

Quick Overview of VDM Operators

The Boolean type

Operator	Name	Signature
not b	Negation	bool \rightarrow bool
a and b	Conjunction	bool * bool \rightarrow bool
a or b	Disjunction	bool * bool \rightarrow bool
a \Rightarrow b	Implication	bool * bool \rightarrow bool
a \Leftrightarrow b	Biimplication	bool * bool \rightarrow bool
a = b	Equality	bool * bool \rightarrow bool
a $<>$ b	Inequality	bool * bool \rightarrow bool

The numeric types

Operator	Name	Signature
-x	Unary minus	real \rightarrow real
abs x	Absolute value	real \rightarrow real
floor x	Floor	real \rightarrow int
x + y	Sum	real * real \rightarrow real
x - y	Difference	real * real \rightarrow real
x * y	Product	real * real \rightarrow real
x / y	Division	real * real \rightarrow real
x div y	Integer division	int * int \rightarrow int
x rem y	Remainder	int * int \rightarrow int
x mod y	Modulus	int * int \rightarrow int
x**y	Power	real * real \rightarrow real
x < y	Less than	real * real \rightarrow bool
x > y	Greater than	real * real \rightarrow bool
x <= y	Less or equal	real * real \rightarrow bool
x >= y	Greater or equal	real * real \rightarrow bool
x = y	Equal	real * real \rightarrow bool
x <> y	Not equal	real * real \rightarrow bool

The character, quote and token types

Operator	Name	Signature
c1 = c2	Equal	char * char \rightarrow bool
c1 <> c2	Not equal	char * char \rightarrow bool

Tuple types

Operator	Name	Signature
t1 = t2	Equality	T * T \rightarrow bool
t1 <> t2	Inequality	T * T \rightarrow bool

Record types

Operator	Name	Signature
r.i	Field select	A * Id \rightarrow A_i
r1 = r2	Equality	A * A \rightarrow bool
r1 <> r2	Inequality	A * A \rightarrow bool
is .A(r1)	Is	Id * MasterA \rightarrow bool

Union and optional types

Operator	Name	Signature
t1 = t2	Equality	A * A \rightarrow bool
t1 <> t2	Inequality	A * A \rightarrow bool

Set types

Operator	Name	Signature
e in set s1	Membership	A * set of A \rightarrow bool
e not in set s1	Not membership	A * set of A \rightarrow bool
s1 union s2	Union	set of A * set of A \rightarrow set of A
s1 inter s2	Intersection	set of A * set of A \rightarrow set of A
s1 \ s2	Difference	set of A * set of A \rightarrow set of A
s1 subset s2	Subset	set of A * set of A \rightarrow bool
s1 psubset s2	Proper subset	set of A * set of A \rightarrow bool
s1 = s2	Equality	set of A * set of A \rightarrow bool
s1 <> s2	Inequality	set of A * set of A \rightarrow bool
card s1	Cardinality	set of A \rightarrow nat
dunion ss	Distributed union	set of set of A \rightarrow set of A
dinter ss	Distributed intersection	set of set of A \rightarrow set of A
power ss	Finite power set	set of A \rightarrow set of set of A

Sequence types

Operator	Name	Signature
hd l	Head	seq1 of A \rightarrow A
tl l	Tail	seq1 of A \rightarrow seq of A
len l	Length	seq of A \rightarrow nat
elems l	Elements	seq of A \rightarrow set of A
inds l	Indices	seq of A \rightarrow set of nat1
l1 ^ l2	Concatenation	(seq of A) * (seq of A) \rightarrow seq of A
reverse l	Reverse	seq of A \rightarrow seq of A
conc ll	Distributed concatenation	seq of seq of A \rightarrow seq of A
l ++ m	Sequence modification	seq of A * map nat to A \rightarrow seq of A
l(i)	Sequence index	seq of A * nat1 \rightarrow A
l1 = l2	Equality	(seq of A) * (seq of A) \rightarrow bool
l1 <> l2	Inequality	(seq of A) * (seq of A) \rightarrow bool

Mapping types

Operator	Name	Signature
dom m	Domain	(map A to B) \rightarrow set of A
rng m	Range	(map A to B) \rightarrow set of B
m1 munion m2	Map union	(map A to B) * (map A to B) \rightarrow map A to B
m1 ++ m2	Override	(map A to B) * (map A to B) \rightarrow map A to B
merge ms	Distributed merge	set of (map A to B) \rightarrow map A to B
s <: m	Domain restrict to	(set of A) * (map A to B) \rightarrow map A to B
s <-: m	Domain restrict by	(set of A) * (map A to B) \rightarrow map A to B
m :> s	Range restrict to	(map A to B) * (set of B) \rightarrow map A to B
m :-> s	Range restrict by	(map A to B) * (set of B) \rightarrow map A to B
m(d)	Mapping apply	(map A to B) * A \rightarrow B
inverse m	Map inverse	inmap A to B \rightarrow inmap B to A
m1 = m2	Equality	(map A to B) * (map A to B) \rightarrow bool
m1 <> m2	Inequality	(map A to B) * (map A to B) \rightarrow bool

Examples

General

```
if predicate then Expression else Expression
cases expression:
  (pattern list 1)-> Expression 1,
  (pattern list 2),
  (pattern list 3)-> Expression 2,
  others -> Expression 3
end;

for all value in set setOfValues
  do Expression

dcl variable : type := Variable creation ;
let variable : type = Variable creation in Expression
let variable in set setOfValues be st pred(variable) in Expression
```

Comprehensions (Structure to Structure)

```
{element(var) | var in set setexpr & pred(var) }

[element(i) | i in set numsetexpr & pred(i)]

Typically:

[element(list(i)) | i in set inds list & pred(list(i)) ]

{dexpr(var) |-> rexpr(var) | var in set setexpr & pred(var) }
```

From Structure to Arbitrary Value

```
Select: set of nat -> nat
Select(s) ==
  let e in set s
  in
    e
pre s <> {}
```

From Structure to Single Value

```
SumSet: set of nat -> nat
SumSet(s) ==
  if s = {}
  then 0
  else let e in set s
    in
      e + SumSet(s\{e})
```

From Structure to single Boolean

```
forall p in set setOfP & pred(p)

exists p in set setOfP & pred(p)

exists1 p in set setOfP & pred(p)
```

```
class Person

types
public String = seq of char;

values

protected Name : seq of char = "Peter";

instance variables

public nationality : seq of char := "Danish";
comment          : String;
yearOfBirth      : int;
sex              : Male | Female;
friends          : map String to Person;

operations

public GetAge : int ==> int
GetAge(year) == CalculateAge(year,yearOfBirth)
pre pre_CalculateYear(year,yearOfBirth);

functions

public CalculateAge : int * int -> int
CalculateAge (year,bornInYear) == year-bornInYear
pre year >= bornInYear;

thread
while true do
  skip;

traces
  Mytrace: regular expression using operation calls

end Person

class Male is subclass of Person
end Male
class Female is subclass of Person
end Female
```

Listing 1: Class Example