SE 2A04 Fall 2001

More on Modules

Instructor: W. M. Farmer

Revised: 23 November 2001

Uses of a Before/After MIS

1. To **design** an MID that satisfies the MIS
2. To **simulate** the behavior of a sequence of interface function calls
3. To **graybox test** that an MID satisfies the MIS
4. To **mathematically verify** that an MID satisfies the MIS

Simulation Example: Stack MIS

Call	State	Output	Exception
(initially)	$s = []$		
`Height()`	$s = []$	0	
`Push(3)`	$s = [3]$		
`Push(17)`	$s = [17, 3]$		
`Empty()`	$s = [17, 3]$	`FALSE`	
`MaxHeight()`	$s = [17, 3]$	max	
`Top()`	$s = [17, 3]$	17	
`Pop()`	$s = [3]$		
`Pop()`	$s = []$		
`Pop()`	$s = []$		EmptyStack

Graybox Testing

- Components of a test case:
  - Sequence σ of interface function calls
  - An initial state s
- An **abstraction function** maps MIS states to MID states:

  $$\Phi : \text{states(MIS)} \rightarrow \text{states(MID)}$$

- The test of (σ, s) is positive if a simulation of σ starting at s using the MIS corresponds via Φ to a simulation of σ starting at $\Phi(s)$ using the MID

Uses of an Axiomatic Input/Output MIS

1. To **design** an MID that satisfies the MIS
2. To **explore** the abstract behavior of the interface functions
3. To **blackbox test** that an MID satisfies the MIS
4. To **mathematically verify** that an MID satisfies the MIS

Mathematical Verification

- Let S be an axiomatic input/output MIS
- Let I be an MID that is intended to satisfy S
- I defines a **translation** Φ from the language of S to the language of I
- I **satisfies** S if, for each axiom A of S, $\Phi(A)$ is true

Example: MIS for Lists ADT 1

- Imported modules: none
- Interface:

```
TYPE List;
CONST Nil: List;
PROCEDURE Cons(i: INTEGER; k: List): List;
PROCEDURE Member(i: INTEGER, k: List): INTEGER;
PROCEDURE Take(i: INTEGER, k: List): List;
PROCEDURE Drop(i: INTEGER, k: List): List;
```

- Exceptions:
  - BadIndex: $\texttt{INTEGER} \times \texttt{List} \to \texttt{BOOLEAN}$

Example: MIS for Lists ADT 2

- Input/output specification (axioms):
  1. **Nil is not a Cons list.**
     $\forall\, i : \texttt{INTEGER}, k : \texttt{List}\,.\ \texttt{Nil} \neq \texttt{Cons}(i, k)$
  2. **Cons is one-to-one.**
     $\forall\, i_1, i_2 : \texttt{INTEGER}, k_1, k_2 : \texttt{List}\,.$
     $\texttt{Cons}(i_1, k_1) = \texttt{Cons}(i_2, k_2) \supset (i_1 = i_2 \wedge k_1 = k_2)$
  3. **Induction axiom for lists.**
     $\forall\, P : \texttt{List} \to \texttt{BOOLEAN}\,.$
     $[P(\texttt{Nil}) \wedge$
     $\forall\, k : \texttt{List}\,.\ P(k) \supset \forall\, i : \texttt{INTEGER}\,.\ P(\texttt{Cons}(i, k))]$
     $\supset \forall\, k : \texttt{List}\,.\ P(k)$
  4. **Membership with respect to Nil.**
     $\forall\, i : \texttt{INTEGER}\,.\ \texttt{Member}(i, \texttt{Nil}){\uparrow}$

Example: MIS for Lists ADT 3

5. **Membership with respect to Cons.**
   $\forall\, i, j : \texttt{INTEGER}, k : \texttt{List}\,.$
   $\texttt{Member}(i, \texttt{Cons}(j,k)) \simeq$
   $\text{if}(i < 0, \perp, \text{if}(i = 0, j, \texttt{Member}(i-1, k)))$

6. **Membership with respect to Take.**
   $\forall\, i, j : \texttt{INTEGER}, k : \texttt{List}\,.$
   $\texttt{Member}(i, \texttt{Take}(j,k)) \simeq$
   $\text{if}(i < 0, \perp, \text{if}(i < j, \texttt{Member}(i,k), \perp))$

7. **Membership with respect to Drop.**
   $\forall\, i, j : \texttt{INTEGER}, k : \texttt{List}\,.$
   $\texttt{Member}(i, \texttt{Drop}(j,k)) \simeq$
   $\text{if}(i < 0, \perp, \texttt{Member}(j+i, k))$

Example: Lists as Stacks 1

```
MODULE Lists;

IMPORT Stacks, Out;


(* INTERFACE *)

(*
TYPE List;
CONST Nil: List;
PROCEDURE Cons(i: INTEGER; k: List): List;
PROCEDURE Member(i: INTEGER, k: List): INTEGER;
PROCEDURE Take(i: INTEGER, k: List): List;
PROCEDURE Drop(i: INTEGER, k: List): List;
*)
```


Example: Lists as Stacks 2

```
(* IMPLEMENTATION *)


(* Types *)

TYPE List* = Stacks.Stack;


(* Constants *)

CONST Nil* = Stacks.Bottom;           (* represents the empty list *)


(* Exceptions: *)

PROCEDURE BadIndexException();
  BEGIN
    Out.String("List.BadIndexException: The index is out of range.");
    HALT(33);  (* Abort program *)
  END BadIndexException;
```


Example: Lists as Stacks 3

```
(* Interface functions defined using functions from Stacks.Mod *)

PROCEDURE Cons*(i: INTEGER; k: List): List;
  BEGIN
    RETURN Stacks.Push(i,k);
  END Cons;

PROCEDURE Member*(i: INTEGER; k: List): INTEGER;
  BEGIN
    IF (i < 0) OR Empty(k) THEN
      BadIndexException();
    ELSIF i = 0 THEN
      RETURN Stacks.Top(k);
    ELSE
      RETURN Member(i - 1, Stacks.Pop(k));
    END;
  END Member;
```

Example: Lists as Stacks 4

```
PROCEDURE Drop*(i: INTEGER; k: List): List;
  BEGIN
    IF (i < 0) THEN
      BadIndexException();
    ELSIF (i = 0) OR (k = Nil) THEN
      RETURN k;
    ELSE
      RETURN Drop(i - 1, Stacks.Pop(k));
    END;
  END Drop;
```


Example: Lists as Stacks 5

```
(* Interface functions defined without using functions from Stacks.Mod *)

PROCEDURE Take*(i: INTEGER; k: List): List;

  PROCEDURE TakeAux(i: INTEGER; k1,k2: List): List;
    BEGIN
      IF (i < 0) THEN
        BadIndexException();
      ELSIF (i = 0) OR (k2 = Nil) THEN
        RETURN Reverse(k1);
      ELSE
        RETURN TakeAux(i - 1,
                       Cons(Head(k2), k1),
                       Tail(k2));
      END;
    END TakeAux;

  BEGIN
    RETURN TakeAux(i, Nil, k);
  END Take;
```


Example: Lists as Stacks 6

```
(* Local functions *)

PROCEDURE Empty(k: List): BOOLEAN;
  BEGIN
    RETURN k = Nil;
  END Empty;

PROCEDURE Head(k: List): INTEGER;
  BEGIN
    RETURN Member(0, k);
  END Head;

PROCEDURE Tail(k: List): List;
  BEGIN
    RETURN Drop(1, k);
  END Tail;
```


Example: Lists as Stacks 7

```
PROCEDURE Reverse(k: List): List;
  VAR k1: List;
  BEGIN
    k1 := Nil;
    WHILE k # Nil DO
      k1 := Cons(Head(k), k1);
      k := Tail(k);
    END;
    RETURN k1;
  END Reverse;


END Lists.
```