + val_b + val_b)
~~ c_s + 2*((val_a + val_b) + b_s)
-> ...(Frex.solve 4 (Frex Nat.Additive) $
     let b_s   = Dyn 0
         c_s   = Dyn 1
         val_a = Dyn 2
         val_b = Dyn 3
     in  ((b_s :+: b_s) :+: c_s)
         :+: (((val_a :+: val_a) :+: val_b) :+: val_b)
     =-= c_s :+: the Nat 2 *:
          (((val_a :+: val_b) :+: b_s)))
```

direct manipulation of proof terms

call frex simplifier

future work: avoid lexicons repetition

- Solve also extracts

Canonical
type isomorphisms

e.g.:

$$(a, b, c) \cong ((c, a), b)$$

(thanks to Setoid usage)

Proof Extraction with setoids

$(A[X], (=)) \cong (Term_{\Sigma}(\underline{A} \uplus X), \vdash\!\equiv)$

setoid isomorphism

free representation with propositional/decidable equality

deeply embedded representation of provability in equational logic

$x + (1 + x) \stackrel{?}{=} 2 \cdot x + 1$

eval in $\mathbb{N}[x]$

$(1, [2]) = (1, [2])$

$\vdash x + (1 + x) \equiv 2 \cdot x + 1$

apply UP w.r.t. term setoid

Process deeply embeded proofs ($\vdash x+(1+x) \equiv 2x+1$)

- Simplify
  (remove "loops")
- Print proofs ↝
- Certify proofs ↙

```
units : (x : U m) -> 01 .+. (x .+. 01) .+. 01 =~= x
units x = CalcWith (cast m) $
  |~ 01 .+. (x .+. 01) .+. 01
  ~~ 01 .+. (01 .+. x .+. 01) .+. 01
    ..<( Cong (\ focus => 02 :+: (focus :+: 02) :+: 02) $ lftNeutrality x )
  ~~ 01 .+. (01 .+. (x .+. 01)) .+. 01
    ..<( Cong (\ focus => 02 :+: focus :+: 02) $ associativity 01 x 01 )
  ~~ 01 .+. 01 .+. (x .+. 01) .+. 01
    ...( Cong (\ focus => focus :+: 02) $ associativity 01 01 (x .+. 01) )
  ~~ 01 .+. 01 .+. x .+. 01 .+. 01
    ...( Cong (\ focus => focus :+: 02) $ associativity (01 .+. 01) x 01 )
  ~~ 01 .+. x .+. 01 .+. 01
    ...( Cong (\ focus => focus :+: Val x :+: 02 :+: 02) $ lftNeutrality 01 )
  ~~ 01 .+. x .+. (01 .+. 01)
    ..<( associativity (01 .+. x) 01 01 )
  ~~ 01 .+. x .+. 01
    ...( Cong (\ focus => 02 :+: Val x :+: focus) $ lftNeutrality 01 )
  ~~ 01 .+. x
    ...( rgtNeutrality (01 .+. x) )
  ~~ x
    ...( lftNeutrality x )
```

Fig. 17. Frex-certificate for the of $0 + (x + 0) + 0 = x$ in a generic monoid `m`

$$(x \bullet 3) \bullet [2] \overset{\uparrow \text{Right neutrality}}{=} (x \bullet 3) \bullet 2 \bullet [\varepsilon] \overset{\downarrow \text{Left neutrality}}{=} (x \bullet [3]) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Right neutrality}}{=} (x \bullet 3 \bullet [\varepsilon]) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Left neutrality}}{=} ([x] \bullet 3 \bullet \varepsilon \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Right neutrality}}{=}$$

$$([x \bullet \varepsilon] \bullet 3 \bullet \varepsilon \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Right neutrality}}{=} ([(x \bullet \varepsilon) \bullet \varepsilon] \bullet 3 \bullet \varepsilon \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Left neutrality}}{=} (([\varepsilon] \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet 3 \bullet \varepsilon \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Evaluate}}{=}$$

$$[(0 \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet 3 \bullet \varepsilon \bullet \varepsilon] \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Associativity}}{=} (0 \bullet [((x \bullet \varepsilon) \bullet \varepsilon) \bullet 3 \bullet \varepsilon \bullet \varepsilon]) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Associativity}}{=} (0 \bullet [((x \bullet \varepsilon) \bullet \varepsilon) \bullet 3] \bullet \varepsilon \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Commutativity}}{=}$$

$$(0 \bullet [(3 \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet \varepsilon \bullet \varepsilon]) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Associativity}}{=} [0 \bullet 3 \bullet ((x \bullet \varepsilon) \bullet \varepsilon) \bullet \varepsilon \bullet \varepsilon] \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Associativity}}{=} ((0 \bullet 3) \bullet ((x \bullet \varepsilon) \bullet \varepsilon) \bullet [\varepsilon \bullet \varepsilon]) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Left neutrality}}{=}$$

$$((0 \bullet 3) \bullet [(x \bullet \varepsilon) \bullet \varepsilon] \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Associativity}}{=} ((0 \bullet 3) \bullet (x \bullet [\varepsilon \bullet \varepsilon]) \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Left neutrality}}{=} ([0 \bullet 3] \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\uparrow \text{Evaluate}}{=}$$

$$(3 \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Associativity}}{=} 3 \bullet [((x \bullet \varepsilon) \bullet \varepsilon) \bullet 2 \bullet \varepsilon \bullet \varepsilon] \overset{\uparrow \text{Associativity}}{=} 3 \bullet [((x \bullet \varepsilon) \bullet \varepsilon) \bullet 2] \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Commutativity}}{=}$$

$$3 \bullet [(2 \bullet (x \bullet \varepsilon) \bullet \varepsilon) \bullet \varepsilon \bullet \varepsilon] \overset{\uparrow \text{Associativity}}{=} 3 \bullet 2 \bullet ((x \bullet \varepsilon) \bullet \varepsilon) \bullet \varepsilon \bullet \varepsilon \overset{\downarrow \text{Associativity}}{=} (3 \bullet 2) \bullet ((x \bullet \varepsilon) \bullet \varepsilon) \bullet [\varepsilon \bullet \varepsilon] \overset{\uparrow \text{Left neutrality}}{=}$$

$$(3 \bullet 2) \bullet [(x \bullet \varepsilon) \bullet \varepsilon] \bullet \varepsilon \overset{\downarrow \text{Associativity}}{=} (3 \bullet 2) \bullet (x \bullet [\varepsilon \bullet \varepsilon]) \bullet \varepsilon \overset{\uparrow \text{Left neutrality}}{=} [3 \bullet 2] \bullet (x \bullet \varepsilon) \bullet \varepsilon \overset{\downarrow \text{Evaluate}}{=} 5 \bullet [(x \bullet \varepsilon) \bullet \varepsilon] \overset{\uparrow \text{Right neutrality}}{=} 5 \bullet [x \bullet \varepsilon] \overset{\downarrow \text{Right neutrality}}{=} 5 \bullet x$$

Fig. 16. Frex-extracted proof of $(x \bullet 3) \bullet 2 = 5 \bullet x$ in the additive monoid over `Nat`

This talk: ongoing work

- [x] Introduction
- [x] Frex tutorial
- [x] equational-proof synthesis
- [ ] modularity
- [ ] Normalisation by Evaluation (NbE)
- [ ] Generalised algebraic theories

Modularity

Example: Reuse monoid free for involutive monoids

$(\underline{A}, 1, (\cdot), \overline{(-)}) \vDash$ involutive monoid

↪ $\overline{x \cdot y} = \bar{y} \cdot \bar{x}$ $\quad \overline{\bar{x}} = x$ e.g. string reversal

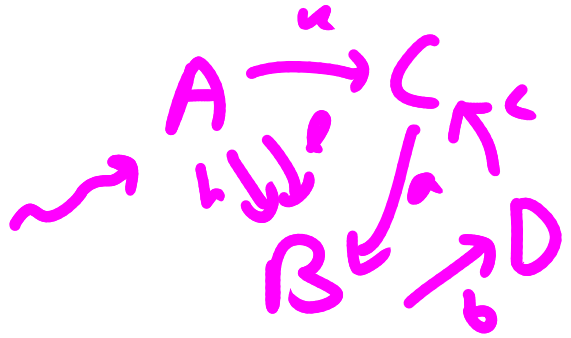
then $(\underline{A}, 1, (\cdot), \overline{(-)})[X] \cong (\underline{A}, 1, (\cdot))[(\mathrm{Bool}, X)]$

modularity is exceptional

celebrate it

Multi-free of homomorphism graphs

G : algebra labelled vertices
homomorphism labelled edges



⇝ multi-sorted algebra $M \models Ax(G)$ → (A, B, C, D, u, a, b, c) ops

↳ each sort is an algebra

Repr. Thm:

$$M[\vec{X}]_V \cong M_V\left[\sum_{u \in G_V} Path(u,v) \times X_v\right]$$

each function homomorphic

V-sort in the extension by $\vec{X} = (X_v)_{v \in G.V}$

↳ sum of algebra labeling V

Application: Free VP.

Distributive tensor of theories (WIP)

Example: $((\cdot) \hookrightarrow \text{Monoid}) \triangleright \text{Commutative monoid} = \text{non-Commutative semiring}$

$((+) \hookrightarrow \text{Commutative monoid}) \triangleright \text{Abelian Group} = \text{Commutative ring}$

$((\cdot)^{-1} \hookrightarrow \text{involutive monoid}) \triangleright \text{Commutative semigroup} = \text{involutive non-zero semi-rings}$

etc.

$$\left\{\begin{array}{l}\text{Ordinary,}\\ \text{involutive}\end{array}\right\} \times \left\{\begin{array}{l}\text{ordinary}\\ \text{Commutative}\end{array}\right\} \times \left\{\begin{array}{l}\text{semiring}\\ \text{monoid}\\ \text{group}\end{array}\right\}^2 = 36$$

more, in fact, since involution can be reversing or not.

Challenge: generalise sparse Horner normal form.

This talk: ongoing work

- [x] Introduction
- [x] Frex tutorial
- [x] equational-proof synthesis
- [x] modularity
- [ ] Normalisation by Evaluation (NbE)
- [ ] Generalised algebraic theories

Normalisation-by-Evaluation (NbE)

Open terms modulo semtics
for higher-order functions

$u \in \mathrm{Neutral}_{\Gamma}\,\tau \longrightarrow n \in \mathrm{Normal}_{\Gamma}\,\tau$

neutral forms typically given uniformly:

$u = f_{\vec{\tau}\to\tau}\; n_{\tau_1} \cdots n_{\tau_k}$

NbE with free $(A \vDash Ax)$

$\tau ::= \cdots \mid \lfloor A \rfloor$ → base type with algebraic structure

$$\frac{c \in \underline{A}}{\Gamma \vdash \underline{c} : \lfloor A \rfloor} \qquad \frac{(op : n) \in \Sigma}{\Gamma \vdash \underline{op} : \lfloor A \rfloor^n \to \lfloor A \rfloor}$$

NbE algorithm:

$$Neutral_{\Gamma}\lfloor A \rfloor \hookrightarrow A[Neutral|_{\Gamma}{}^{\lfloor A \rfloor}] =: Nanf_{\Gamma}\lfloor A \rfloor$$

Stable under variations (e.g. monadic metalanguage)

This talk: ongoing work

- [x] Introduction
- [x] Frex tutorial
- [x] equational-proof synthesis
- [x] modularity
- [x] Normalisation by Evaluation (NbE)
- [ ] Generalised algebraic theories

Generalised algebraic theories (GATs) [Cartmell '78]

- multi-sorted algebraic theory for sorts
  - "kinds"

eg.: Categories Kind = { Obj , morphisms }

Kind operation: $Hom : Obj^2 \to morphisms$

No equations

$\rightsquigarrow$ dependent contexts eg. $x, y : Obj, f, g : Hom(x, y)$

- operation symbols sorted by sort normal forms

eg.: $x : Obj \vdash id_x : Hom(x, x)$

$x, y, z : Obj \vdash (\circ) : Hom(y, z) \times Hom(x, y) \to Hom(x, z)$

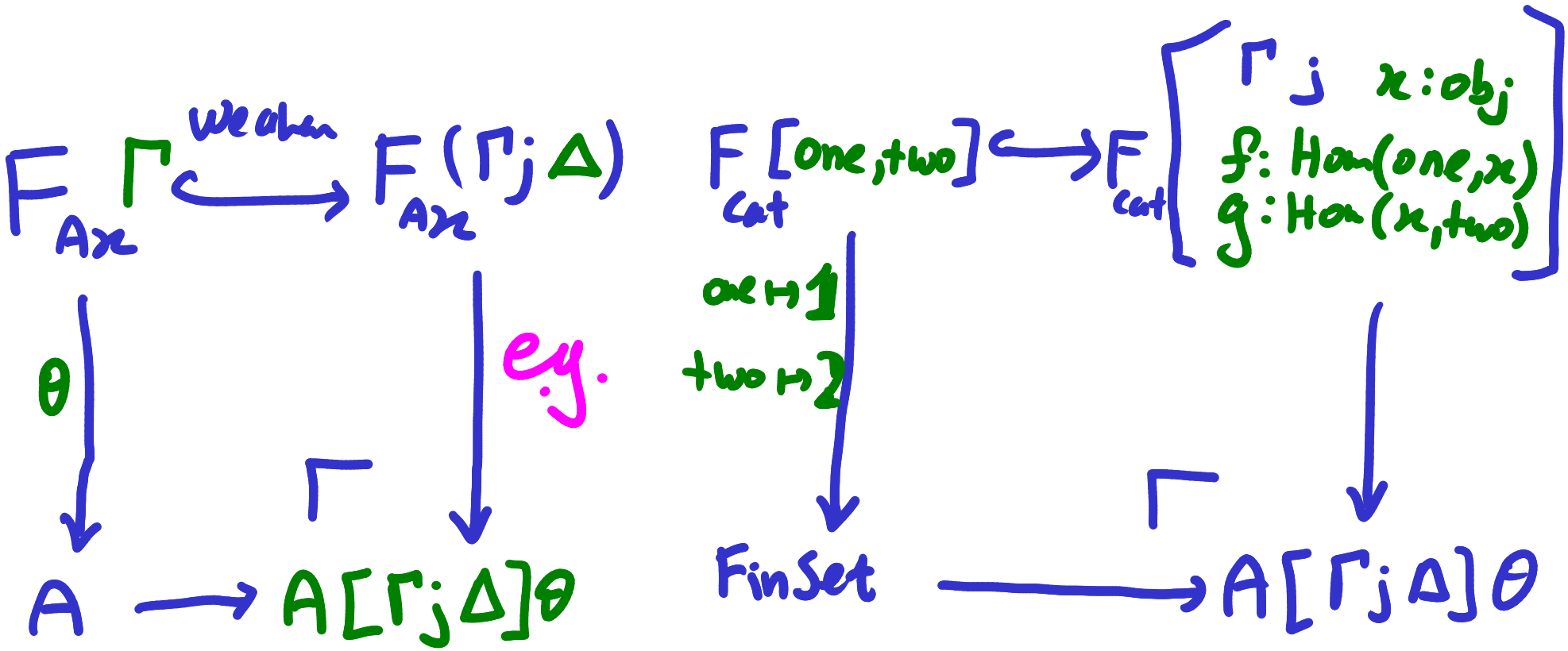
- Equations

eg. $x, y : Obj, f : Hom(x, y) \vdash f \circ id_x = f : Hom(x, y)$

GAT Free

extend by a telescope

eg: $\mathrm{FinSet}[x:\mathrm{obj},\ f:1\to x,\ g:x\to 2]$



NbE as Free

- [x] categories
- [ ] Cartesian categories
- [ ] Cartesian closed categories (CCC,
- [ ] CCC + primitive algebraic structure
- [ ] CCC + primitive higher-order structure (e.g. map/reduce)

This talk: ongoing work

joint with:

- ☑ Introduction
- ☑ Frex tutorial
- ☑ equational-proof synthesis
- ☑ modularity
- ☑ Normalisation by Evaluation (NbE)
- ☑ Generalised algebraic theories


Yallop


van Glabbeek


Allais


Brady


Corbyn


Lindley


Valliappan