The VDM-SL Quick Reference

General Operators

expr = *expr*	Equality
expr <> *expr*	Inequality

Character, Quote, Token Operators

See *General Operators*

Logical Operators

not *expr*	Negation
expr and *expr*	Conjunction
expr or *expr*	Disjunction
expr => *expr*	Implication
expr <=> *expr*	Biimplication

Numerical Operators

- *expr*	Unary minus
abs *expr*	Absolute value
floor *expr*	Floor
expr + *expr*	Sum
expr - *expr*	Difference
expr * *expr*	Product
expr / *expr*	Division
expr div *expr*	Integer division
expr rem *expr*	Remainder
expr mod *expr*	Modulus
expr ** *expr*	Power
expr < *expr*	Less than
expr > *expr*	Greater than
expr <= *expr*	Less or equal
expr >= *expr*	Greater or equal

Set Operators

expr in set *expr*	Membership
expr not in set *expr*	Not membership

Sequence Operators

hd *expr*	Head
tl *expr*	Tail
len *expr*	Length
elems *expr*	Elements
inds *expr*	Indexes
expr ^ *expr*	Concatenation
conc *expr*	Distributed concatenation
expr (*expr*)	Sequence application
expr ++ *expr*	Override

Map Operators

dom *expr*	Domain
rng *expr*	Range
expr munion *expr*	Union
expr ++ *expr*	Override
merge *expr*	Distributed merge
expr <: *expr*	Domain restrict to
expr <-: *expr*	Domain restrict by
expr :> *expr*	Range restrict to
expr :-> *expr*	Range restrict by
expr (*expr*)	Map apply
expr comp *expr*	Map composition
expr ** *expr*	Map iteration
inverse *expr*	Map inverse

stmt =

def ⟨ *pattern bind* = *expr* ⟩$_{[[;]]}$ in *stmt*	def statement
let ⟨ *local definition* ⟩$_{[,]}$ in *stmt*	let statement
let *bind* [be st *expr*] in *stmt*	let be statement
state designator := *expr*	assign statement
({ *dcl stmt* } ⟨ *stmt* ⟩$_{[[;]]}$)	block statement
for *pattern* in [reverse] *expr* do *stmt*	sequence for loop
for all *pattern* in set *expr* do *stmt*	set for loop
for *id* = *expr* to *expr* [by *expr*] do *stmt*	index for loop

exit *expr*	exit statement
skip	identity statement
error	error statement

dcl stmt =

dcl ⟨ *id* : *type* [:= *expr*] ⟩$_{[,]}$;	dcl statement

expr =

def ⟨ *pattern bind* = *expr* ⟩$_{[[;]]}$ in *expr*	def expression
let ⟨ *local definition* ⟩$_{[,]}$ in *expr*	let expression
let *bind* [be st *expr*] in *expr*	let be expression
if *expr* then *expr* else *expr*	if expression
cases *expr* : ⟨ ⟨ *pattern* ⟩$_{[,]}$ -> *expr* ⟩$_{[,]}$ [, others -> *expr*] end	cases expression
forall ⟨ *mult bind* ⟩$_{[,]}$ & *expr*	forall quantifier
exists ⟨ *mult bind* ⟩$_{[,]}$ & *expr*	exist quantifier
exists1 *bind* &*expr*	exists unique quantifier
{[⟨ *expr* ⟩$_{[,]}$]}	set enumeration
{*expr* | ⟨ *mult bind* ⟩$_{[,]}$ [& *expr*]}	set comprehension
{*expr* , ... , *expr*}	set range
[[⟨ *expr* ⟩$_{[,]}$]]	sequence enumeration
[*expr* | *bind* [& *expr*]]	sequence comprehension
expr (*expr* , ... , *expr*)	subsequence
expr ++ {⟨ *expr* |-> *expr* ⟩$_{[,]}$ }	sequence modifier
{⟨ *expr* |-> *expr* ⟩$_{[,]}$ } | {|-> }	map enumeration
{*expr* |-> *expr* | ⟨ *mult bind* ⟩$_{[,]}$ [& *expr*]}	map comprehension
mk_(*expr*, ⟨ *expr* ⟩$_{[,]}$)	tuple constructor
mk_*id*([⟨ *expr* ⟩$_{[,]}$])	record constructor
mu (*expr* , ⟨ *id* |-> *expr* ⟩$_{[,]}$)	record modifier
expr ([⟨ *expr* ⟩$_{[,]}$])	apply expression
expr . *id*	field select expression
is_ (*basic type* | *id*) (*expr*)	is expression
undefined	undefined expression
name [⟨ *type*⟩$_{[,]}$]	function type instatiation

bind =

pattern in set *expr*	set bind
pattern : *type*	type bind

mult bind =

⟨ *pattern* ⟩$_{[,]}$ in set *expr*	multiple set bind
⟨ *pattern* ⟩$_{[,]}$: *type*	multiple type bind

pattern =

id | -	pattern identifier
(*expr*)	match value
{⟨ *pattern* ⟩$_{[,]}$ }| {}	set enumeration pattern
pattern union *pattern*	set union pattern
[⟨ *pattern* ⟩$_{[,]}$]| []	sequence enumeration pattern
pattern ^ *pattern*	sequence concatination pattern
mk_(⟨ *pattern* ⟩$_{[,]}$)	tuple pattern
mk_*id* (⟨ *pattern* ⟩$_{[,]}$)	record pattern

state designator =

id	identifier reference
state designator . *id*	field reference
state designator (*expr*)	map reference
state designator (*expr*)	sequence reference

The meaning of ⟨ *xx* ⟩$_{[s]}$ is at least one occurrence of *xx*, possibly more but then separated by the symbol *s* which is either empty or comma "," or semicolon ";".
⟨ *xx* ⟩$_{[[s]]}$ means the same, except that *s* may follow the last occurence of *xx*.
The meaning of [*xx*] is zero or one occurrence of *xx*.
Definitions that span multiple lines are per default alternatives. An exception to this rule are lines that are ended by comma ",", which imply a multiline sequential definition. Commas are also used to separate symbols where ambiguity would otherwise exist (cf. the definition of *module*).

document =
⟨ *module* | *dynamic link module* ⟩$_{[\ ]}$ | ⟨ *definition block* ⟩$_{[\ ]}$

module = module, *id*, *interface*,
definitions ⟨ *definition block* ⟩$_{[\ ]}$, end *id*

functions ⟨ *name list* : function type ⟩$_{[[;]]}$

import definition list = imports ⟨ *import definition* ⟩$_{[,]}$

import definition = from *id* (all |⟨ *import signature* ⟩$_{[\ ]}$)

import signature =
  types ⟨ (*name* | *type definition*) [renamed *name*] ⟩$_{[[;]]}$
  values ⟨ *name* [: *type*] [renamed *name*] ⟩$_{[[;]]}$
  functions ⟨ *name* [: *function type*] [renamed *name*] ⟩$_{[[;]]}$
  operations ⟨ *name* [: *operation type*] [renamed *name*] ⟩$_{[[;]]}$

instantiation instance list = instantiations ⟨ *instantiation instance* ⟩$_{[,]}$

instantiation instance =
  id as ([*substitutions*]) (all | ⟨ *import signature*)

substitutions = ⟨ *id* -> (*basic type* | *expression*) ⟩$_{[,]}$

export definition = exports (all | ⟨ *export signature* ⟩$_{[\ ]}$)

export signature =
  types ⟨ [struct] *name* ⟩$_{[[;]]}$
  values ⟨ *name list* : *type* ⟩$_{[[;]]}$
  functions ⟨ *name list* : *function type* ⟩$_{[[;]]}$
  operations ⟨ *name list* : *operation type* ⟩$_{[[;]]}$

dynamic link module =
  dlmodule, *id*, *dynamic link interface*, [*use signature*], end *id*

dynamic link interface =
  [*dynamic link import definition list*], *dynamic link export definition*

import definition list =
  imports ⟨ *dynamic link import definition* ⟩$_{[,]}$

dynamic link export signature =
  values ⟨ *name list*, :, type ⟩$_{[[;]]}$
  functions ⟨ *name list*, :, function type ⟩$_{[[;]]}$
  operations ⟨ *name list* : *operation type* ⟩$_{[[;]]}$

definition block = *type definitions*
  state definitions
  value definitions
  function definitions
  operation definitions

type definitions = types ⟨ *type definition* ⟩$_{[[;]]}$

type definition = *id* = *type* [*invariant*]
  id :: [⟨ [*id* :] *type* ⟩$_{[\ ]}$] [*invariant*]

type =

(*type*)|	bracketed type
bool | nat | nat1 | int | rat | real |	
char | token	basic type
< *identifier* >	quote type
compose *id* of [⟨ [*id* :] *type* ⟩$_{[\ ]}$] end	composite type
type | ⟨ *type* ⟩$_{[\|]}$	union type
type * ⟨ *type* ⟩$_{[*]}$	product type
[*type*]	optional type
set of *type*	set type
seq of *type*	seq0 type
seq1 of *type*	seq1 type
map *type* to *type*	general map type
inmap *type* to *type*	injective map type
discretionary type -> *type*	partial function type
name	type name
@ *id*	type variable

function type =

initialization = **init** *invariant initial function*

invariant initial function = *pattern* **==** *expr*

value definitions = **values** ⟨ *pattern* [**:** *type*] **=** *expr* ⟩$_{[[;]]}$

function definitions = **functions** ⟨ *function definition* ⟩$_{[[;]]}$

function definition =
  explicit function definition | *implicit function definition* |
  extended explicit function definition

explicit function definition =
  id [*type variable list*] **:** *function type*
  id parameters list **==** *expr*
  [**pre** *expr*] [**post** *expr*]

implicit function definition =
  id [*type variable list*] *parameter types* ⟨ *id* **:** *type* ⟩$_{[,]}$
  [**pre** *expr*] **post** *expr*

extended explicit function definition =
  id [*type variable list*] *parameter types* **==** ⟨ *id* **:** *type* ⟩$_{[,]}$
  expr
  [**pre** *expr*] [**post** *expr*]

type variable list = **[**⟨ *type variable identifier* ⟩$_{[,]}$ **]**

parameters list = ⟨ *parameters* ⟩$_{[\ ]}$

parameters = **(** [*pattern list*] **)**

parameter types = **(** [⟨ *pattern list* **:** *type* ⟩$_{[,]}$] **)**

operation definitions =

implicit operation definition =
  id parameter types [⟨ *id* : *type*⟩$_{[,]}$]
  [*externals*] [pre *expr*] post *expr*,
  [*exceptions*]

extended explicit operation definition =
  id parameter types [⟨ *id* : *type*⟩$_{[,]}$] ==
  statement
  [*externals*] [pre *expr*] [post *expr*] [*exceptions*]

externals = ext ⟨ rd|wr *name list* [: *type*] ⟩$_{[,]}$

exceptions = errs ⟨ *id* : *expr* -> *expr* ⟩$_{[\ ]}$