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 Tr = **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 $\to$ String
  length: String $\to$ Nat
  substr: String $\times$ Nat $\times$ Nat $\to$ String

- Computer store locations

  Domain Location, the address space in a computer store

  Operations

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

Product Domains

Payroll information (name, payrate, hours)

Domain Payroll =
String × Rat × Rat

Operations

newemp: String → Payroll
newemp(name) = (name, min, 0)
where min ∈ Rat
and 0 = makerat(0)(1)
upd-payrate: Rat × Payroll → Payroll
upd-payrate(pay, emp) =
(emp↓1, pay, emp↓3)
upd-hours: Rat × Payroll → Payroll
upd-hours(hours, emp) =
(emp↓1,emp↓2,
addrat(hours)(emp↓3))
compute-pay: Payroll → Rat
compute-pay(emp) =
multrat(emp↓2)(emp↓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, inDay(min), 0)
move-to-day: Payroll → Payroll
move-to-day(emp) = (emp↓1,
    cases emp↓2 of
        isDay(wage) → inDay(wage)
        isNight(wage) → inDay(wage)
    end,
    emp↓3)
compute-pay: Payroll → Rat
compute-pay(emp) =
    cases emp↓2 of
        isDay(wage) →
            multrat(wage)(emp↓3)
        isNight(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 = 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

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

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

Domain Array = Nat $\to$ A

where *A* is a domain with an *error* element

Operations

newarray: Array

$\quad$ newarray = λn.error

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

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

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

$\quad$ 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*
= (λm.(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 = λn.error

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

access = λn.λr.r(n)

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

access = λn.λv.λr.[n $\mapsto$ v]r

Functions take one argument at a time!

access(k) = λr.r(k)

access(k)(r) $\to$ r(k)

(access k r) $\to$ r(k)

Lifted Domains

Unsafe arrays of unsafe values

Domain Uarr = $\text{Array}_\perp$

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

Operations

new-unsafe: Uarr
  new-unsafe = newarray
access-unsafe: $\text{Nat}_\perp \rightarrow \text{Uarr} \rightarrow \text{Tr}'$
  access-unsafe = $\underline{\lambda}$n.$\underline{\lambda}$r.(access n r)
upd-unsf: $\text{Nat}_\perp \rightarrow \text{Tr}' \rightarrow \text{Uarr} \rightarrow \text{Uarr}$
  upd-unsf = $\underline{\lambda}$n.λt.$\underline{\lambda}$r.(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$'$($\perp$))(λn.error)
= upd-unsf(three)($\perp$)(λn.error)
= update(three)($\perp$)(λn.error)
= [three $\mapsto$ $\perp$](λn.error)   (not$'$=$\underline{\lambda}$t.not(t))

Recursive Function Definitions

Recursive function definitions need not define a function uniquely!

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

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

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

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

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