Module Internal Design
MID

SFWR ENG 2B03

2003

Robert L. Baber

MID: Purpose

Module Internal Design (MID)

- *internal structure* of a module
- access *and internal* routines of a module
- implementation ("concrete") state variables
- connection between implementation ("concrete") and application ("abstract") state variables
- mathematically precisely

MID: Language

A Module Internal Design is written in

- mathematical
- internal, implementation oriented

language.

Its language is oriented to programming languages in general, possibly but *not necessarily* to the programming language(s) to be used.

MID: Target audience

MID is written for

- module designers and implementers
- inspectors, testers of the module and its parts
- people modifying the module or its components

Note: MID is *not* for designers and implementers of program segments using the module's access routines because of the secrets in the MID.

MID = MIS + internal details

Simply put, the MID consists of the MIS

plus

- specification of the internal (“concrete”) state variables and their internal data structure
- relation between the abstract and the concrete state variables (“abstraction relation”)
- semantics of the access routines in terms of the concrete state variables
- semantics of the internal routines

Abstraction Relation

Abstraction relation

- defines the association between the values of the concrete and the abstract state variables
- normally a function from the concrete to the abstract data spaces (the *abstraction function*)
- i.e. usually one or more concrete states represent one abstract state (not the other way around)

Why Different State Spaces?

Reasons for introducing a concrete state space that is different from the abstract state space

- types of abstract state variables not available in implementation language
- implementation using abstract state space inefficient

What If Concrete = Abstract State Space?

If there is no difference between the concrete state space and the abstract state space

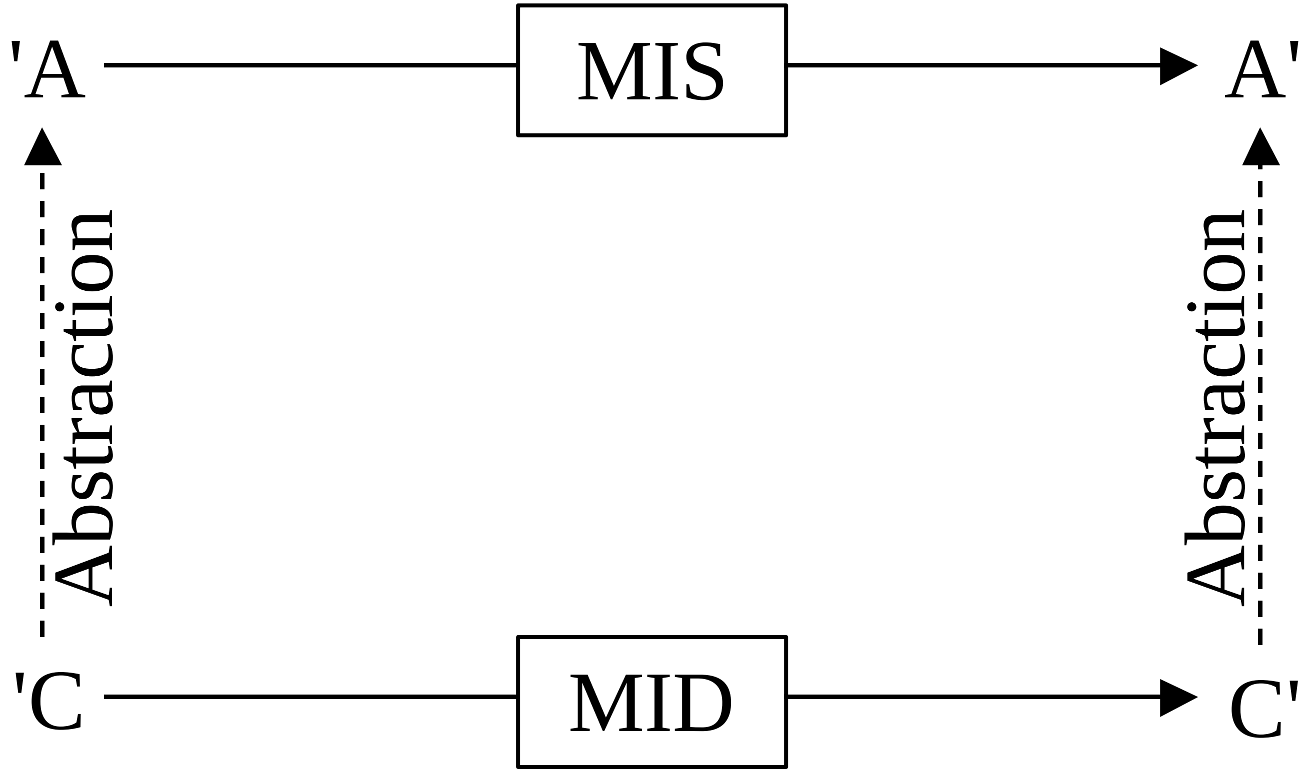
the MID reduces to

- a statement that the concrete state space is the same as the abstract state space and
- semantics of the internal routines

The fact that the concrete and abstract state spaces are the same is still a secret of the module

Abstract and Concrete State Spaces

A = Abstract State Space



C = Concrete State Space

MIS, MID and Abstraction Relation

The three relations on the state spaces

- I/O relation on the abstract state space in MIS
- I/O relation on the concrete state space in MID
- abstraction relation

must be consistent.

E.g. if all three relations are functions, the system must form a homomorphism.

MID Example: Stack Module

- name: Stack
- imported identifiers: A (data type, see below)
- exported access routines: init, push, pop, depth, full
- assumptions: init called before any other access routine

MID Example: Stack Module

- abstract state variables: s, where $s \in A^*$ (s is a sequence of elements of A, A is any set)
- abstract state invariant: $|s| \leq MaxDepth$
- concrete state variables:
  - size (a non-negative integer)
  - stack[0 … MaxDepth-1] of A
- concrete state invariant:
  - $size \in Z \wedge 0 \leq size \leq MaxDepth$

MID Example: Stack Module

- abstraction function:
  - $s = (\& \, i : i \in Z \wedge 0 \leq i \leq size-1 : stack[i])$
  - note that this implies that $|s| = size$
- Note: no apostrophe ' appears before or after any variable name in the abstraction function

MID Example: Stack Module

where MaxDepth

- a positive integer
- an internal implementation parameter
- value not specified at design time

Example: Abstract and Concrete States

Abstract	Concrete	
s	size	stack
[ ]	0	?, … ?
[b]	1	b,?, … ?
[r, w]	2	r, w, ?, … ?
[a, x, p]	3	a, x, p, ?, … ?

? = anything

MID Example: Routine Semantics

- name: init
- input/output relation: size' = 0
- restrictions on use: none

MID Example: Routine Semantics

- name: push(x)
- input/output relation:

  $(\wedge\, i : i \in Z \wedge 0 \le i \le \text{'size}-1 : \text{stack'}[i] = \text{'stack}[i])$

  $\wedge\, \text{stack'}[\text{'size}] = x \wedge \text{size'} = \text{'size}+1$
- domain: $(\text{'size} < \text{MaxDepth}) \wedge (x \in A)$

MID Example: Routine Semantics

- name: pop
- input/output relation:

  $\mathit{result} = \text{'stack['size-1]} \wedge \mathit{result} \in A$

  $\wedge (\wedge\ i : i \in Z \wedge 0 \leq i \leq \text{'size-2} : \text{stack'[i]='stack[i]})$

  $\wedge\ \text{size'} = \text{'size-1}$
- restrictions on use: $0 < \text{'size}$

MID Example: Routine Semantics

- name: depth
- input/output relation:

  $(\mathit{result} = \text{'size}) \wedge (\text{size'} = \text{'size})$

- restrictions on use: none

MID Example: Routine Semantics

- name: full
- input/output relation:

  $(\mathit{result} = (\text{'size} = \text{MaxDepth})) \wedge (\text{size'} = \text{'size})$

- restrictions on use: none

MID Example: Routine Semantics

Convention:

- If any state variable is not explicitly mentioned in the input/output relation, its value is not changed by the access routine in question

MID: Summary

Module Internal Design

- internal view
- mathematically precise
- with implementation ("concrete") state variables
- relation between abstract and concrete state variables
- internal routine semantics