Bauer 2

Signature $\Sigma = \{(op_i, n_i)\}_{i \in I}$

Terms $x_1, \dots, x_n \mid t$

Equations $x_1, \dots, x_n \mid l = r$

Theory $T = (\Sigma_T, \mathcal{E}_T)$

Interpretation I — carrier set; for each op say w/ fn on carrier ~~set~~ supposed to be

Model M

Free model $\mathrm{Free}_T(X)$

---

(4) Generalize arities & parameters

operation symbol op_i, n_i

$$[\![op_i]\!]_I : \underbrace{|I| \times \dots \times |I|}_{n_i} \longrightarrow |I|$$

$|I|^{n_i}$ (another way to write)

$[n] := \{0, 1, \dots, n-1\}$

$$\underbrace{X \times \dots \times X}_{n} \cong X^{[n]}$$

B^A set of all functions $A \to B$

Scanned with CamScanner

Arity = set (first idea)

Signature (temporary)

$$\Sigma = \{ (op_i, A_i) \}_{i \in I}$$

"symbols" sets

$$[\![op_i]\!]_I : |I|^{A_i} \to |I|$$

(arguments given to me as a function instead of a tuple)

(eg. A_i Powerset of Powerset of $\mathbb{R}$)

not $\times$ $op_i(\dots t_n \dots)_{n \in A_i}$ $\times$

but : $op_i(K)$, more like $op_i(\lambda a. t_n)$

Kappa

Parameters.

Example Modules Vector space (special kind of module)

V. (over $\mathbb{R}$)

$v + w$ addition

0

$-v$ (opposite)

$s \cdot v$ $s \in \mathbb{R}$ scalar } scalar multiplication

$s \cdot v$ $v \in V$

$\cdot : \mathbb{R} \times V \to V$

unary (one V) but w/ a parameter

want all ops to look same

add unit parameter to non-param ops

eg.)

$+ : 1 \times V \times V \to V$

$[\![op]\!]_I : \underset{\text{parameter}}{P} \times |I|^{\overset{\text{(arity)}}{A}} \to |I|$

Signature $\Sigma = \{ \underset{\text{symbol}}{op_i} : \underset{\substack{\text{set}\\\text{parameter}}}{P_i} \rightsquigarrow \underset{\substack{\text{set}\\\text{arity}}}{A_i} \}_{i \in I}$

Trees (instead of terms)

$\mathrm{Tree}_\Sigma(x)$:

- $(\text{return } x) \in \mathrm{Tree}_\Sigma(X)$ for $x \in X$
- $op_i(p, \kappa) \in \mathrm{Tree}_\Sigma(x)$ for $p \in P_i$, $\kappa : A_i \to \mathrm{Tree}_\Sigma(x)$

read as inductive definition

Equation $X \mid l = r$ where $l, r \in \mathrm{Tree}_\Sigma(X)$

Theory $T = (\Sigma_T, \mathcal{E}_T)$

Interpretation I:

- carrier set $|I|$
- for each $op_i : P_i \rightsquigarrow A_i$, give

$$[\![op_i]\!] : P_i \times |I|^{A_i} \to |I|$$

extend to interpretation of trees / terms

For $t \in \mathrm{Tree}_\Sigma(X)$,

recall $[\![x_1 \dots x_k | t |]\!] : |I|^k \to |I|$
$\quad\quad\quad\quad\quad X$

$[\![t_i]\!]_I : |I|^X \to |I|$

$[\![\text{return } x]\!]_I (\gamma) = \gamma(x) \quad \text{for } x \in X$

$[\![op_i(p, \kappa)]\!]_I (\gamma) = [\![op_i]\!]_I (p, \lambda a \in A_i. [\![\kappa(a)]\!]_I (\gamma))$

Model as before (Interpret. which satisfies equations)

$\mathrm{Free}_T(X) = \mathrm{Tree}_\Sigma(X)/_{\approx_T}$

where $\approx_T$ is least congruence enforcing

equations $\mathcal{E}_T$

(5) Computational effects as algebraic operations

stuff not in λ calc print, disk, memory

(distinguish progs from math)

computations:

- pure return v (canonical example)
(means terminating)
- effectful
(non-termination is most interesting effect)

$op(p, \kappa)$ → the rest of computation awaiting result of operation
parameter

K is like a function awaiting result of computation

Example $\ell{+}{+}$

$\text{lookup}(\ell, \lambda x.\ \text{update}((\ell, x+1), \lambda\_.\ \text{return } x))$

$\text{print}(\text{"Hello world!"}, \lambda\_.\ \text{return } ())$

order
~~outer~~ operation happens first

once value is known, then activate the computation

signatures

set of location

$\text{lookup} : L \rightsquigarrow S$

set of states

$\text{update} : L \times S \rightsquigarrow 1$

$\text{print} : \text{String} \rightsquigarrow 1$

Scanned with CamScanner

Example: State holding elements of a set S

signature

$$put : S \rightsquigarrow \underline{1}$$
$$get : 1 \rightsquigarrow S$$

$$get((), \lambda x. get((), \lambda y. \kappa(x,y))) = get((), \lambda z. \kappa(z,z))$$

$$get((), \lambda x. put(x, \kappa)) = \kappa()$$

$$put(s, get((), \kappa)) = put(s, \lambda\_. \kappa(s))$$
$$put(s, \lambda\_. put(t,\kappa)) = put(t, \kappa)$$

4 equations bec. 4 ways of combining get, put

non-algebraic for Power, et.
- continuations

- exception handler

Example exceptions

$$abort : \underline{1} \rightsquigarrow \emptyset$$

$$[\![abort]\!]_M : 1 \times |M|^{\emptyset} \to |M|$$

non-resumable
because $\emptyset$ is empty

but we can also handle an exception
which did not fit in this algebraic theory
Plotkin et al - handlers

Which model is what you want ; going on.

If all equations are what you want then free model is the one you want.

If not then you need more / diff equations

The free model $Free_T(V)$ is the set of computations with effects described by theory T and returning values from set V

$Free_T(V)$

effects (T)

return values (V)

$Free_{State}(Int)$

computations that can use State and return integers

$Free_{State}(V) \cong S \to S \times V$ (State monad)

Scanned with CamScanner

$\text{Free}_{\text{State}}(V) = \text{Free}_{\text{State}}(V) / \approx_{\text{State}}$

look at return, get, put
w/ subtrees

never have 2 conseq. gets

get to form

read then write

$$\underset{\text{get}}{S} \to \underset{\text{put}}{S} \times \underset{\text{result}}{V}$$

Free models form a monad

general monad structure of free

agree w/ monad structure on

Improve notation

(explicitly continuations not good)

(6) Generic operations & sequencing

(just notation)

- generic operation

$$\overline{op}(p) := op(p, \lambda x. \text{return } x)$$

Scanned with CamScanner

• <u>Sequencing</u>

$$\text{do } x \leftarrow t_1 \text{ in } t_2(x)$$

first perform t_1

(does effects)

gives results $- x$

then feed into t_2

Haskell $\quad$ do $x \leftarrow t_1$

$\qquad\qquad t_2(x)$

ML $\quad$ let $x = t_1$ in t_2

(works for free model)

t_1 either return or operation

$$(\text{do } x \leftarrow \text{return } v \text{ in } t_2(x)) = t_2(v)$$

$$(\text{do } x \leftarrow op(p, \underset{y}{\kappa}) \text{ in } t_2(x)) = op(p, \lambda y.\, \text{do } x \leftarrow \kappa(y) \text{ in } t_2(x))$$

first op

produces result

feeds to κ

κ does work

return into x

feed into t_2

Scanned with CamScanner

old notation - get rid altogether

~~$op(p_1, \kappa)$~~

$$op(p, \lambda x. \kappa(x)) = do \; x \leftarrow \overline{op}(p) \text{ in } \kappa(x)$$

old notation:

know about continuations

Programming is for humans (not machine)

give <u>do</u> and <u>op</u>

$\overline{op}$ look like functions

$\overline{op} : P \to A$

f.) $op : P \rightsquigarrow A$

Scanned with CamScanner