

Algorithmic Specification

Carlos Cotta

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

<http://www.lcc.uma.es/~ccottap>

Comput Eng, Softw Eng, Comput Sci & Math – 2023-2024



UNIVERSIDAD
DE MÁLAGA

C. Cotta

Algorithmic Specification

1 / 34

Introduction
Pre/Post Specification and Semantics
Conclusions

Specification vs Implementation

The **specification** of an algorithm is a description of **what it does** and **under which conditions** it does it.

The **implementation** of an algorithm is a description of a **sequence of instructions fulfilling the specification**. Hence, specification is a required stage previous to implementation.

A particular specification –using a certain logical formalism– allows verifying whether the implementation performs its intended purpose (**verification**).

C. Cotta

Algorithmic Specification

4 / 34

Algorithmic Specification

1 Introduction

2 Pre/Post Specification and Semantics

- Pre/Post Specification
- Semantics

3 Conclusions

- Outlook

C. Cotta

Algorithmic Specification

2 / 34

Introduction
Pre/Post Specification and Semantics
Conclusions

Specification vs Implementation

Assume that given two integers $a \geq 0$ and $b > 0$, we wish to find their quotient q and remainder r . A potential specification might be:

Integer division – pre/post specification

$\{a \geq 0, b > 0\}$

Division algorithm // Implementation here

$\{a = qb + r\}$

The specification is described by mean of logical **asserts** that can take a Boolean value (true or false).

The specification is important for **program users** (the programmer that wishes to use the program and/or other programs that use it).

C. Cotta

Algorithmic Specification

5 / 34

Motivational Example of Formal Specification

A potential implementation:

Integer division

 $\{a \geq 0, b > 0\}$

```
(1)  $q \leftarrow 0$ 
(2)  $r \leftarrow a$ 
(3) while  $r > b$  do
(4)    $r \leftarrow r - b$ 
(5)    $q \leftarrow q + 1$ 
(6) endwhile
```

 $\{a = qb + r\}$

Program states before running each line

 $a = 5, b = 2$

```
(3)  $a = 5, b = 2, q = 0, r = 5$ 
(4)  $a = 5, b = 2, q = 0, r = 5$ 
(5)  $a = 5, b = 2, q = 0, r = 3$ 
(3)  $a = 5, b = 2, q = 1, r = 3$ 
(4)  $a = 5, b = 2, q = 1, r = 3$ 
(5)  $a = 5, b = 2, q = 1, r = 1$ 
(3)  $a = 5, b = 2, q = 2, r = 1$ 
 $a = 5, b = 2, q = 2, r = 1$ 
```

Motivational Example of Formal Specification

Let us modify the loop condition and the final assert:

Integer division

 $\{a \geq 0, b > 0\}$

```
(1)  $q \leftarrow 0$ 
(2)  $r \leftarrow a$ 
(3) while  $r \geq b$  do
(4)    $r \leftarrow r - b$ 
(5)    $q \leftarrow q + 1$ 
(6) endwhile
```

 $\{a = qb + r, r < b\}$

Program states before running each line

 $a = 6, b = 2$

```
(3)  $a = 6, b = 2, q = 0, r = 6$ 
(4)  $a = 6, b = 2, q = 0, r = 6$ 
(5)  $a = 6, b = 2, q = 0, r = 4$ 
(3)  $a = 6, b = 2, q = 1, r = 4$ 
(4)  $a = 6, b = 2, q = 1, r = 4$ 
(5)  $a = 6, b = 2, q = 1, r = 2$ 
(3)  $a = 6, b = 2, q = 2, r = 2$ 
(4)  $a = 6, b = 2, q = 2, r = 2$ 
(5)  $a = 6, b = 2, q = 2, r = 0$ 
(3)  $a = 6, b = 2, q = 3, r = 0$ 
 $a = 6, b = 2, q = 3, r = 0$ 
```

Motivational Example of Formal Specification

A potential implementation:

Integer division

 $\{a \geq 0, b > 0\}$

```
(1)  $q \leftarrow 0$ 
(2)  $r \leftarrow a$ 
(3) while  $r > b$  do
(4)    $r \leftarrow r - b$ 
(5)    $q \leftarrow q + 1$ 
(6) endwhile
```

 $\{a = qb + r\}$

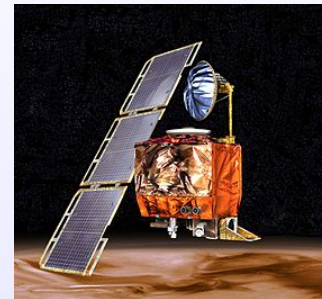
Program states before running each line

 $a = 6, b = 2$

```
(3)  $a = 6, b = 2, q = 0, r = 6$ 
(4)  $a = 6, b = 2, q = 0, r = 6$ 
(5)  $a = 6, b = 2, q = 0, r = 4$ 
(3)  $a = 6, b = 2, q = 1, r = 4$ 
(4)  $a = 6, b = 2, q = 1, r = 4$ 
(5)  $a = 6, b = 2, q = 1, r = 2$ 
(3)  $a = 6, b = 2, q = 2, r = 2$ 
 $a = 6, b = 2, q = 2, r = 2$ 
```

The algorithm is not correct!

There's money at stake ...



On 23 September 1999, the *Mars Climate Orbiter* space probe crashed on Mars, due to a mixup of metric units and imperial units.

The mission cost was USD 327 million.

... much money actually ...



On May 6, 2010 US stock markets began to fall rapidly, dropping more than 600 points in 5 minutes. That was the biggest one-day point decline in history.

This may have been caused by a positive feedback loop started by an automatic sell program and reinforced by high-frequency trading algorithms.

... and much more than just money!



On 26 October 1992, the *London Ambulance System* started to use LASCAD, a computer-aided ambulance delivery system.

The functioning of the system was chaotic, with waiting times of up to 11 hours. About 30 people died most likely due to these delays.

- ❶ The system had been poorly designed and implemented.
- ❷ It had not been tested at full load.
- ❸ Users were not familiar with the program, and the system was unfriendly.

... and much more than just money!



The [Therac-25](#) was a radiation therapy machine produced in 1982. It completely depended on software for operational safety controls.

At least 6 incidents of radiation overdose reported between 1985 and 1987: 3 patients died as a result.

- ❶ Failure caused by a counter overflowing (preventing safety checks) and some infrequent sequence of keystrokes provoking race conditions.
- ❷ Software was re-used from previous models that implemented hardware interlocks for safety. The software could not check the proper functioning of sensors.
- ❸ Overconfidence in the functioning of the system.

Pre/Post Specification

State

Let \mathcal{V} be the set of variables of an algorithm, and let \mathcal{T} be the set of values these variables can take. Thus, a [state](#) is a function

$$\sigma : \mathcal{V} \longrightarrow \mathcal{T}$$

mapping each variable to a value of its corresponding type.

Example

Given the integer division algorithm shown before, the expression $\{a = 6, b = 2, q = 3, r = 0\}$ represents the program state

$$\sigma : \{a, b, q, r\} \longrightarrow \mathbb{N}$$

defined as $\sigma(a) = 6, \sigma(b) = 2, \sigma(q) = 3, \sigma(r) = 0$

Pre/Post Specification

State

Let \mathcal{V} be the set of variables of an algorithm, and let \mathcal{T} be the set of values these variables can take. Thus, a **state** is a function

$$\sigma : \mathcal{V} \longrightarrow \mathcal{T}$$

mapping each variable to a value of its corresponding type.

Program Run

A run of a program can be described as a sequence of states.

Pre/Post Specification

Pre/Post Specification

A pre/post specification is a triple $\{Q\}S\{R\}$ where

- $\{Q\}$ is an assert termed **precondition**, characterizing valid initial states.
- S is an algorithm.
- $\{R\}$ is an assert termed **postcondition**, characterizing the final state (relationship between input and output values).

Pre/Post Specification

Assert and Assert Satisfaction

An assert is a logical expression defined over the variables of a program.

A state σ satisfies an assert A if A is true for the variable values implied by σ . We denote this as $\sigma \models A$

Example

The state $\{a = 6, b = 2, q = 3, r = 0\}$ satisfies the assert $\{a = qb + r, r < b\}$ since $6 = 3 \cdot 2 + 0$ and $0 < 2$.

Notice

A comma-separated list of logical expressions in an assert represents their logical conjunction (AND).

Pre/Post Specification

Example

$\{a \geq 0, b > 0\}$ Division $\{a = qb + r, r < b\}$

A pre/post specification $\{Q\}S\{R\}$ is read as:

"If S is run from a state for which $\{Q\}$ holds, then if it halts it will reach a state satisfying $\{R\}$."

Pre/post specifications are useful for:

- 1 Describing the conditions under which S has to run (and hence has to account for).
- 2 Describing the expected outcome of S (and hence what it can be used for).
- 3 Verifying that the algorithm fulfills its purpose as indicated by the postcondition.

Semantics

Predicate Evaluation

Let P be a predicate and let σ be a state. We denote as $[[P]]\sigma$ the result of evaluating P at state σ . This is either true or false.

Example

Let $x, y \in \mathbb{N}$, and let $\sigma \equiv \{x = 9, y = 1\}$. Then $[[x/y > 0]]\sigma = \text{true}$, $[[x > (y + 7)]]\sigma = \text{true}$ and $[[y < 0]]\sigma = \text{false}$.

Semantics

Logical consequence

Let P, Q be predicates. Then:

- 1 Q is a **logical consequence** of P (denoted as $P \models Q$) iff $\forall \sigma \in \Sigma : (\sigma \models P \Rightarrow \sigma \models Q)$.
- 2 \models is a partial order.
- 3 $\{P_1, P_2, \dots, P_n\} \models Q$ iff $\models (P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q)$.

Example

- $P \equiv (a = qb + r) \wedge (r < b)$.
- $Q \equiv (a = qb + r)$.
- $P \models Q$.

Semantics

Well Defined Predicate

Let P be a logical predicate and let σ be a state. If for all $v \in \mathcal{V}(P)$, $\sigma(v)$ is a valid value for v and the expressions in which v appear are defined for this value, then P is well defined. Otherwise it is undefined.

Example

Let $x, y \in \mathbb{N}$, let $\sigma_1 \equiv \{x = 9, y = 1\}$ and $\sigma_2 \equiv \{x = 9, y = 0\}$, and let $P \equiv x/y > 0$. Then P is well defined in σ_1 and undefined in σ_2 .

Specification with Predicates

States defined by a predicate

Given a predicate P , the set of states defined by P is

$$\text{states}(P) = \{\sigma \mid [[P]]\sigma\}$$

Predicate comparison

Given two predicates P and Q ,

- P is stronger than Q iff $P \models Q$, i.e., $\text{states}(P) \subseteq \text{states}(Q)$.
- P is equivalent to Q iff $\forall \sigma \in \Sigma : [[P]]\sigma = [[Q]]\sigma$, i.e., $\text{states}(P) = \text{states}(Q)$.
- P is weaker than Q iff $Q \models P$, i.e., $\text{states}(Q) \subseteq \text{states}(P)$.

Specification with Predicates

Example

Predicate $P \equiv x > 0$ is stronger than predicate $Q \equiv x \geq 0$.

Problem Specification

Example

```
{N > 0}
func Max (↓a: ARRAY [1..N] OF ℕ):ℕ
{∃j ∈ {1..N} : ((Max(a) = a[j]) ∧ (∀k ∈ {1, ..., N} : a[k] ≤ a[j]))}
```

Example

```
{true}
func Sum (↓a: ARRAY [1..N] OF ℕ):ℕ
{Sum(a) = ∑i=1n a[i]}
```

Specification with Predicates

Sort from weakest to strongest

- 1 $x > 0$
- 2 $(x > 0) \wedge (y > 0)$
- 3 $(x > 0) \vee (y > 0)$
- 4 $(y \geq 0)$
- 5 $(x \geq 0) \wedge (y \geq 0)$

Verification

Axiomatic Semantics

The **axiomatic semantics** of a language is given by the collection of axioms (rules) defining the behavior of each instruction in the language.

In addition to language-specific axioms, there are also some rules of general application.



Verification

Having the pre/post specification of an algorithm, use the rules implied by the axiomatic semantics of the language to prove that the postcondition is a logical consequence of the application of the algorithm given its precondition.

Advices

- 1 Before attempting the implementation of an algorithm, try to specify it as formally as possible.
- 2 One can be informal when dealing with obvious things, but rigor is essential when tackling complex issues.
- 3 Documenting the program is a huge help, e.g., pre/post conditions of each method/function.

Specific Bibliography

-  R. Peña,
Diseño de programas: Formalismo y Abstracción,
Prentice-Hall, 1997 (2nd edition).
-  D. Gries,
The Science of Programming,
Springer-Verlag, 1981

Conclusions

Formal verification is a crucial tool for **saving**
program-malfunctioning costs.

But its use is sometimes out of question due to the **sheer complexity**
of the problem, and the enormous amount of time required...

...and we have just scratched the surface!

Image Credits

- Mars Climate Orbiter: NASA/JPL/Corby Waste, public domain.
- Stockbrokers: Richard Drew, AP/Scanpix
- Therac 25: uncredited, taken from [https://hackaday.com.files.wordpress.com/2015/10/therac-machine.jpg](https://hackaday.com/files.wordpress.com/2015/10/therac-machine.jpg)
- London Ambulance: Oxyman, CC BY 2.5