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

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

DI/UM, 2007 (Last update: Oct. 2014)

Requirements → invariants

Recall:

> *(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) each list stores up to 10 entries.*

Clause (c) leads to invariant

$$ListOfCalls = Call^{\star}$$
$$\mathbf{inv(c)}\ s \triangleq\ length\ s \leq 10$$

Clause (b) leads to invariant

$$\mathbf{inv(b)}\ s \triangleq\ \langle \forall\ i, j \leq length\ s\ :\ l\ i = l\ j :\ i = j \rangle$$

What if invariants are not met?

Suppose that *store* is modelled as simply as follows, in a first attempt:

$$store : Call \to ListOfCalls \to ListOfCalls$$
$$store\ c\ s \triangleq\ c : s$$

Clearly, *store* **fails** to preserve invariant *ListOfCalls* in case

- $length\ s = 10$, or
- $c \in elems\ s$, equivalent to $\langle \exists\ i\ :\ 1 \leq i \leq length\ s :\ s\ i = c \rangle$

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

What if invariants are not met?

Suppose that *store* is modelled as simply as follows, in a first attempt:

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

Clearly, *store* **fails** to preserve invariant *ListOfCalls* in case

- $length\ s = 10$, or
- $c \in elems\ s$, equivalent to $\langle \exists\ i\ :\ 1 \leq i \leq length\ s :\ s\ i = c \rangle$

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

Need for pre-conditions

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

$$
\begin{array}{l}
store : Call \rightarrow ListOfCalls \rightarrow ListOfCalls \\
store\ c\ s \triangleq\ c : s \\
\mathbf{pre}\ length\ s < 10 \land c \notin elems\ s
\end{array}
$$

- 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 have only been partly understood)

In practice

- 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 $i > length\ s$? — list *indexing* is a **partial** operation.

(Pure) functions are not enough

Thus

- *store* would become a **partial function** (clearly a symptom that the requirements have only been partly understood)

In practice

- 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 $i > \mathit{length}\ s$? — list *indexing* is a **partial** operation.

Pre-conditions for safety

Because

- the formal **meaning** of a program should always be a well-defined mathematical object
- one has to ensure that **no** partial function is called 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 **pre-conditions** ensuring that all such functions are called within their domain of definition.

Pre-conditions for safety

Because

- the formal **meaning** of a program should always be a well-defined mathematical object
- one has to ensure that **no** partial function is called 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 **pre-conditions** ensuring 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,

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

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

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

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,

$$\mathit{Sub21} : \mathbb{N}^\star \longrightarrow \mathbb{N}$$
$$\mathit{Sub21}\ s \triangleq\ s\ 2 - s\ 1$$

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

$$\mathit{Sub21} : \mathbb{N}^\star \longrightarrow \mathbb{N}$$
$$\mathit{Sub21}\ s \triangleq\ s\ 2 - s\ 1$$
$$\textbf{pre}\ \mathit{length}\ s \geq 2$$

Pre-conditions for safety

However, subtraction in $\mathbb{N}$ is a partial function too (e.g. $1 - 2$ in not in $\mathbb{N}$). So we add another clause to the pre-condition:

$$\begin{aligned}
&Sub21 : \mathbb{N}^{\star} \to \mathbb{N} \\
&Sub21\ s \triangleq\ s\ 2 - s\ 1 \\
&\textbf{pre}\ length\ s \geq 2\ \wedge\ s\ 2 > s\ 1
\end{aligned} \tag{26}$$

What if the specifier decides to write clause

$$\textbf{pre}\ length\ s = 2\ \wedge\ s\ 2 > s\ 1 \tag{27}$$

instead?

Weakest preconditions

Clearly,

- both (26) and (27) are suitable pre-conditions for $Sub21$
- (27) is **stronger** than (26), since $length\ l = 2 \ \Rightarrow\ length\ l \geq 2$
- (26) is therefore "better" than (27), as the latter restricts the use of $Sub21$ too much.

It turns out that

- predicate (26) is the **weakest pre-condition** (WP) for $Sub21$ to be **safe**
- one should aim at always *specifying* 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 $f : X \rightarrow Y$ be a function where type Y is constrained by an invariant, $\text{inv-}Y : Y \rightarrow \mathbb{B}$. Then the **weakest pre-condition** to be enforced on f with respect to $\text{inv-}Y$ is

$$wp(f, \text{inv-}Y)\ x \quad \triangleq \quad \text{inv-}Y(f\ x) \tag{28}$$

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

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

$\square$

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

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:

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

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

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

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

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

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

$$\begin{aligned} reset &: Bag \longrightarrow Bag \\ reset\ b &\triangleq \{red, blue, yellow\} \end{aligned}$$

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

$$\begin{aligned} Pick &: Bag \longrightarrow (Marble \times Bag) \\ Pick\ b &\triangleq \ \ldots \end{aligned}$$

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

Let

- x denote a marble to be taken from bag b
- 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:

$$\begin{array}{l} Pick : (x : Marble, r : Bag) \leftarrow (b : Bag) \\ \textbf{pre } b \neq \{\} \\ \textbf{post } \; x \in b \;\wedge\; r = b - \{x\} \end{array} \tag{29}$$

Vague requirements

Another use of **pre-/post-** pairs is for tolerating more than one result (vagueness).

Example: we want to specify *"the function"* **square root** of an integer:

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

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

Implicit specifications

Post-conditions are also an elegant way of **hiding** algorithmic details which a particular function always embodies.

Wherever post-condition is intended to specify a function f, we refer to such a condition as an **implicit specification** of f.

Example: **explicit** definition of *abs* function

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

followed by an **implicit specification** of the same function:

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

Implicit specifications

Post-conditions are also an elegant way of **hiding** algorithmic details which a particular function always embodies.

Wherever post-condition is intended to specify a function f, we refer to such a condition as an **implicit specification** of f.

Example: **explicit** definition of abs function

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

followed by an **implicit specification** of the same function:

$$abs : (i : \mathbb{Z}) \rightarrow (r : \mathbb{Z})$$
$$\mathbf{post} \quad r \geq 0 \land (r = i \lor r = -i)$$

Examples

Explicit definition of *max* function

$$\begin{aligned} &max : (\mathbb{Z} \times \mathbb{Z}) \to \mathbb{Z} \\ &max(i,j) \triangleq \ \ \textbf{if } i \leq j \textbf{ then } j \textbf{ else } i \end{aligned} \tag{30}$$

followed by its **implicit specification**:

$$\begin{aligned} &max : (i : \mathbb{Z}, j : \mathbb{Z}) \to (r : \mathbb{Z}) \\ &\textbf{post} \ \ r \in \{i,j\} \wedge i \leq r \wedge j \leq r \end{aligned} \tag{31}$$

Now the implicit specification of a **partial function**:

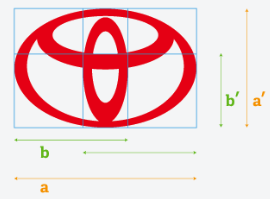
$$\begin{aligned} &Maxs : (s : \mathcal{P}\mathbb{N}) \to (r : \mathbb{N}) \\ &\textbf{pre } s \neq \{\} \\ &\textbf{post} \ \ r \in s \wedge \langle \forall\, i \ : \ i \in s : \ i \leq r \rangle \end{aligned}$$

Exercises

Exercise 10: Give an implicit definition for function $f\ x \triangleq x^2 + 1$ over the natural numbers.

□

Exercise 11:
A **golden multiple** of a given dimension a is another dimension a' obtained by multiplying a by a real number whose square equals its "successor". Write a post-condition for $GoldenMultiple : (a' : \mathbb{R}) \leftarrow (a : \mathbb{R})$.



□

Exercise 12: Write implicit and explicit specifications for function $inseq : \mathbb{N}_0 \to \mathbb{N}^\star$ which, for argument n, yields the sequence $[1, \ldots, n]$.

□

The **inv/pre/post** trilogy

By writing specification the specification S as a **pre/post** pair,

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

we mean the definition of two **predicates**

$$
\begin{array}{l}
\text{pre-}S : A \to \mathbb{B} \\
\text{post-}S : B \times A \to \mathbb{B}
\end{array}
$$

which **preserve** the **invariants** of A and B:

$$
\langle \forall\ a, b \ :\ a \in A \wedge \text{pre-}S\ a \wedge \text{post-}S(b, a) :\ b \in B \rangle \tag{32}
$$

Satisfiability

The following condition, known as **satisfiability**,

$$\langle \forall\ a\ :\ a \in A:\ \text{pre-}S\ a \Rightarrow \langle \exists\ b\ :\ b \in B:\ \text{post-}S(b, a)\rangle\rangle \tag{33}$$

ensures the consistency of a **pre**/**post** specification.

A non-satisfiable **pre**/**post** pair is *pathological* in the sense that

- for some **valid** a meeting **pre**
- **post** is unable to produce any **valid** output b.

Thus any **program** derived from the spec is doomed to **fail** for such inputs (!)

Safety and liveness

In a computer system one always wishes that

- **bad** things **never** happen
- **good** things **eventually** happen.

Terminology:

- *bad things never happen* — **safety**
- *good things eventually happen* — **liveness**

Formally:

- **Invariant** preservation (32) ensures **safety**
- **Satisfiability** (33) ensures **liveness**

Exercises

Exercise 13: Given $A = \mathbb{Z}_0$ and $B = \mathbb{Z}_0$ such that $\textsf{inv-}B\ b = b > 0$, is

$$S : (b : B) \leftarrow (a : A)$$
$$\textbf{post}\ \ b = a + 1 \lor b + 1 = a$$

invariant preserving (32)? If not, find a pre-condition for this to happen.

□

Exercise 14: Show that, without the pre-condition $b \neq \emptyset$, $Pick$ (29)

$$Pick : (x : Marble, r : Bag) \leftarrow (b : Bag)$$
$$\textbf{post}\ \ x \in b\ \land\ r = b - \{x\}$$

is **not** satisfiable. **Hint**: negate (33).

□

Exercises

Exercise 15: Assuming that the implicit definition of a total function $B \xleftarrow{f} A$ uniquely determines f, that is

$$\text{post-}f(r, a) \quad \equiv \quad r = f \ a \tag{34}$$

holds, use the Eindhoven quantifier calculus to show that (33) reduces to $\langle \forall\ a\ :\ a \in A :\ (f\ a) \in B \rangle$ for $S := f$. In summary: in the case of functions, **satisfiability** is the same as **invariant preservation.**

□