Normalisation in PL & Logic: What?

Want to identify equivalent syntactic representation

Examples

Programs: $x := 1; x := 2$ vs $x := 2$ vs if $y \geq 3$ then $x := 2$ else $x := 2$

Expressions: $5 + x + (0 + (-6))$ vs $-1 + x$

λ-terms: $(\lambda x.\, x+1)(y-3)$ vs $-2 + y$ [not today]

Normalisation: why?

- Proofs can deal with fewer cases
- robust systems (compilers, partial evaluators) behave the same for equivalent programs.
- Search for programs over smaller search space (Synthesis)

Blind Rewriting (BRew)

Tempting:

- represent AST
- apply all transformations
- everywhere
- at once

But:

- BRew is wasteful in time & space (NB: E-graphs)
- BRew is chaotic / non-robust
- BRew is limited to syntactic representations (E.g.: x+y vs y+x)

Don't confuse BRew with Strategic Rewriting & Term Rewriting:

- deep maths & time-tested properties: confluence, strong normalisation
  Knuth-Bendix completion
- deep connections to Foundations of Logic, Mathematics & Computation.

What you may not know is the connections between

Normalisation and Modern algebra

Structure

1) (My) Goals

2) Universal Algebra basics

3) Modern algebra: homomorphism & representation thms

Not today:

~~4)~~ Free extensions & partial evaluation

~~5)~~ second-order algebra & higher-order functions

$\Longrightarrow$ more at my SPLV course!

My Goals (or: why should you care?)

1) Motivate you to learn algebra → formally
   → independently

2) raise awareness: so you one day recognize you're dealing with normalisation & turn to algebraic tools (or me!)

3) raise tolerance: when you become:

- project manager;
- system architect;
- professor; or
- policy maker

} and one of your {

- programmers;
- engineers;
- students; or
- advisors

suggests normalisation/algebra, you will pay attention.

Universal algebra basics

template: sets $(X_s)_{s \in Sort}$ $\mathcal{A}$ operations $\left(f_i : X_{i_1} \times \cdots \times X_{i_n} \longrightarrow X_{k_i}\right)_{i \in I}$

instances:

integer + addition $\mathbb{Z}$ $(Sort = \{*\})$ $\mathcal{A}$ $(+) : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$ $0 : \mathbb{1} := \{*\} \longrightarrow \mathbb{Z}$

matrix algebra $(M_{m \times n})_{m,n \in \mathbb{N}}$ $\mathcal{A}$ $(+) : M_{m \times n} \times M_{m \times n} \longrightarrow M_{m \times n}$

$(\cdot) : M_{m \times n} \times M_{n \times k} \longrightarrow M_{m \times k}$

Combinatory logic $\Lambda := \text{closed } \lambda\text{-terms}/\equiv$ $\mathcal{A}$ $(@) : \Lambda \times \Lambda \longrightarrow \Lambda$

$I := (\lambda x.\, x)$, $K := (\lambda x, y.\, x)$, $S := (\lambda x, y, z.\, x\, z\, (y\, z))$ $: \mathbb{1} \longrightarrow \Lambda$

$\vdots$

Today, I'll stick to $\mathbb{Z}_2(\mathcal{M}_{mon})$ for concreteness.
The other examples carry over without fuss.

Universal Algebra:

Study common properties of all such structures
(so universal results)

Def: an algebraic signature $S = (\text{Sort}_S, \text{op}_S, \text{arity}_S)$:

set of sorts — set of operators — function assigning arity

$\text{Sort}_S \ni s$ $\quad \text{op}_S \ni f$ $\quad \text{arity} : \text{op} \to (\text{List Sort}) \times \text{Sort}$

When $\text{arity}_S f = ([s_1, \ldots, s_n], s)$ we write $(f : s_1 \times \cdots \times s_n \to s) \in S$

Ex: $\text{Sort}_{\text{monoidStructure}} = \{*\}$ $\quad \text{op}_{\text{monoidStructure}} := \{(\cdot), 1\}$ $\quad \text{arity}(\cdot) := ([*, *], *)$ $\quad \text{arity}\, 1 := ([], *)$

& we'll write $(\cdot) : * \times * \to *$ $\quad 1 : \mathbb{1} \to *$

$\text{Sort}_{\text{trans}} = \{ *_{m \times n} \mid m, n \in \mathbb{N} \}$

$(+) : *_{m\times n} \times *_{m\times n} \to *_{m\times n}$

$(\cdot) : *_{m\times n} \times *_{n\times k} \to *_{m\times k}$

$I_n : \mathbb{1} \to *_{n\times n}$ $\quad \underline{0} : \mathbb{1} \to *_{m\times n}$

etc.

Def: Algebra for signature Σ:

$$A = \left((A_s)_{s \in Sorts_\Sigma}, (f_A : A_{s_1} \times \dots \times A_{s_n} \to A_s)_{(f : s_1 \times \dots \times s_n \to s) \in \Sigma} \right)$$

↑ carrier sets, indexed by sorts

↑ operations, indexed by operators

Ex monoid structures are algebras for signature $monoidStruct$:

$A_* := \underline{A} := \mathbb{Z}$ $\quad (\cdot)_A := (+)$ $\quad 1_A := 0$

matrices are algebras for signature trans:

$A_{*_{m \times n}} := M_{m \times n} :=$ m rows n colums tables of real numbers

$(+)_A := (+)$ $\quad 0 := \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$

$(\cdot)_A := (\cdot)$ $\quad I_n := \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$

as usual.

typically only consider algebras validating some equations. $\square_{10}$

Modern algebra & representation

Def: the sets of **terms** for a signature Σ over a sort_Σ-indexed family of variables $(X_s)_{s \in \Sigma}$:

$$\frac{}{x \in (\mathrm{Term}_\Sigma X)_s}\,(x \in X_s) \qquad \frac{t_i \in (\mathrm{Term}_\Sigma X)_{s_i} \text{ for } i = 1, \ldots, n}{f(t_1, \ldots, t_n) \in (\mathrm{Term}_\Sigma X)_s}\,(f : s_1 \times \cdots \times s_n \to s \in \Sigma)$$

$\mathrm{Term}_\Sigma X$ is the **data structure** we use to represent **abstract syntax trees** for expressions with operators in Σ and variables in X.

Modern algebra specifies this data structure as the "free Σ-algebra over the S-indexed set X"

$$F_\Sigma X \qquad \mathrm{Var} : X \longrightarrow \mathrm{Term}_\Sigma X =: F_\Sigma \underline{X}$$

Intuitively, we form elements of $\mathrm{Term}_\Sigma X$ by either:

1) using operators from Σ:

$$f : (F_\Sigma \underline{X})_{s_1} \times \cdots \times (F_\Sigma \underline{X})_{s_n} \longrightarrow (F_\Sigma \underline{X})_s$$

2) using variables

$$\mathrm{var} : X \longrightarrow F_\Sigma X$$

We can specify this "freeness" property by structure preserving maps

Def: A **homomorphism** $h : A \longrightarrow B$ between Σ-algebras

consists of:

$$(h_s : A_s \longrightarrow B_s)_{s \in \text{sorts}_\Sigma}$$

← sort-indexed family of functions between corresponding carriers

such that:

$$h_s(f_A(a_1, \ldots, a_n)) = f_B(h_{s_1} a_1, \ldots, h_{s_n} a_n)$$

for any $(f : s_1 \times \cdots \times s_n \to s) \in \Sigma$ and $a_i \in A_{s_i}$

Ex:

$(\lambda a.\, 2^a) : (\mathbb{Z}, +, 0) \longrightarrow (\mathbb{Q}, \cdot, 1)$ is a monoid sort homomorphism.

Thm (representation for free algebras)

$F_S X$ with $var : X \to F_S X$ is a free S-algebra over X:

for every S-algebra A and functions $(env_s : X_s \to A_s)_s$

there is a unique homomorphism $eval_{env} : F_S X \to A$

extending env along var:

$$eval_{env}\, x = env\, x \qquad eval_{env}(f(t_1, \dots, t_n)) = f_A(eval_{env}\, t_1, \dots, eval_{env}\, t_n)$$

Succinctly:
$$\frac{env : X \to A}{eval_{env} : F_S X \to A} \qquad eval_{var} = id \qquad eval_{env} \circ var = env$$

Ex $X := \{x, y, z\}$ $\quad A := (\mathbb{Z}, (+), 0)$

$env : X \to \mathbb{Z}$ $\quad x \mapsto 10 \quad y \mapsto 5 \quad z \mapsto -6$

$eval_{env}((y \cdot x) \cdot (1 \cdot z)) = (5 + 10) + (0 - 6) = 9$

$\square$

We can now phrase the main takeaway:

Core idea

a representation theorem provides a normalisation procedure

We'll unpack this for monoids.

Thm (Representation for monoids)

The monoid $F_{monoid} X := (List\, X, (\#), [])$ equipped with

$$var : x \longmapsto [x]$$

is the free monoid over X.

The unique monoid homomorphism to a monoid M equipped with $env : X \longrightarrow \underline{M}$:

$$eval_{env}\, [x_1, \ldots, x_n] := env\, x_1 \cdot (\ldots \cdot (env\, x_n \cdot 1_M) \ldots)$$

Succinctly:

$$\frac{env : X \longrightarrow \underline{M}}{eval_{env} : F_{monoid}\, X \to M}$$

Back to normalisation:

Given two expressions $t, s \in Term_{\text{monoid sig}}{}^{X}$

e.g.: $(x \cdot y) \cdot (1 \cdot z)$ vs $1(x \cdot (y \cdot z))$

evaluate it in the free monoid as a monoid structure

Thm: $\underset{\text{var}}{\text{eval}}\, t = \text{eval}_{\text{var}}\, s$ in $F_{\text{monoid}}{}^{X}$

$\Updownarrow$

$t = s$ can be proved from monoid axioms.