

Semantic Algebras

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
<http://www.risc.uni-linz.ac.at/people/schreine>

Primitive Domains

- Truth values

Domain $\text{Tr} = \mathbf{B}$

Operations

true: Tr

false: Tr

not: $\text{Tr} \rightarrow \text{Tr}$

or: $\text{Tr} \times \text{Tr} \rightarrow \text{Tr}$

$(- \rightarrow - \ \& \ -)$: $\text{Tr} \times D \times D \rightarrow D$

(for every domain D)

- Additional Nat operations

equals: $\text{Nat} \times \text{Nat} \rightarrow \text{Tr}$

lessthan: $\text{Nat} \times \text{Nat} \rightarrow \text{Tr}$

greaterthan: $\text{Nat} \times \text{Nat} \rightarrow \text{Tr}$

- One element domain

Domain Unit, the one element-domain

Operations $()$: Unit

Primitive Domains

- Character strings

Domain String = the character strings from elements of \mathbb{C} (including “error”)

Operations

$A, B, C, \dots, Z: \text{String}$

$\text{empty}: \text{String}$

$\text{error}: \text{String}$

$\text{concat}: \text{String} \times \text{String} \rightarrow \text{String}$

$\text{length}: \text{String} \rightarrow \text{Nat}$

$\text{substr}: \text{String} \times \text{Nat} \times \text{Nat} \rightarrow \text{String}$

- Computer store locations

Domain Location , the address space in a computer store

Operations

$\text{first-locn}: \text{Location}$

$\text{next-locn}: \text{Location} \rightarrow \text{Location}$

$\text{eq-locn}: \text{Location} \times \text{Location} \rightarrow \text{Tr}$

$\text{less-l}: \text{Location} \times \text{Location} \rightarrow \text{Tr}$

Product Domains

Payroll information (name, payrate, hours)

Domain Payroll =
 $\text{String} \times \text{Rat} \times \text{Rat}$

Operations

newemp: $\text{String} \rightarrow \text{Payroll}$

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

where $\text{min} \in \text{Rat}$

and $0 = \text{makerat}(0)(1)$

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

upd-payrate(pay, emp) =

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

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

upd-hours(hours, emp) =

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

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

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

compute-pay(emp) =

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

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

Sum Domains

Revised payroll information

Domain Payroll =

$\text{String} \times (\text{Day} + \text{Night}) \times \text{Rat}$

where $\text{Day} = \text{Night} = \text{Rat}$

Operations

$\text{newemp}: \text{String} \rightarrow \text{Payroll}$

$\text{newemp}(n) = (n, \text{inDay}(\text{min}), 0)$

$\text{move-to-day}: \text{Payroll} \rightarrow \text{Payroll}$

$\text{move-to-day}(\text{emp}) = (\text{emp}\downarrow 1,$

cases $\text{emp}\downarrow 2$ of

$\text{isDay}(\text{wage}) \rightarrow \text{inDay}(\text{wage})$

$\text{isNight}(\text{wage}) \rightarrow \text{inDay}(\text{wage})$

end,

$\text{emp}\downarrow 3)$

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

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

cases $\text{emp}\downarrow 2$ of

$\text{isDay}(\text{wage}) \rightarrow$

$\text{multrat}(\text{wage})(\text{emp}\downarrow 3)$

$\text{isNight}(\text{wage}) \rightarrow \text{multrat}(1.5)$

$(\text{multrat}(\text{wage})(\text{emp}\downarrow 3))$

end

Sum Domains

Truth values as disjoint union

Domain $Tr = TT + FF$

where $TT = Unit$ and $FF = Unit$

Operations

true = inTT()

false = inFF()

not(t) =

cases t of

isTT() → inFF()

isFF() → inTT()

end

or(t, u) =

cases t of

isTT() → inTT()

isFF() → cases u of

isTT() → inTT()

isFF() → inFF()

end

end

(t → e [] f) = cases t of

isTT() → e

isFF() → f

end

Sum Domains

Finite lists

$$\text{Domain } D^* = \text{Unit} + D + (D \times D) \\ + (D \times (D \times D)) + \dots$$

Operations

nil: D^*

nil = inUnit()

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

cons(d,l) =

cases l of

isUnit() \rightarrow inD(d)

isD(y) \rightarrow inD \times D(d,y)

isD \times D(y) \rightarrow inD \times (D \times D)(d,y)

...

end

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

cases l of

isUnit() \rightarrow error

isD(y) \rightarrow y

isD \times D(y) \rightarrow fst(y)

isD \times (D \times D)(y) \rightarrow fst(y)

...

end

Function Domains

Dynamic arrays

$$\text{Domain Array} = \text{Nat} \rightarrow A$$

where A is a domain with an *error* element

Operations

newarray: Array

$$\text{newarray} = \lambda n. \text{error}$$

access: $\text{Nat} \times \text{Array} \rightarrow A$

$$\text{access}(n, r) = r(n)$$

update: $\text{Nat} \times A \times \text{Array} \rightarrow \text{Array}$

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

Bounds are not restricted!

$$\begin{aligned} & \text{access}(m, \text{update}(n, v, r)) \\ &= (\text{update}(n, v, r))(m) \\ &= ([n \mapsto v]r)(m) \\ &= (\lambda m. (\text{equals } m \ n) \rightarrow v \ [] \ r(m))(m) \\ &= (\text{equals } m \ n) \rightarrow v \ [] \ r(m) \\ &= \text{false} \rightarrow v \ [] \ r(m) \\ &= r(m) \end{aligned}$$

where $m \neq n$
 definition of *access*
 definition of *update*
 update
 function application

Function Domains

Dynamic arrays with curried operations

Domain $\text{Array} = \text{Nat} \rightarrow A$

where A is a domain with an *error* element

Operations

$\text{newarray}: \text{Array}$

$\text{newarray} = \lambda n.\text{error}$

$\text{access}: \text{Nat} \rightarrow \text{Array} \rightarrow A$

$\text{access} = \lambda n.\lambda r.r(n)$

$\text{update}: \text{Nat} \rightarrow A \rightarrow \text{Array} \rightarrow \text{Array}$

$\text{update} = \lambda n.\lambda v.\lambda r.[n \mapsto v]r$

Functions take one argument at a time!

$\text{access}(k) = \lambda r.r(k)$

$\text{access}(k)(r) \rightarrow r(k)$

$(\text{access } k \ r) \rightarrow r(k)$

Lifted Domains

Unsafe arrays of unsafe values

Domain $Uarr = Array_{\perp}$

where $Array = Nat \rightarrow Tr'$

and $Tr' = (\mathbf{B} \cup \{\text{error}\})_{\perp}$

Operations

new-unsafe: $Uarr$

new-unsafe = newarray

access-unsafe: $Nat_{\perp} \rightarrow Uarr \rightarrow Tr'$

access-unsafe = $\underline{\lambda}n.\underline{\lambda}r.(\text{access } n \ r)$

upd-unsf: $Nat_{\perp} \rightarrow Tr' \rightarrow Uarr \rightarrow Uarr$

upd-unsf = $\underline{\lambda}n.\underline{\lambda}t.\underline{\lambda}r.(\text{update } n \ t \ r)$

Indices and elements may be improper!

upd-unsf(plus one two)(not'(\perp))(new-unsafe)

= upd-unsf(plus one two)(not'(\perp))(newarray)

= upd-unsf(plus one two)(not'(\perp))($\underline{\lambda}n.\text{error}$)

= upd-unsf(three)(\perp)($\underline{\lambda}n.\text{error}$)

= update(three)(\perp)($\underline{\lambda}n.\text{error}$)

= [three \mapsto \perp]($\underline{\lambda}n.\text{error}$)

(not' = $\underline{\lambda}t.\text{not}(t)$)

Recursive Function Definitions

Recursive function definitions need not define a function uniquely!

$$q(x) = \begin{array}{l} (\text{equals } x \text{ zero}) \rightarrow \text{one} \\ \square q(\text{plus } x \text{ one}) \end{array}$$

$$f_1(x) = \begin{cases} \text{one} & \text{if } x = \text{zero} \\ \perp & \text{otherwise} \end{cases}$$

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

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

How to formalize that q yields the intended denotation f_1 ?

(the problem will be handled later)

(same with recursive *domain* definitions)