# 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 $\rightarrow$ String
  length: String $\rightarrow$ Nat
  substr: String $\times$ Nat $\times$ Nat $\rightarrow$ String

- Computer store locations

  Domain Location, the address space in a computer store

  Operations

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

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

where Day = Night = Rat

Operations

newemp: String $\rightarrow$ Payroll
newemp(n) = (n, inDay(min), 0)
move-to-day: Payroll $\rightarrow$ Payroll
move-to-day(emp) = (emp$\downarrow$1,
    cases emp$\downarrow$2 of
        isDay(wage) $\rightarrow$ inDay(wage)
        isNight(wage) $\rightarrow$ inDay(wage)
    end,
    emp$\downarrow$3)
compute-pay: Payroll $\rightarrow$ Rat
compute-pay(emp) =
    cases emp$\downarrow$2 of
        isDay(wage) $\rightarrow$
            multrat(wage)(emp$\downarrow$3)
        isNight(wage) $\rightarrow$ multrat(1.5)
            (multrat(wage)(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

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

Operations

nil: $D^*$
nil = inUnit()
cons: $D \times D^* \to D^*$
cons(d,l) =
    cases l of
        isUnit() $\to$ inD(d)
        isD(y) $\to$ inD$\times$D(d,y)
        isD$\times$D(y) $\to$ inD$\times$(D$\times$D)(d,y)
        ...
    end
hd: $D^* \to D$ hd(l) =
    cases l of
        isUnit() $\to$ error
        isD(y) $\to$ y
        isD$\times$D(y) $\to$ fst(y)
        isD$\times$(D$\times$D)(y) $\to$ fst(y)
        ...
    end

# Function Domains

## Dynamic arrays

### Domain Array = Nat $\rightarrow$ A

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

### Operations

newarray: Array
    newarray = $\lambda$n.error
access: Nat $\times$ Array $\rightarrow$ A
    access(n, r) = r(n)
update: Nat $\times$ A $\times$ Array $\rightarrow$ 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) $\rightarrow$ v [] r(m))(m) | update |
| = (equals m n) $\rightarrow$ v [] r(m) | function application |
| = false $\rightarrow$ v [] r(m) | |
| = r(m) | |

# Function Domains

Dynamic arrays with curried operations

Domain Array = Nat $\rightarrow$ A

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

Operations

newarray: Array

$\quad$ newarray = $\lambda$n.error

access: Nat $\rightarrow$ Array $\rightarrow$ A

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

update: Nat $\rightarrow$ A $\rightarrow$ Array $\rightarrow$ Array

$\quad$ 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 = $\text{Array}_\perp$

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

### Operations

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