Verificación de funciones

Desarrollo Formal de Software

Jorge Arturo Mina Riascos
Mike Giovanny Pineda Moreno



¿Por qué verificar funciones?

Verificar una función es demostrar que cumple su contrato

Probar

Ejecuta casos concretos. Muestra que la función funciona en los casos probados.

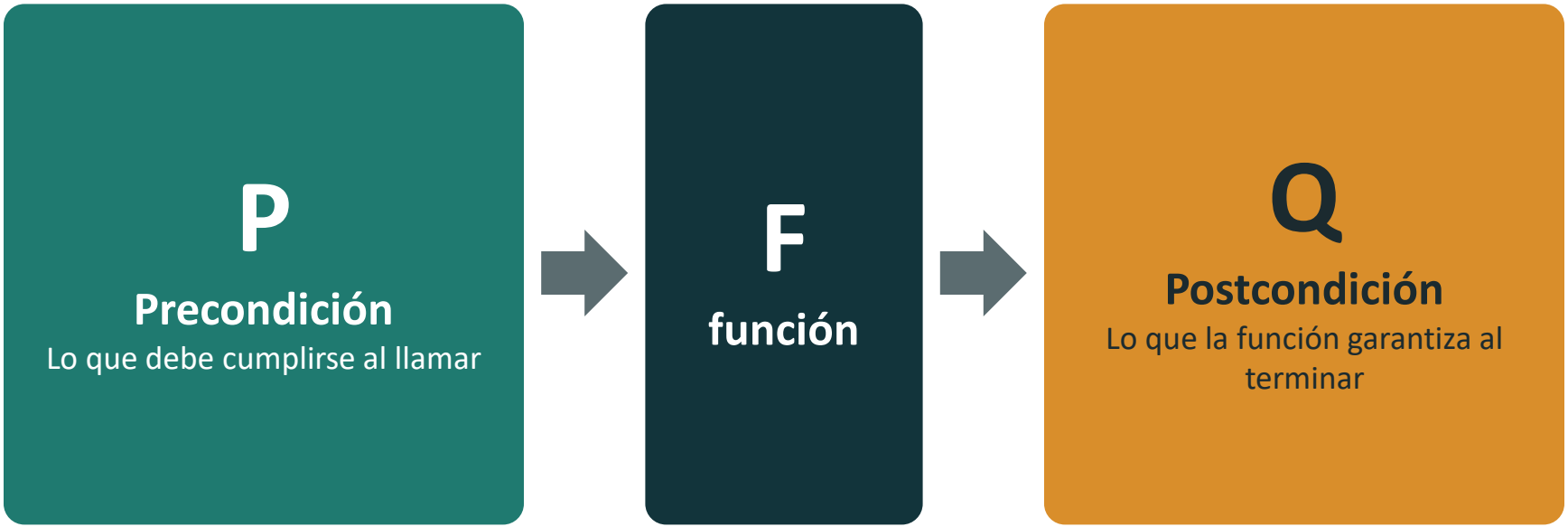
Verificar

Razona sobre todos los casos. Demuestra que funciona para toda entrada que cumpla la precondición.

Dónde importa

Sistemas críticos como aviación, medicina y finanzas, donde un error cuesta caro.

Contrato de una función



Tripleta de Hoare

$$\{P\} \quad F \quad \{Q\}$$

Se lee: si P es verdadera antes de ejecutar F, entonces Q es verdadera al terminar.

Ejemplo $\{x \geq 0\} \quad y := x + 1 \quad \{y \geq 1\}$

Verificación modular

Quien implementa F

- Demuestra una sola vez que {P} cuerpo {Q}.
- Puede asumir P.
- Debe garantizar Q.

Quien llama a F

- Demuestra que P se cumple antes de la llamada.
- Después usa solo Q.
- No necesita leer el cuerpo.

Regla de la consecuencia

Fortalecer la precondición

Si {P} F {Q} y P' ⇒ P, entonces {P'} F {Q}

Ejemplo con {x ≥ 0} y := x + 1 {y ≥ 1}

{x ≥ 5} y := x + 1 {y ≥ 1}
porque x ≥ 5 ⇒ x ≥ 0

Debilitar la postcondición

Si {P} F {Q} y Q ⇒ Q', entonces {P} F {Q'}

Ejemplo con {x ≥ 0} y := x + 1 {y ≥ 1}

{x ≥ 0} y := x + 1 {y ≥ 0}
porque y ≥ 1 ⇒ y ≥ 0

Ejemplo: máximo de tres números

```
función maxTres(a, b, c) → r
{P: verdadero}
t := max(a, b)
r := max(t, c)
{Q: r ≥ a ∧ r ≥ b ∧ r ≥ c ∧
    (r = a ∨ r = b ∨ r = c)}
```

Contrato dado de max(a, b)

```
{P: verdadero}
{Q: r ≥ a ∧ r ≥ b ∧
    (r = a ∨ r = b)}
```

Verificación

1. Tras max(a, b): t ≥ a, t ≥ b y t es a o b.
2. Tras max(t, c): r ≥ t, r ≥ c y r es t o c.
3. Entonces r ≥ a, r ≥ b, r ≥ c y r es a, b o c. Se cumple Q.

Aplicación: evaluar solicitud

```
función evaluarSolicitud(solicitud)
{P: solicitud.estado = pendienteEvaluacion ∧
    solicitud.dias ≥ 0}
v := esValida(solicitud.dias,
              solicitud.excusa)
si v entonces
    solicitud.estado := pendientePago
sino
    solicitud.estado := rechazada
    solicitud.nota := 0
{Q: (v ⇒ solicitud.estado = pendientePago) ∧
    (¬v ⇒ solicitud.estado = rechazada ∧
          solicitud.nota = 0)}
```

Contrato dado de esValida

```
{P: dias ≥ 0}
{Q: r ⇔ (dias ≤ 5 ∧
    excusa)}
```

Verificación

1. De P se obtiene dias ≥ 0: la llamada es válida.
2. Se usa solo el contrato: v ⇔ (dias ≤ 5 ∧ excusa).
3. Cada rama deja el estado que exige Q.

Errores frecuentes y límites

Errores frecuentes

- Postcondición débil: omitir r = a ∨ r = b deja pasar resultados incorrectos.
- Llamar a la función sin cumplir su precondición.
- Olvidar las variables que la función modifica.

Limitaciones

- La especificación también puede estar mal: se verifica contra el contrato, no contra la intención.
- Exige tiempo y práctica.
- Complementa a las pruebas, no las reemplaza.

Ideas clave

1. Verificar una función es demostrar que cumple su contrato {P} F {Q}.
2. La verificación es modular: quien llama usa solo el contrato.
3. Un buen contrato exige una postcondición completa, no solo plausible.
4. Escribir contratos en el proyecto reduce errores desde el diseño.

¡GRACIAS POR SU ATENCION!

Referencias

- Software Foundations, Vol. 2: Programming Language Foundations. Hoare Logic, Part I. University of Pennsylvania. https://softwarefoundations.cis.upenn.edu/plf-current/Hoare.html
- Sabirov, R. (23 de diciembre de 2024). Verification vs validation. Qase. https://www.qase.io/blog/verification-vs-validation/
- Material de clase de Desarrollo Formal de Software.