3. Domain Theory I: Semantic Algebras

Primitive Domains

- Truth values

  Domain $Tr = \mathbf{B}$

  Operations

  true: Tr

  false: Tr

  not: $Tr \rightarrow Tr$

  or: $Tr \times Tr \rightarrow Tr$

  $(_ \rightarrow _ \,[]\, _)$: $Tr \times D \times D \rightarrow D$

  (for every domain D*)*

- Additional Nat operations

  equals: $Nat \times Nat \rightarrow Tr$

  lessthan: $Nat \times Nat \rightarrow Tr$

  greaterthan: $Nat \times Nat \rightarrow Tr$

- One element domain

  Domain *Unit*, the one element-domain

  Operations *():* *Unit*

Primitive Domains

- Character strings

  Domain *String* = the character strings from elements of $\mathbf{C}$ (including "error")
  Operations

  A, B, C, . . . , Z: String
  empty: String
  error: String
  concat: String $\times$ *String* $\longrightarrow$ *String*
  length: String $\longrightarrow$ *Nat*
  substr: String $\times$ *Nat* $\times$ *Nat* $\longrightarrow$ *String*

- Computer store locations

  Domain *Location*, the address space in a computer store
  Operations

  first-locn: Location
  next-locn: Location $\longrightarrow$ *Location*
  eq-locn: Location $\times$ *Location* $\longrightarrow$ *Tr*
  less-l: Location $\times$ *Location* $\longrightarrow$ *Tr*

Product Domains

Payroll information (name, payrate, hours)

Domain $Payroll = String \times Rat \times Rat$

Operations

$newemp: String \to Payroll$

$newemp(name) = (name, min, 0)$

where $min \in Rat$

and $0 = makerat(0)(1)$

$upd\text{-}payrate: Rat \times Payroll \to Payroll$

$upd\text{-}payrate(pay, emp) =$

$(emp{\downarrow}1, pay, emp{\downarrow}3)$

$upd\text{-}hours: Rat \times Payroll \to Payroll$

$upd\text{-}hours(hours, emp) =$

$(emp{\downarrow}1, emp{\downarrow}2,$

$addrat(hours)(emp{\downarrow}3))$

$compute\text{-}pay: Payroll \to Rat$

$compute\text{-}pay(emp) =$

$multrat(emp{\downarrow}2)(emp{\downarrow}3)$

$(a_1, a_2, \ldots, a_n) \downarrow i = a_i$

Sum Domains

Revised payroll information

Domain *Payroll* =

String × (*Day* + *Night*) × *Rat*

where *Day* = *Night* = *Rat*

Operations

newemp: String → *Payroll*

newemp(n) = (*n*, in*Day(min)*, *0*)

move-to-day: Payroll → *Payroll*

move-to-day(emp) = (*emp*↓*1*,

cases *emp*↓*2* of

is*Day(wage)* → in*Day(wage)*

is*Night(wage)* → in*Day(wage)*

end, *emp*↓*3*)

compute-pay: Payroll → *Rat*

compute-pay(emp) =

cases *emp*↓*2* of is*Day(wage)* →

multrat(wage)(emp↓*3)*

is*Night(wage)* → *multrat(1.5)*

(multrat(wage)(emp↓*3))* end

Sum Domains

Truth values as disjoint union

Domain $Tr = TT + FF$
  where $TT = Unit$ and $FF = Unit$
Operations

$true = \mathrm{in}TT()$
$false = \mathrm{in}FF()$
$not(t) =$ cases t of
  $\mathrm{is}TT() \rightarrow \mathrm{in}FF()$
  $\mathrm{is}FF() \rightarrow \mathrm{in}TT()$
 end
$or(t, u) =$ cases t of
  $\mathrm{is}TT() \rightarrow \mathrm{in}TT()$
  $\mathrm{is}FF() \rightarrow$ cases u of
    $\mathrm{is}TT() \rightarrow \mathrm{in}TT()$
    $\mathrm{is}FF() \rightarrow \mathrm{in}FF()$
  end
end
$(t \rightarrow e \,[]\, f) =$ cases t of $\mathrm{is}TT() \rightarrow e$
 $\mathrm{is}FF() \rightarrow f$ end

Sum Domains

Finite lists

Domain $D^* = Unit + D + (D \times D) + (D \times (D \times D)) + \ldots$

Operations

nil: D^*

$nil = \text{in}Unit()$

$cons$: $D \times D^* \rightarrow D^*$

$cons(d,l) =$ cases l of

$\text{is}Unit() \rightarrow \text{in}D(d)$

$\text{is}D(y) \rightarrow \text{in}D\times D(d,y)$

$\text{is}D\times D(y) \rightarrow \text{in}D\times(D\times D)(d,y)$

$\ldots$

end

hd: $D^* \rightarrow D$ $hd(l) =$ cases l of

$\text{is}Unit() \rightarrow error$

$\text{is}D(y) \rightarrow y$

$\text{is}D\times D(y) \rightarrow fst(y)$

$\text{is}D\times(D\times D)(y) \rightarrow fst(y)$

$\ldots$

end

Function Domains

Dynamic arrays

Domain $Array = Nat \to A$

where A is a domain with an *error* element

Operations

newarray: $Array$

$newarray = \lambda n.error$

access: $Nat \times Array \to A$

$access(n, r) = r(n)$

update: $Nat \times A \times Array \to Array$

$access(n, v, r) = [n \mapsto v]r$

Bounds are not restricted!

$access(m, update(n, v, r))$ where m $\neq$ n

$= (update(n, v, r))(m)$ definition of *access*

$= ([n \mapsto v]r)(m)$ definition of *update*

$= (\lambda m.(equals\ m\ n) \to v\ [\!]\ r(m))(m)$ update

$= (equals\ m\ n) \to v\ [\!]\ r(m)$ function application

$= false \to v\ [\!]\ r(m)$

$= r(m)$

Function Domains

Dynamic arrays with curried operations

Domain $Array = Nat \to A$

where A is a domain with an *error* element

Operations

$newarray:\ Array$

$newarray = \lambda n.error$

$access:\ Nat \to Array \to A$

$access = \lambda n.\lambda r.r(n)$

$update:\ Nat \to A \to Array \to Array$

$access = \lambda n.\lambda v.\lambda r.[n \mapsto v]r$

Functions take one argument at a time!

$access(k) = \lambda r.r(k)$

$access(k)(r) \Rightarrow r(k)$

$(access\ k\ r) \Rightarrow r(k)$

Lifted Domains

Unsafe arrays of unsafe values

Domain $Uarr = Array_\perp$
where $Array = Nat \to Tr'$
and $Tr' = (\mathbf{B} \cup \{error\})_\perp$
Operations

new-unsafe: $Uarr$
$new\text{-}unsafe = newarray$
access-unsafe: $Nat_\perp \to Uarr \to Tr'$
$access\text{-}unsafe = \underline{\lambda} n.\underline{\lambda} r.(access\ n\ r)$
upd-unsf: $Nat_\perp \to Tr' \to Uarr \to Uarr$
$upd\text{-}unsf = \underline{\lambda} n.\lambda t.\underline{\lambda} r.(update\ n\ t\ r)$

Indices and elements may be improper!

$upd\text{-}unsf(plus\ one\ two)(not'(\perp))(new\text{-}unsafe)$
$= upd\text{-}unsf(plus\ one\ two)(not'(\perp))(newarray)$
$= upd\text{-}unsf(plus\ one\ two)(not'(\perp))(\lambda n.error)$
$= upd\text{-}unsf(three)(\perp)(\lambda n.error)$
$= update(three)(\perp)(\lambda n.error)$
$= [three \mapsto \perp](\lambda n.error)$ $\quad (not' = \underline{\lambda} t.not(t))$

Recursive Function Definitions

Recursive function definitions need not define a function uniquely!

$$q(x) = (\mathit{equals}\ x\ \mathit{zero}) \rightarrow\ \mathit{one}$$
$$[]\ q(\mathit{plus}\ x\ \mathit{one})$$

$$f_1(x) = \begin{cases} \mathit{one} \text{ if } x = \mathit{zero} \\ \bot \quad \text{otherwise} \end{cases}$$

$$f_2(x) = \begin{cases} \mathit{one} \text{ if } x = \mathit{zero} \\ \mathit{two} \text{ otherwise} \end{cases}$$

$$f_3(x) = \mathit{one}$$

How to formmalize that q yields the intended denotation f_1?
(the problem will be handled later)
(same with recursive *domain* definitions)