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

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

DI/UM, 2007 (Updated 2008, 2011-12)

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:

$$store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls$$
$$store\ c\ l \triangleq c : l$$

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

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

$$store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls$$
$$store\ c\ l \triangleq c : l$$

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

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

Need for pre-conditions

- So, designers would 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**:

$$store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls$$
$$store\ c\ l \triangleq c : l$$
$$\textbf{pre}\ length\ l < 10 \land c \notin elems\ l$$

- Such a pre-condition is a predicate telling a 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 $i > length\ l$? — 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 $i > length\ l$? — 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,

$$Sub21 : \mathbb{N}^{\star} \rightarrow \mathbb{N}_0$$
$$Sub21\ s \triangleq\ s\ 2 - s\ 1$$

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

$$Sub21 : \mathbb{N}^{\star} \rightarrow \mathbb{N}_0$$
$$Sub21\ s \triangleq\ s\ 2 - s\ 1$$
$$\textbf{pre}\ length\ l \geq 2$$

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,

$$Sub21 : \mathbb{N}^\star \to \mathbb{N}_0$$
$$Sub21\ s \triangleq\ s\ 2 - s\ 1$$

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

$$Sub21 : \mathbb{N}^\star \to \mathbb{N}_0$$
$$Sub21\ s \triangleq\ s\ 2 - s\ 1$$
$$\textbf{pre}\ length\ l \geq 2$$

Pre-conditions for safety

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

$$
\begin{aligned}
& Sub21 : \mathbb{N}^{\star} \rightarrow \mathbb{N}_0 \\
& Sub21\ s \triangleq\ s\ 2 - s\ 1 \\
& \mathbf{pre}\ length\ l \geq 2\ \wedge\ s\ 2 \geq s\ 1
\end{aligned}
\tag{15}
$$

What if the specifier decides to write clause

$$
\mathbf{pre}\ length\ l = 2\ \wedge\ s\ 2 \geq s\ 1 \tag{16}
$$

instead?

Weakest preconditions

Clearly,

- both (15) and (16) are suitable pre-conditions for *Sub*21
- (16) is **stronger** than (15), since $length\ l = 2 \ \Rightarrow\ length\ l \geq 2$
- (15) is therefore "better" than (16), as the latter restricts the applicability of *Sub*21 too much.

It turns out that

- predicate (15) is the **weakest pre-condition** (WP) for *Sub*21 to be safe
- one should aim at always *stopping* at WPs.

We will learn later how to **calculate** WPs. A thumb rule is given in the next slide for a special (in fact, easiest) case.

Weakest preconditions

Let $f : X \to Y$ be a function where type Y is constrained by an invariant, $inv\text{-}Y : Y \to \mathbb{B}$. Then the **weakest pre-condition** to be enforced on f with respect to $inv\text{-}Y$ is

$$wp(f, inv\text{-}Y)\ x \quad \triangleq \quad inv\text{-}Y(f\ x) \tag{17}$$

Exercise 8: Calculate the weakest precondition $wp(f, inv\text{-}Y)$ for each situation below:

X	Y	f	$inv\text{-}Y$
$\mathbb{N}_0$	$\mathbb{N}$	$f\ x \triangleq x^2 + 1$	$1 \leq y$
$\mathbb{N}_0$	$\mathbb{N}$	the same	$y \leq 10$
$\mathbb{N}_0$	$\mathbb{N}$	$f = succ$	$even\ y$
$\mathbb{N} \times \mathbb{N}^\star$	$\mathbb{N}^\star$	$f(n, x) \triangleq n : x$	$\langle \forall\ m\ :\ m \in\ elems\ y :\ m \leq 10 \rangle$

$\square$

Weakest preconditions

Exercise 9: Indicate which predicates p below are stronger (or weaker) than the weakest precondition (WP) on each f with respect to the corresponding output invariant:

X	Y	f	inv-$Y(y)$	$p(x)$
$\mathbb{R}$	$\mathbb{R}$	$f\ x \triangleq x^2 + 1$	$0 \leq y \leq 10$	$0 < x < 3$
$\mathbb{N}^\star$	$\mathbb{N}^\star$	$f = map\ \underline{1}$	$\langle \forall\ i\ :\ i \in \text{inds}\, y :\ y\ i > 10 \rangle$	$\textsc{True}$
$A^\star$	$A^\star$	$f = tail$	$length\ y > 0$	$x \neq [\,]$
$BTree\ A$	$BTree\ A$	$f = mirror$	$depth\ y \geq 1$	$depth\ x > 1$

where *map* and *tail* are well known list operators and *mirror* and *depth* are the obvious functions over binary trees.

□

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:

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

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

Need for more

The extension of *Bag* is as follows:

$$
\begin{tikzcd}
 & & \{red, blue, yellow\} \arrow[dll] \arrow[d] \arrow[drr] & & \\
\{red, blue\} \arrow[d] \arrow[drr] & & \{red, yellow\} \arrow[dll] \arrow[drr] & & \{blue, yellow\} \arrow[dll] \arrow[d] \\
\{red\} \arrow[drr] & & \{blue\} \arrow[d] & & \{yellow\} \arrow[dll] \\
 & & \{\} & &
\end{tikzcd}
$$

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

$$reset : Bag \to Bag$$
$$reset\ b \triangleq \{red, blue, yellow\}$$

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

$$Pick : Bag \to (Marble \times Bag)$$
$$Pick\ b \triangleq \ \ldots$$

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

$$reset : Bag \to Bag$$
$$reset\ b \triangleq \{red, blue, yellow\}$$

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

$$Pick : Bag \to (Marble \times Bag)$$
$$Pick\ b \triangleq \ \ldots$$

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 a marble to be taken 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:

$$\begin{array}{l} \mathit{Pick} : (x : \mathit{Marble}, r : \mathit{Bag}) \leftarrow (b : \mathit{Bag}) \\ \textbf{pre} \ \ b \neq \{\} \\ \textbf{post} \ \ x \in b \ \wedge \ r = b - \{x\} \end{array}$$

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:

$$
\begin{array}{l}
Sqrt : (r : \mathbb{R}) \leftarrow (i : \mathbb{Z}) \\
\textbf{pre } i \geq 0 \\
\textbf{post } \; r^2 = i
\end{array}
$$

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

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

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

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

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

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

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

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

Examples

Explicit definition of *max* function

$$max : (\mathbb{Z} \times \mathbb{Z}) \to \mathbb{Z}$$
$$max(i, j) \triangleq \ \ \textbf{if } i \leq j \textbf{ then } j \textbf{ else } i \qquad (18)$$

followed by its **implicit** specification:

$$max : (i : \mathbb{Z}, j : \mathbb{Z}) \to (r : \mathbb{Z})$$
$$\textbf{post} \ \ r \in \{i, j\} \wedge i \leq r \wedge j \leq r \qquad (19)$$

Now the implicit specification of a partial function:

$$Maxs : (s : \mathcal{P}\mathbb{N}) \to (r : \mathbb{N})$$
$$\textbf{pre } s \neq \{\}$$
$$\textbf{post} \ \ r \in s \wedge \langle \forall\ i \ : \ i \in s : \ i \leq r \rangle$$

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

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

$$max(i,j) \leq r \ \equiv \ i \leq r \wedge j \leq r \qquad (20)$$

Let us calculate *explicit* definition (18) from (20):

- Case $i \leq j = \textsc{True}$:

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

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

- Case $j \leq i = \textsc{True}$:

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

Putting both cases together:

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

Post-conditions for relational specification

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

$$
\begin{array}{rcl}
[1,2] & \mathit{IsPrefixOf} & [1,2,4,1] \\
[] & \mathit{IsPrefixOf} & [] \\
[] & \mathit{IsPrefixOf} & [1]
\end{array}
$$

We write:

$$\mathit{IsPrefixOf} : (s : A^{\star}) \leftarrow (r : A^{\star})$$

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

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.

$$\begin{array}{rll} [1,2] & \mathit{IsPrefixOf} & [1,2,4,1] \\ [] & \mathit{IsPrefixOf} & [] \\ [] & \mathit{IsPrefixOf} & [1] \end{array}$$

We write:

$$\begin{array}{l} \mathit{IsPrefixOf} : (s : A^\star) \leftarrow (r : A^\star) \\ \textbf{post} \quad \mathit{length}\ s \leq \mathit{length}\ r \wedge \langle \forall\ i\ :\ i \leq \mathit{length}\ s :\ (s\ i) = (r\ i) \rangle \end{array}$$

NB: note that this spec is parametric on A.

Post-conditions for relational specification

Another example: the relation which expresses sequence permutation:

$$\begin{aligned} &\mathit{Permutes} : (s : A^\star) \leftarrow (r : A^\star) \\ &\textbf{post} \ \ \langle \forall\ a \ : \ a \in \mathit{elems}(s +\!\!+\ r) : \ \mathit{count}\ a\ s = \mathit{count}\ a\ r \rangle \end{aligned} \tag{21}$$

assuming

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

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

Example: sorting

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

$$\begin{array}{l} Sort : (s : A^\star) \leftarrow (r : A^\star) \\ \textbf{post} \;\; isOrdered(\leq)s \wedge s \; Permutes \; r \end{array} \tag{22}$$

As seen in the following exercise, predicate $isOrdered$ assumes a total order $(\leq)$ on datatype A.

Exercise 10: Complete the following (inductive) specification of $isOrdered$:

$$\begin{array}{l} isOrdered(\leq)[\,] = \textsc{True} \\ isOrdered(\leq)(a : x) = \ldots isOrdered(\leq)x \ldots \end{array}$$

□

Exercises

Exercise 11: Give an implicit definition for function $f\ x \triangleq\ x^2 + 1$ over the natural numbers.

$\square$

Exercise 12: A *golden multiple* of a given length is obtained by multiplying this length by a real number whose square equals its "successor". Write an implicit specification for *golden multiple*.

$\square$

Exercise 13: Write implicit and explicit specifications for function $inseq : \mathbb{N}_0 \to \mathbb{N}^\star$ which, for argument n, yields the sequence $[1, \ldots, n]$.

$\square$

The **pre/pos/inv** trilogy

By writing

$$
\begin{array}{l}
\mathit{Spec} : (b : B) \leftarrow (a : A) \\
\textbf{pre} \;\; \ldots \\
\textbf{post} \;\; \ldots
\end{array}
$$

we mean the definition of two predicates

$$
\begin{array}{l}
\text{pre-}\mathit{Spec} : A \rightarrow \mathbb{B} \\
\text{post-}\mathit{Spec} : B \times A \rightarrow \mathbb{B}
\end{array}
$$

such that

$$
\langle \forall\ a \ : \ a \in A : \ \text{pre-}\mathit{Spec}\ a \Rightarrow \langle \exists\ b \ : \ b \in B : \ \text{post-}\mathit{Spec}(b, a) \rangle \rangle \tag{23}
$$

Proof obligation: satisfiability

Thus (23) is another proof obligation known as **satisfiability**:

> **Satisfiability** *ensures 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 14: Assuming that the implicit definition of a total function $B \xleftarrow{f} A$ uniquely determines f, that is

$$\text{post-}f(r, a) \quad \equiv \quad r = f\ a \tag{24}$$

holds, use the Eindhoven quantifier calculus to show that (23) reduces to $\langle \forall\ a\ :\ a \in A :\ (f\ a) \in B \rangle$ for $Spec := f$. In summary: in the case of functions, *satisfiability* is the same as *invariant preservation*.

□

Exercises

Exercise 15: Consider datatype

$$NRSeq\ A = A^\star$$
$$\textbf{inv}\ \ x \triangleq\ length\ x = card(elems\ x)$$

1. What is the informal meaning of the type's invariant?
2. Tell which of the following new types for *Permutes* (21),

$$Permutes : (s : NRSeq\ A) \leftarrow (r : A^\star) \tag{25}$$
$$Permutes : (s : NRSeq\ A) \leftarrow (r : NRSeq\ A) \tag{26}$$

would lead to a non satisfiable specification.

□

Exercises

Exercise 16: Back to

$$Permutes : (s : A^\star) \leftarrow (r : A^\star)$$
$$\textbf{post} \quad \langle \forall\ a \ :\ a \in elems(s +\!\!+\ r) :\ count\ a\ s = count\ a\ r \rangle$$

show that

1. $Permutes$ is a **reflexive** relation: $x\ Permutes\ x \equiv \text{TRUE}$ for all x.
2. $Permutes$ is a **symmetric** relation: $y\ Permutes\ x \equiv x\ Permutes\ y$ for all x, y.

□

Exercise 17: How would you write an explicit definition of (partial) function $Maxs$?

□

Background — Eindhoven quantifier calculus (cont.)

Splitting:

$$\langle \forall\, k : R \vee S : T \rangle = \langle \forall\, k : R : T \rangle \wedge \langle \forall\, k : S : T \rangle \quad (27)$$

$$\langle \exists\, k : R \vee S : T \rangle = \langle \exists\, k : R : T \rangle \vee \langle \exists\, k : S : T \rangle \quad (28)$$

Rearranging:

$$\langle \forall\, k : R : T \wedge S \rangle = \langle \forall\, k : R : T \rangle \wedge \langle \forall\, k : S : T \rangle \quad (29)$$

$$\langle \exists\, k : R : T \vee S \rangle = \langle \exists\, k : R : T \rangle \vee \langle \exists\, k : R : S \rangle \quad (30)$$

de Morgan:

$$\neg\langle \forall\, k : R : T \rangle = \langle \exists\, k : R : \neg T \rangle \quad (31)$$

$$\neg\langle \exists\, k : R : T \rangle = \langle \forall\, k : R : \neg T \rangle \quad (32)$$