MIS/MID Specifications

SE3A04 – Tutorial

Jason Jaskolka

Department of Computing and Software
Faculty of Engineering
McMaster University
Hamilton, Ontario, Canada
jaskolj@mcmaster.ca

October 30, 2012

Outline

Specifications vs. Descriptions

Definition (Description)

A **description** of a product is a *model* of the product. It should include only certain key aspects of the product and should be *easier to understand* than the product itself.

- Mathematics is used to make descriptions precise
- A variety of descriptions, instead of a single description, is used to efficiently describe the different aspects of a product
- There is *never a complete description* of a product

Specifications vs. Descriptions

Definition (Specification)

A **specification** describes the attributes *required* of a product. A product *satisfies* a specification if it possesses the attributes described by the specification. The product is acceptable if and only if it satisfies the specification.

- A specification serves several purposes:
  1. An agreement between client and developer, designer and implementer, etc.
  2. Blueprint for developing the product
  3. Basis for verifying the correctness of the product
  4. High-level description of the product

Specifications vs. Descriptions

- Both specifications and descriptions describe attributes, but they are different in intent:
  - A **specification** describes the attributes that the product is **required to have**
  - A **description** describes the attributes that the product **actually has**
- The same descriptive item may be interpreted as either a specification or a description
- **Specifications** are often interpreted as **abstract descriptions**
- **Descriptions** are often interpreted as **concrete specifications**

Input/Output Specification

- Let I be a set of possible inputs, and O be a set of possible outputs
- A procedure *without side-effects* can be viewed as a function $f : I \rightarrow O$ that maps inputs to outputs

Definitional Specification

- A **definition** specifies a unique object
- So a definition of a function specifies a unique function:
  - **Syntax**: $f = E$ where E is an expression
  - **Semantics**: f is the unique function denoted by E

Example (Integer Square Function)

$$f = \lambda(x \mid x \in \mathbb{Z} : x \times x)$$

Example (Integer Square Root Function)

$$g = \lambda(x \mid x \in \mathbb{Z} : \exists!(y \mid 0 \leq y : y \times y = x))$$

Relational Specification

- A **relational specification** is a pair (R, D) where:
  1. $R \subseteq I \times O$
  2. $D \subseteq \text{dom}(R) = \{x \in I \mid \exists(y \in O \mid: R(x, y))\} \subseteq I$
- $f : I \to O$ satisfies (R, D) if:
  1. $\forall(x \mid x \in I : x \in \text{dom}(f) \implies R(x, f(x)))$
  2. $D \subseteq \text{dom}(f)$

Relational Specification

Example (Integer Square Function)

$$\begin{aligned} R &= \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid y = x \times x\} \\ D &= \mathbb{Z} \end{aligned}$$

Example (Integer Square Root Function)

$$\begin{aligned} R &= \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x = y \times y\} \\ D &= \{x \in \mathbb{Z} \mid \exists (y \mid y \in \mathbb{Z} : x = y \times y)\} \subseteq \{x \in \mathbb{Z} \mid 0 \leq x\} \end{aligned}$$

Axiomatic Specification

- An **axiomatic specification** is a formula $A(f)$:
  - $A(f)$ is an axiom that expresses the behaviour of f
- $g : I \to O$ satisfies $A(f)$ if $A(g)$ is true

Example (Integer Square Function)

$$A(f) = \forall(x \mid x \in \mathbb{Z} : f(x) = x \times x)$$

Example (Integer Square Root Function)

$$A(f) = \forall(x \mid x \in \mathbb{Z} : \exists(y \mid y \in \mathbb{Z} : y = x \times x) \implies f(x) \times f(x) = x)$$

Before/After Specification

- Let I be a set of possible inputs, O be a set of possible outputs, and S be a set of possible states
- A procedure (*possibly with side-effects*) can be viewed as a function $f : (I \times S) \rightarrow (O \times S)$ that maps inputs and before-states to outputs and after-states
- The function f can be represented as a pair (f_1, f_2) of functions where:
  - $f_1 : (I \times S) \rightarrow O$
  - $f_2 : (I \times S) \rightarrow S$
- An input/output function is a special case of a before/after function where the after-state is always the same as the before-state

Module Interface Specification

Definition (Module Interface Specification)

A **Module Interface Specification (MIS)** specifies the *externally observable behaviour* of a module's access routines.

- Precisely specified with mathematics
- Defines input/output relationships, domains, restrictions on use, exceptions, and application ("abstract") state variables
- Written in a mathematical application language, not the language of the implementation
- Application, not implementation, oriented

Module Interface Specification

Viewpoint

- MIS describes the intended behaviour of a module's access routines from an **external viewpoint**
- Contains absolutely **no information about the insides** of the module or its access routines not implied by the specified external behaviour
- Internal aspects of the module's implementation are **secrets** of the module

Module Interface Specification

Target Audience

- A Module Interface Specification is written for
  1. Module designers and implementers
  2. Inspectors, testers of the module
  3. Designers and implementers of program segments using the module's access routines
- The last group above needs only the MIS
  - If the MIS is not enough for them, then the MIS is not complete, and is not really an MIS

Module Interface Specification

Components

1. Module Name
2. Imported Modules
3. Interface
   - Types
   - Constant Signatures
   - Procedure Signatures
   - Exceptions
4. State Constants with Value Conditions (Invariants)
5. State Variables with Initial Values
6. Behaviour Rules
   - Output Rules
   - State Transition Rules
   - Exception Rules

Before/After MIS Example: Stack

1. **Module Name**: `Stack`
2. **Imported Modules**: `Element`
3. **Interface**:
   - `procedure top(): Element`
   - `procedure height(): Int`
   - `procedure push(e: Element)`
   - `procedure pop()`
   - `exception EmptyStack`
   - `exception FullStack`
4. **State Constants**:
   - `max:Int [ State Invariant: 0` $\leq$ `max ]`
5. **State Variables**:
   - `s` $\in$ A^* `[ s is a sequence of elements of A, initially s =` $\emptyset$ `]`

Before/After MIS Example: Stack

6. **Behaviour Rules**:

Procedure	Input	Output	Transition	Exception
top		first(s)		s = ∅ ⇝ EmptyStack
height		|s|		
push	e:Element		s' = e & s	|s| = max ⇝ FullStack
pop		s' = tail(s)		s = ∅ ⇝ EmptyStack

Module Module Internal Design

Definition (Module Internal Design)

A **Module Internal Design (MID)** specifies the *internal structure* of a module.

- Defines the access and internal routines of a module and the implementation ("concrete") state variables
- Gives the connection between implementation ("concrete") and application ("abstract") state variables
- Precisely specified mathematically
- Written in a mathematical internal, implementation oriented language
- Language is oriented to programming languages in general

Module Internal Design
Target Audience

- A Module Internal Design is written for
  1. Module designers and implementers
  2. Inspectors, testers of the module and its parts
  3. People modifying the module or its components
- An MID is not for designers and implementers of program segments using the module's access routines because of the secrets in the MID

Module Internal Design

Components: 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
  - The 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

Module Internal Design

Components: Abstraction Relation

Definition (Abstraction Relation)

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

- If the concrete state space and the abstract state space are equal, the MID reduces to
  - A statement that the concrete state space is the same as the abstract state space (still a secret of the module)
  - Semantics of the internal routines

MID Example: Stack

1. **Module Name**: `Stack`
2. **Imported Modules**: `Element`
3. **Interface**:
   - `procedure top(): Element`
   - `procedure height(): Int`
   - `procedure push(e: Element)`
   - `procedure pop()`
   - `exception EmptyStack`
   - `exception FullStack`
4. **State Constants**:
   - **Abstract**:
     - `max:Int [ State Invariant: 0` $\leq$ `max ]`
   - **Concrete**:
     - `max:Int [ State Invariant: 0` $\leq$ `max ]`

MID Example: Stack

5. **State Variables**:
   - **Abstract**:
     - `s` $\in$ `A`* `[ s is a sequence of elements of A; initially s =` $\emptyset$ `]`
   - **Concrete**:
     - `size:Int [ State Invariant:` $0 \leq$ `size` $<$ `max; initially size = 0 ]`
     - `a[0 .. max-1]: Element`
6. **Abstraction Function**:
   - `s = &(i |` $\mid$ `i:Int` $\wedge$ $0 \leq$ `i` $\leq$ `size-1 : a[i] )`

MID Example: Stack

7. **Behaviour Semantics**:

Procedure	Input	Output	Transition	Exception
top		a[size-1]		size = 0 ⇝ EmptyStack
height		size		
push	e:Element		a[size] = e ∧ size' = size+1	size = max ⇝ FullStack
pop		size' = size-1		size = 0 ⇝ EmptyStack

Summary

- Three relations on the state spaces
  1. I/O relation on the abstract state space in the MIS
  2. I/O relation on the concrete state space in the MID
  3. Abstraction relation must be consistent throughout

Questions

- Questions?