# 13 Imperative Specification

1. (a) x := 1; x := 2 $\equiv$ x := 2

   (b) x := x + 1 $\equiv$ x := x + 1

   (c) **initialise**; x $\equiv$ **initialise**; 0

   (d) ((x := 1 ; x) $\equiv$ (x := 0 ; x+1)) $\equiv$ **false**

   (e) ((x := 1 ; x) = (x := 0 ; x+1)) $\equiv$ x := 0 ; **true**

2. **scheme** I_STACK1 =
   **class**
   **type** Elem
   **variable** st : Elem$^*$

   **value**
   empty : **Unit** $\rightarrow$ **write** st **Unit**
   empty() $\equiv$ st := $\langle\rangle$,

   push : Elem $\rightarrow$ **write** st **Unit**
   push(e) $\equiv$ st := $\langle$e$\rangle$ $\frown$ st,

   is_empty : **Unit** $\rightarrow$ **read** st **Bool**
   is_empty() $\equiv$ st = $\langle\rangle$,

   top : **Unit** $\stackrel{\sim}{\rightarrow}$ **read** st Elem
   top() $\equiv$ **hd** st **pre** st $\neq$ $\langle\rangle$,

   pop : **Unit** $\stackrel{\sim}{\rightarrow}$ **write** st **Unit**
   pop() $\equiv$ st := **tl** st **pre** st $\neq$ $\langle\rangle$
   **end**

3. **scheme** I_STACK2 =
   **class**
   **type** Elem
   **variable** st : Elem$^*$

   **value**
   empty : **Unit** $\rightarrow$ **write** st **Unit**
   empty() **post** st = $\langle\rangle$,

   push : Elem $\rightarrow$ **write** st **Unit**
   push(e) **post** st = $\langle$e$\rangle$ $\frown$ st\`,

   is_empty : **Unit** $\rightarrow$ **read** st **Bool**
   is_empty() **as** b **post** b = (st = $\langle\rangle$),

   top : **Unit** $\stackrel{\sim}{\rightarrow}$ **read** st Elem
   top() **as** e **post** e = **hd** st **pre** st $\neq$ $\langle\rangle$,

   pop : **Unit** $\stackrel{\sim}{\rightarrow}$ **write** st **Unit**
   pop() **post** st = **tl** st\` **pre** st $\neq$ $\langle\rangle$
   **end**

4. **scheme**
   I_STACK3 =
   **class**
   **type** Elem

   **value**
   empty : **Unit** $\rightarrow$ **write any Unit**,
   push : Elem $\rightarrow$ **write any Unit**,
   is_empty : **Unit** $\rightarrow$ **read any Bool**,
   top : **Unit** $\stackrel{\sim}{\rightarrow}$ **read any** Elem,
   pop : **Unit** $\stackrel{\sim}{\rightarrow}$ **write any Unit**

   **axiom**
   empty() ; is_empty() $\equiv$ empty() ; **true**,

   $\forall$ e : Elem • push(e) ; is_empty() $\equiv$ push(e) ; **false**,

   $\forall$ e : Elem • push(e) ; top() $\equiv$ push(e) ; e,

   $\forall$ e : Elem • push(e) ; pop() $\equiv$ **skip**
   **end**

   One can discuss whether the last axiom should be included or not. It depends on which implementations one would allow. Lists are possible implementations in both cases, but circular buffers only if the axiom is not included.

5. **scheme**
   DATABASE =
   **class**
   **type** Key, Data

   **variable** database : Key $\overrightarrow{m}$ Data

   **value**
   empty : **Unit** $\rightarrow$ **write** database **Unit**,
   insert : Key $\times$ Data $\rightarrow$ **write** database **Unit**,
   remove : Key $\rightarrow$ **write** database **Unit**,
   defined : Key $\rightarrow$ **read** database **Bool**,
   lookup : Key $\stackrel{\sim}{\rightarrow}$ **read** database Data

   **axiom**
   empty() **post** database = [ ],

   $\forall$ k : Key, d : Data • insert(k, d) **post** database = database\` $\dagger$ [k $\mapsto$ d],

   $\forall$ k : Key • remove(k) **post** database = database\` $\backslash$ {k},

   $\forall$ k : Key • defined(k) **as** b **post** b = (k $\in$ **dom** database),

   $\forall$ k : Key • lookup(k) **as** d **post** d = database(k) **pre** defined(k)
   **end**