

# Introducción a la Verificación Formal

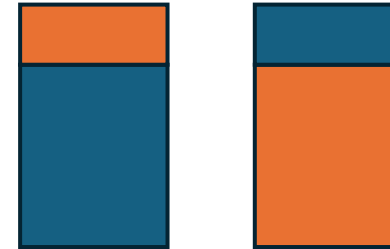
Clase 7 – 9/10/2025

# Acertijo

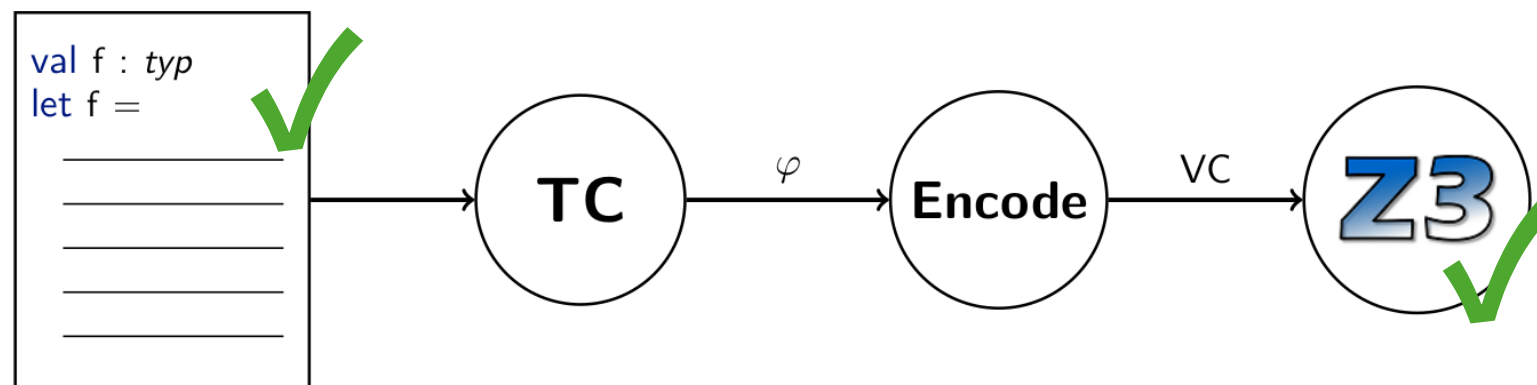
## Acertijo:

1. Tengo dos tazas, una con 100ml de café y una con 100ml leche. Tomo una cucharada (X ml) de café de la primera taza y la pongo en la segunda taza, mezclando todo perfectamente. Luego tomo una cucharada de la segunda taza y la paso a la primera taza. ¿Hay más leche en el café o más café en la leche?
2. Mismo problema pero, antes de empezar, reservo 50ml de café en otra taza, y al final lo devuelvo.
3. Mismo problema pero ahora tengo dos cucharas de distinto volumen. Primero muevo C->L con cuchara 1, luego L->C con cuchara 2, luego L->C con cuchara 1, luego C->L con cuchara 2.

1. Cantidad de café constante
2. Cantidad de leche constante
3. Al terminar, ambos vasos tienen el mismo volumen de líquido



# Verificación “auto-activa”




# Lógica de Hoare - Reglas

$e ::= \text{var} \mid c \mid e + e \mid e - e \mid e * e \mid e = e \mid \dots$

$s ::= \text{skip} \mid x := e \mid s; s \mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ s}$

La propiedad P debe valer en el estado actual, modificado para que x tenga el valor de la expression e.



H-SKIP

$$\frac{}{\{P\} \text{ skip } \{P\}}$$

H-ASSIGN

$$\frac{}{\{\lambda s. P(s \oplus (x, \llbracket e \rrbracket))\} x := e \{P\}}$$

H-SEQ

$$\frac{\{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$$

$(f \oplus (x, y)) = \text{fun } z \rightarrow \text{if } z = x \text{ then } y \text{ else } f z$

# Las reglas tienen “juego” (*slack*)

$$\frac{\{x = 1 \wedge y = 1\} \ x := 2 \ \{Q\} \quad \{Q\} \ y := 3 \ \{x > 0 \wedge y > 0\}}{\{x = 1 \wedge y = 1\} \ x := 2; y := 3 \ \{x > 0 \wedge y > 0\}}$$

Q puede ser:

- $x = 2 \ /\ \ y = 1$
- $x > 0 \ /\ \ y = 1$
- $x = 2 \ /\ \ y > 0$
- $x > 0 \ /\ \ y > 0$
- $x = 2 \ /\ \ y = 1 \ /\ \ \dots$

- ¿Cómo escribimos un verificador que no tenga que adivinar?

# ¿Hay alguna forma *canónica*?

- Es la pregunta que se hace [Dijkstra \(1975\)](#). Notando que siempre Podemos fortalecer una precondición, i.e.:
  - Si  $\{P\}C\{Q\}$  y  $(\text{forall } x. P' \ x \implies P \ x)$ , entonces  $\{P'\}C\{Q\}$   
¿podemos encontrar, dado C y Q, la precondición P *más débil* (*weakest precondition*)?
- Resulta que sí, y que son fácilmente computables.
  - $wp(S, Q)$  computa la precondición más débil para que S cumpla Q (reglas más adelante)
  - Correctitud:  $\{wp(S, Q)\} S \{Q\}$
  - Optimalidad:  $\{P\} S \{Q\} \iff P \implies wp(S, Q)$
  - A diferencia de las reglas de Hoare, no hay juego en la composición secuencial

# Computando WPs

$$wp(skip, Q) = Q$$

$$wp((S_1; S_2), Q) = wp(S_1, wp(S_2, Q))$$

$$wp(x := E, Q) = \lambda s. Q(s \oplus (x, E))$$

$$\begin{aligned} wp(\mathbf{if} \ c \ \mathbf{then} \ t \ \mathbf{else} \ e, Q) &= (c \wedge wp(t, Q)) \vee (\neg c \wedge wp(e, Q)) \\ &= (c \implies wp(t, Q)) \wedge (\neg c \implies wp(e, Q)) \end{aligned}$$

# Bucles

- Si bien buscamos un proceso automático, no podemos adivinar invariantes
- Vamos a anotar los invariantes en el código
  - Cambiamos la sintaxis para agregar un  $inv : cond$  al bucle `while`
  - Este campo se ignora por completo para la semántica y la lógica de Hoare
  - El cómputo de WPs lo toma, aunque no “sabe” si está bien

$$\begin{aligned} wp(\mathbf{while}_{inv}(C) \{E\}, Q) &= inv \\ &\wedge (\forall s. C \wedge inv \ s \implies wp(E, inv)) \\ &\wedge (\forall s'. \neg C \wedge inv \ s' \implies Q) \end{aligned} \left. \vphantom{\begin{aligned} wp(\mathbf{while}_{inv}(C) \{E\}, Q) &= inv \\ &\wedge (\forall s. C \wedge inv \ s \implies wp(E, inv)) \\ &\wedge (\forall s'. \neg C \wedge inv \ s' \implies Q) \end{aligned}} \right\} \begin{array}{l} \text{Proposiciones} \\ \text{puras} \end{array}$$

- (nota: Dijkstra hace otra cosa porque le interesa la correctitud total, pero ni siquiera está claro que está bien definida su WP para bucles.)



# Más propiedades de las WP

- Monotonía:  $(\forall s. Q \ s \implies Q' \ s) \implies wp(S, Q) \implies wp(S, Q')$

- Conjuntividad:  $wp(S, Q_1 \ \wedge \ Q_2) \iff wp(S, Q_1) \ \wedge \ wp(S, Q_2)$

- “Ley del Milagro excluído” (sólo para correctitud total):

$$wp(S, \mathbf{false}) \iff \mathbf{false}$$

- Disjuntividad (sólo para programas deterministas, si no, sólo vale  $\leq$ )

$$wp(S, Q_1 \ \vee \ Q_2) \iff wp(S, Q_1) \ \vee \ wp(S, Q_2)$$

# WPs en $F^*$

- Estuvimos usando WPs todo el tiempo.
- El efecto `Pure a pre post` está definido sobre `PURE a wp`. Toda especificación de computaciones es con WPs
  - Estas WPs son funcionales (las funciones devuelven un valor) a diferencia de las puramente imperativas que estuvimos formalizando. No es una diferencia mayor.
- `Pure a pre post = PURE a (fun p -> pre /\ (forall x. post x ==> p x))`
- Cada efecto tiene su propio *cálculo* de WPs, que indica cómo computar WPs para programas en ese efecto. Cada cálculo forma una mónada.
- Opcional: ver paper [Dijkstra Monads for Free](#)
  - Desde una mónada à-la-Haskell, deriva un cálculo de WPs correcto automáticamente
  - $F^*$  se extiende con un nuevo efecto (simulado)

# Cerrando

- Las WPs proveen un forma más composicional y automatizable de verificar que la lógica de Hoare básica
  - Sin embargo son interconvertibles, y la correctitud de las WPs puede demostrarse para una lógica dada
- Varias herramientas reales usan WPs para computar VCs
  - F\*, Dafny, Boogie (y por lo tanto VCC y varios otros), Why3
  - En F\* son muy visibles. En otras herramientas no tanto.

# Tareas

Completar Clase07.WP.fst

- Completar cómputo de WPs y demostrar correctitud
  - Hay que agregar reglas de weakening y proposiciones puras a la lógica de Hoare. Es fácil demostrarlas correctas.
- No demostrar optimalidad
- Demostrar que los programas de ejemplo son correctos, encontrando sus invariantes y precondition más débil.
- Demostrar monotonía
- **Clases invitadas: piensen si les interesa algo en particular**