

Pre / post-conditions — starting where (pure) functions stop

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

DI/UM, 2007 (Updated 2008, 2011)

What if invariants are not met?

- Back to the *mobile phone problem*, suppose that the requirements were (partly) misunderstood and that *store* was modelled simply as follows:

$$\text{store} : \text{Call} \rightarrow \text{ListOfCalls} \rightarrow \text{ListOfCalls}$$

$$\text{store } c \ l \triangleq c : l$$

- Clearly, *store* **fails** to preserve invariant *ListOfCalls* in case
 - $\text{length } l = 10$, or
 - $c \in \text{elems } l$, equivalent to $\langle \exists i : 1 \leq i \leq \text{length } l : l\ i = c \rangle$

NB: $\text{elems } l \triangleq \{l\ i : i \in \text{inds } l\}$ yields the set of all elements of a finite list l , where $\text{inds } l$ denotes the set of all indices of l , that is, $\text{inds } [] = \{\}$ and $\text{inds } l = \{1, \dots, \text{length } l\}$ otherwise.

What if invariants are not met?

- Back to the *mobile phone problem*, suppose that the requirements were (partly) misunderstood and that *store* was modelled simply as follows:

$$\text{store} : \text{Call} \rightarrow \text{ListOfCalls} \rightarrow \text{ListOfCalls}$$

$$\text{store } c \ l \triangleq c : l$$

- Clearly, *store* **fails** to preserve invariant *ListOfCalls* in case
 - $\text{length } l = 10$, or
 - $c \in \text{elems } l$, equivalent to $\langle \exists i : 1 \leq i \leq \text{length } l : l\ i = c \rangle$

NB: $\text{elems } l \triangleq \{l\ i : i \in \text{inds } l\}$ yields the set of all elements of a finite list l , where $\text{inds } l$ denotes the set of all indices of l , that is, $\text{inds } [] = \{\}$ and $\text{inds } l = \{1, \dots, \text{length } l\}$ otherwise.

Need for pre-conditions

- So, designers should have to **restrict** the application of *store* to input values c, l such that the invariant is preserved.
- This could be achieved by adding a **pre-condition**:

$store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls$

$store\ c\ l \triangleq c : l$

pre $length\ l < 10 \wedge c \notin elems\ l$

- Such a pre-condition is a predicate telling the range of **acceptable** input values — to be read as a *warning* provided by the designer that the function may misbehave outside such a range of values.

(Pure) functions are not enough

Thus

- *store* would become a **partial function** (clearly a symptom that the requirements had been misunderstood)

However,

- Partial functions are the rule (rather than the exception) in mathematics and computing.

Examples:

- Numbers — we know what $1/2$ means; what about $1/0$? — *division* is a partial function
- List processing: given a sequence s , what does $s\ i$ mean in case $i > \text{length } l$? — list *indexing* is a partial operation.

(Pure) functions are not enough

Thus

- *store* would become a **partial function** (clearly a symptom that the requirements had been misunderstood)

However,

- Partial functions are the rule (rather than the exception) in mathematics and computing.

Examples:

- Numbers — we know what $1/2$ means; what about $1/0$? — *division* is a partial function
- List processing: given a sequence s , what does $s\ i$ mean in case $i > \text{length } l$? — list *indexing* is a partial operation.

Pre-conditions for safety

Since

- The **meaning** of a formal model must always be a well-defined mathematical object
- One has to ensure that **no** partial function is used outside its domain of definition

the following strategy is recommended for **safety**, in presence of partial functions:

- Write your model as if all functions were total
- Chase the partial ones and add predicates to the pre-condition which ensure that all such functions are called within their domain of definition.

Pre-conditions for safety

Since

- The **meaning** of a formal model must always be a well-defined mathematical object
- One has to ensure that **no** partial function is used outside its domain of definition

the following strategy is recommended for **safety**, in presence of partial functions:

- Write your model as if all functions were total
- Chase the partial ones and add predicates to the pre-condition which ensure that all such functions are called within their domain of definition.

Pre-conditions for safety

Example: wishing to specify the operation which subtracts the first from the second element of a finite sequence of natural numbers,

$$\text{Sub21} : \mathbb{N}^* \rightarrow \mathbb{N}_0$$

$$\text{Sub21 } s \triangleq s_2 - s_1$$

we realize that the argument list is *required* to have at least two elements. So we add a pre-condition

$$\text{Sub21} : \mathbb{N}^* \rightarrow \mathbb{N}_0$$

$$\text{Sub21 } s \triangleq s_2 - s_1$$

$$\text{pre } \text{length } l \geq 2$$

Pre-conditions for safety

Example: wishing to specify the operation which subtracts the first from the second element of a finite sequence of natural numbers,

$$\text{Sub21} : \mathbb{N}^* \rightarrow \mathbb{N}_0$$

$$\text{Sub21 } s \triangleq s_2 - s_1$$

we realize that the argument list is *required* to have at least two elements. So we add a pre-condition

$$\text{Sub21} : \mathbb{N}^* \rightarrow \mathbb{N}_0$$

$$\text{Sub21 } s \triangleq s_2 - s_1$$

$$\text{pre } \text{length } l \geq 2$$

Pre-conditions for safety

However, within the natural numbers, subtraction is a partial function too. So we add another clause to the precondition:

$$\begin{aligned} \text{Sub21} &: \mathbb{N}^* \rightarrow \mathbb{N}_0 \\ \text{Sub21 } s &\triangleq s_2 - s_1 \\ \text{pre } \text{length } l &\geq 2 \wedge s_2 \geq s_1 \end{aligned} \tag{1}$$

What if the specifier decides to write clause

$$\text{pre } \text{length } l = 2 \wedge s_2 \geq s_1 \tag{2}$$

instead?

Weakest preconditions

Clearly,

- both (1) and (2) are suitable pre-conditions for *Sub21*
- (2) is **stronger** than (1), since $\text{length } l = 2 \Rightarrow \text{length } l \geq 2$
- (1) is therefore “better” than (2), as the latter restricts the applicability of *Sub21* too much.

In fact,

- predicate (1) is the **weakest pre-condition** (WP) for *Sub21* to be safe
- one should aim at always *stopping* at WPs.

(We will learn later how to **calculate** WPs.)

Need for more

When studying **probability** theory and **statistics** one is faced with problems such as the following:

*One is picking up marbles from a bag initially with a **red**, a **blue** and a **yellow** marble. Compute the probability of the experiment in which **red** is picked first, **yellow** second and **blue** third.*

Suppose you want to build an abstract model of a program you want to run as much as possible to confirm the theory:

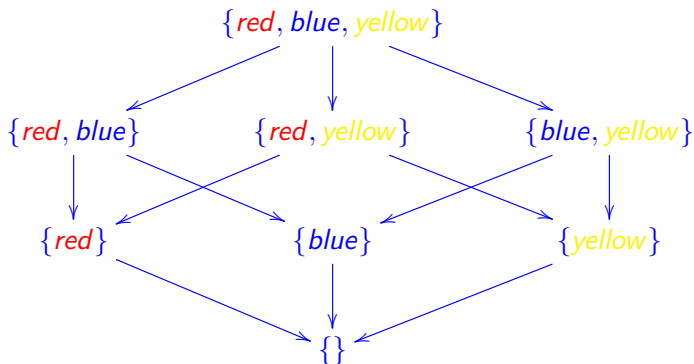
- Datatypes:

$$\text{Marble} = \{\text{red}, \text{blue}, \text{yellow}\}$$
$$\text{Bag} = \{B : B \subseteq \text{Marble}\}$$

NB: one may alternatively write $\text{Bag} = \mathcal{P}\text{Marble}$, see next slide.

Need for more

The extension of *Bag* is as follows:



This is known as the **powerset lattice** of set *Marble*.

Need for more

- Operations: one needs the operation which puts all marbles back into the bag

$reset : Bag \rightarrow Bag$

$reset\ b \triangleq \{red, blue, yellow\}$

and another to simulate the experiment of picking the next marble:

$Pick : Bag \rightarrow (Marble \times Bag)$

$Pick\ b \triangleq \dots$

However, for the experiment to be valid, the choice of the next marble to pick must be **non-deterministic**: *Pick* is **not** a function!

Need for more

- Operations: one needs the operation which puts all marbles back into the bag

$reset : Bag \rightarrow Bag$

$reset\ b \triangleq \{red, blue, yellow\}$

and another to simulate the experiment of picking the next marble:

$Pick : Bag \rightarrow (Marble \times Bag)$

$Pick\ b \triangleq \dots$

However, for the experiment to be valid, the choice of the next marble to pick must be **non-deterministic**: *Pick* is **not** a function!

Post-conditions for liveness

- Let x denote the marble obtained from bag b
- Let r denote b without such a marble
- The best we can say about the experiment is

$$x \in b \wedge r = b - \{x\}$$

assuming $b \neq \{\}$.

We are led to a specification based on a **pre-/post**-condition pair:

Pick : $(x : \text{Marble}, r : \text{Bag}) \leftarrow (b : \text{Bag})$

pre $b \neq \{\}$

post $x \in b \wedge r = b - \{x\}$

Post-conditions for vagueness

- Another use of **pre-/post-** pairs is that of tolerating more than one result
- Example: we want to specify “*the function*” **square root** of an integer:

$Sqrt : (r : \mathbb{R}) \leftarrow (i : \mathbb{Z})$

pre $i \geq 0$

post $r^2 = i$

The specifier is telling the implementer that either solution $r = +\sqrt{i}$ or $r = -\sqrt{i}$ will do.

Post-conditions for implicit specification

- Post-conditions — elegant way of *hiding* algorithmic details which a particular function always embodies.
- Wherever we write a post-condition bearing in mind to specify a function f , we refer to such a condition as an **implicit specification** of f .

Example: **explicit** definition of abs function

$$abs : \mathbb{Z} \rightarrow \mathbb{N}$$
$$abs\ i \triangleq \text{if } i < 0 \text{ then } -i \text{ else } i$$

followed by **implicit** definition of the same function:

$$abs : (i : \mathbb{Z}) \rightarrow (r : \mathbb{N})$$
$$\text{post } r \geq 0 \wedge (r = i \vee r = -i)$$

Post-conditions for implicit specification

- Post-conditions — elegant way of *hiding* algorithmic details which a particular function always embodies.
- Wherever we write a post-condition bearing in mind to specify a function f , we refer to such a condition as an **implicit specification** of f .

Example: **explicit** definition of abs function

$$abs : \mathbb{Z} \rightarrow \mathbb{N}$$

$$abs\ i \triangleq \text{if } i < 0 \text{ then } -i \text{ else } i$$

followed by **implicit** definition of the same function:

$$abs : (i : \mathbb{Z}) \rightarrow (r : \mathbb{N})$$

$$\text{post } r \geq 0 \wedge (r = i \vee r = -i)$$

Examples

Explicit definition of *max* function

$$\begin{aligned} \text{max} &: (\mathbb{Z} \times \mathbb{Z}) \rightarrow \mathbb{Z} \\ \text{max}(i, j) &\triangleq \text{if } i \leq j \text{ then } j \text{ else } i \end{aligned} \quad (3)$$

and now its **implicit** specification:

$$\begin{aligned} \text{max} &: (i : \mathbb{Z}, j : \mathbb{Z}) \rightarrow (r : \mathbb{Z}) \\ \text{post } &r \in \{i, j\} \wedge i \leq r \wedge j \leq r \end{aligned} \quad (4)$$

Now the implicit specification of a partial function:

$$\begin{aligned} \text{Maxs} &: (s : \mathcal{P}\mathbb{N}) \rightarrow (r : \mathbb{N}) \\ \text{pre } &s \neq \{\} \\ \text{post } &r \in s \wedge \langle \forall i : i \in s : i \leq r \rangle \end{aligned}$$

A glimpse at deriving **explicit** from **implicit**

The “best” specification of *max* is as follows, cf. its post-condition:

$$\mathit{max}(i,j) \leq r \equiv i \leq r \wedge j \leq r \quad (5)$$

Let us calculate *explicit* definition (6) from (5):

- Case $i \leq j = \text{TRUE}$:

$$\begin{aligned} & \mathit{max}(i,j) \leq r \equiv i \leq r \wedge j \leq r \\ = & \quad \{ i \leq r \Leftarrow i \leq j \wedge j \leq r \} \\ & \mathit{max}(i,j) \leq r \equiv j \leq r \\ \therefore & \quad \{ \text{indirect equality (more about this later on...)} \} \\ & \mathit{max}(i,j) = j \end{aligned}$$

A glimpse at deriving **explicit** from **implicit**

- Case $j < i = \text{TRUE}$:

$$\begin{aligned} & \max(i,j) \leq r \equiv i \leq r \wedge j \leq r \\ = & \quad \{ j \leq r \Leftarrow j < i \wedge i \leq r \} \\ & \max(i,j) \leq r \equiv i \leq r \\ :: & \quad \{ \text{indirect equality (more about this later on...)} \} \\ & \max(i,j) = i \end{aligned}$$

Putting both cases together:

$$\max(i,j) \triangleq \text{if } i \leq j \text{ then } j \text{ else } i$$

Post-conditions for relational specification

We want to specify the **prefix** relation between two finite sequences, eg.

$[1, 2] \text{ IsPrefixOf } [1, 2, 4, 1]$
 $[] \text{ IsPrefixOf } []$
 $[] \text{ IsPrefixOf } [1]$

We write:

$\text{IsPrefixOf} : (r : A^*) \leftarrow (s : A^*)$

post $\text{length } r \leq \text{length } s \wedge (\forall i : i \leq \text{length } r : (r\ i) = (s\ i))$

NB: note that this spec is parametric on A .

Post-conditions for relational specification

We want to specify the **prefix** relation between two finite sequences, eg.

$[1, 2] \text{ IsPrefixOf } [1, 2, 4, 1]$

$[] \text{ IsPrefixOf } []$

$[] \text{ IsPrefixOf } [1]$

We write:

$\text{IsPrefixOf} : (r : A^*) \leftarrow (s : A^*)$

post $\text{length } r \leq \text{length } s \wedge \langle \forall i : i \leq \text{length } r : (r\ i) = (s\ i) \rangle$

NB: note that this spec is parametric on A .

Post-conditions for relational specification

Another example: the relation which expresses sequence permutation:

$$\begin{aligned} \text{Permutes} &: (r : A^*) \leftarrow (s : A^*) && (6) \\ \text{post} & \langle \forall e : e \in \text{elems } s \cup \text{elems } r : \text{count } e \text{ } s = \text{count } e \text{ } r \rangle \end{aligned}$$

assuming

$$\begin{aligned} \text{count} &: A \rightarrow A^* \rightarrow \mathbb{N}_0 \\ \text{count } a \text{ } s &\triangleq \text{card}\{i : i \in \text{inds } s \wedge (s \text{ } i) = a\} \end{aligned}$$

where $\text{card} : \mathcal{P}A \rightarrow \mathbb{N}_0$ computes the number of elements of a set.

Example: sorting

- The following implicit specification of **sorting** abstracts from the particular algorithm one has in mind:

$$\begin{aligned} & \textit{Sort} : (r : A^*) \leftarrow (s : A^*) \\ & \mathbf{post} \quad \textit{isOrdered} \ r \wedge r \ \textit{Permutes} \ s \end{aligned} \quad (7)$$

- Predicate *isOrdered* (to be studied later on) assumes a total order on datatype *A*.

Exercises

Exercise 1: Give an implicit definition for function $f \ x \triangleq x^2 + 1$ over the natural numbers.



Exercise 2: Write implicit and explicit specifications for function $inseq : \mathbb{N}_0 \rightarrow \mathbb{N}^*$ which, for argument n , yields the sequence $[1, \dots, n]$.



Exercise 3: Write an explicit definition of (partial) function $Maxs$.



Notation

By writing

$$Spec : (b : B) \leftarrow (a : A)$$

pre ...

post ...

we mean the definition of two predicates

$$pre-Spec : A \rightarrow \mathbb{B}$$
$$post-Spec : B \times A \rightarrow \mathbb{B}$$

such that

$$\langle \forall a : a \in A : pre-Spec\ a \Rightarrow \langle \exists b : b \in B : post-Spec(b, a) \rangle \rangle \quad (8)$$

Proof obligation: satisfiability

Thus (8) is another proof obligation:

- **satisfiability**: ensure that *pre-Spec* and *post-Spec* are such that, for all acceptable inputs, there must be some possible result.
- this includes the situation in which *A* and *B* have invariants.

Exercise 4: Assuming that the implicit definition of a total function $B \xleftarrow{f} A$ uniquely determines f , that is

$$\text{post-}f(r, a) \equiv r = f a \quad (9)$$

holds, use the Eindhoven quantifier calculus to show that (8) reduces to

$$\langle \forall a : a \in A : (f a) \in B \rangle$$

for $\text{Spec} := f$. In summary: in the case of functions, *satisfiability* is the same as *invariant preservation*.

□

Exercises

Exercise 5: Add an invariant to

$$NRSeq\ A = A^*$$

inv ...

preventing repeated elements. Then tell which of the following new types for *Permutes* (6),

$$Permutes : (r : NRSeq\ A) \leftarrow (s : A^*) \quad (10)$$

$$Permutes : (r : NRSeq\ A) \leftarrow (s : NRSeq\ A) \quad (11)$$

will lead to a non satisfiable specification.

