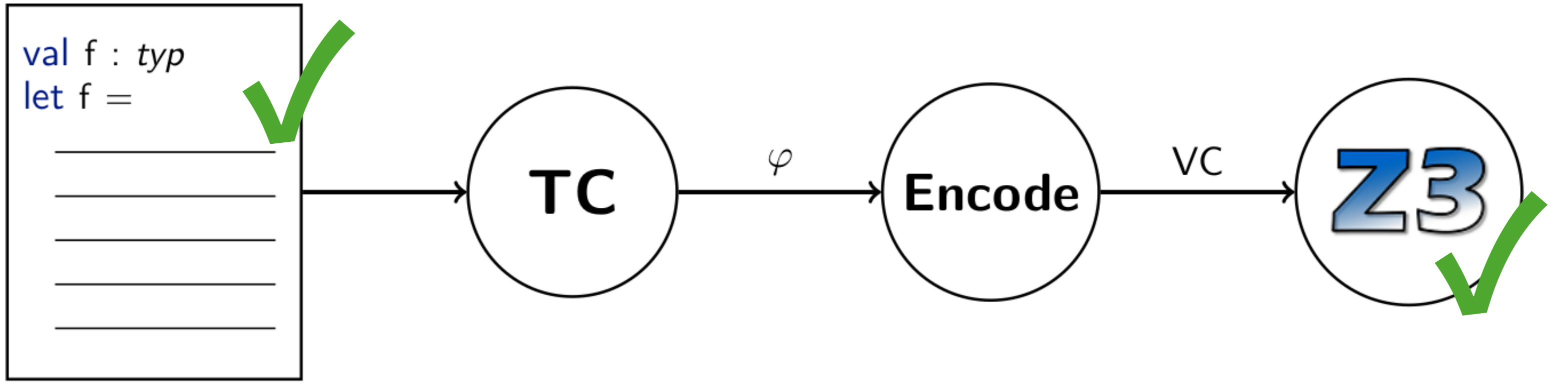
Verificación de Programas con F*

Clase 8 – 14/05/2024

Verificación "auto-activa"



Lógica de Hoare - Reglas

e ::= var | c | e + e | e – e | e * e | e = e |....

s ::= **skip** | x := e | s;s | **if** e **then** s **else** s | **while** e s

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

H-SKIP

$$\frac{}{\{P\}\ \mathbf{skip}\ \{P\}}$$

H-ASSIGN

$$\frac{}{\{\lambda s. P(s \oplus (x, [\![e]\!]))\}\ x := e\ \{P\}}$$

H-SEQ

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

(f ⊕ (x,y)) = fun z -> if z = x then y else f z

Las reglas tienen “juego” (*slack*)

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

Q puede ser:

- x = 2 /\ y = 1
- x > 0 /\ y = 1
- x = 2 /\ y > 0
- x > 0 /\ y > 0
- x = 2 /\ y =1 /\ ....

  - ¿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 (forall x. P' x ==> 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

$$
\begin{array}{rcl}
wp(skip, Q) & = & Q \\
wp((S_1; S_2), Q) & = & wp(S_1, wp(S_2, Q)) \\
wp(x := E, Q) & = & \lambda s.Q(s \oplus (x, E)) \\
wp(\textbf{if } c \textbf{ then } t \textbf{ else } e, Q) & = & (c \wedge wp(t, Q)) \vee (\neg c \wedge wp(e, Q)) \\
& = & (c \implies wp(t, Q)) \wedge (\neg c \implies wp(e, Q))
\end{array}
$$

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

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

- (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 <==)

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

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, (VCC?)
  - En F* son muy visibles. En otras herramientas no tanto.

Tareas

Completar Clase8.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 precondición más débil.
- Demostrar monotonía

- **Próxima clase: Gabriel Ebner sobre Lean4 y Mathlib**
    - Sería bueno si ven algo de Lean antes de la clase, aunque no hace falta.