Introducción a la Verificación Formal

Clase 10 – 27/11/2025

Concurrencia en lógica de separación

- Dada la localidad de las especificaciones, se tiene una regla muy directa para la composición paralela de programas

$$\textsc{Par}\ \frac{\{pre_1\}\ C_1\ \{post_1\} \qquad \{pre_2\}\ C_2\ \{post_2\}}{\{pre_1 * pre_2\}\ C_1||C_2\ \{post_1 * post_2\}}$$

- La separación implica que C2 no puede interferir con pre1/post1, y viceversa. En general, ningún otro hilo puede invalidar el *conocimiento* de otro.
  - Esto lleva a que verificar programas masivamente paralelos sea razonable

```
(* Suma paralela *)
fn incr_par (x y : ref int)
  requires x |-> 'vx ** y |-> 'vy
  ensures  x |-> ('vx + 1) ** y |-> ('vy + 1)
{
  Pulse.Lib.Par.par
    fn () { x := !x + 1; }
    fn () { y := !y + 1; }
}
```

Concurrencia en lógica de separación

- ¿Cómo compartir recursos sin invalidar la regla previa?
- 1. Necesitamos poder compatir recursos de solo lectura
  - Permisos fraccionales
- 2. Recursos mutables, protegidos de alguna forma
  - Invariantes concurrentes

Permisos fraccionales

- Un thread puede tener cualquier *fracción* de permiso sobre una locación de memoria
  - Permiso completo: 1, cualquier fracción menor (positiva) es posible.
  - Alocar retorna permiso completo. Liberar requiere permiso completo también.
- El permiso sobre una referencia puede fraccionarse o acumularse.
  - Se fracciona arbitrariamente
  - Al acumular, obtenemos la igualdad entre ambos valores
- Si tenemos x |->_f v en el contexto, ningún otro hilo puede invalidarlo, ni podemos invalidar el conocimiento de otro hilo.
  - Si f = 1, sólo este hilo tiene conocimiento sobre x. Modificarlo está bien.
  - Si f < 1, no podemos modificar el valor, pero a la vez ningún otro hilo puede hacerlo.
- Bajo el capot: usualmente esto se construye sobre una teoría de *monoides parciales conmutativos (PCM)*.
  - Los PCM capturan las nociones de separación e independencia.
  - Las referencias son solo un caso en particular.

ALLOC

$$\frac{}{\{\}\ x := \mathbf{alloc}()\ \{\exists v. x \mapsto_1 v\}}$$

SHARE

$$\frac{f_1, f_2 > 0, f_1 + f_2 = f}{\{x \mapsto_f v\}\ \mathbf{share}\ x\ \{x \mapsto_{f_1} v \star x \mapsto_{f_2} v\}}$$

$$\frac{}{\{x \mapsto_{f_1} v_1 \star x \mapsto_{f_2} v_2\}\ \mathbf{gather}\ x\ \{x \mapsto_{f_1+f_2} v_1 \wedge f_1 + f_2 \leq 1 \wedge v_1 = v_2\}}$$

Invariantes en Lógica de Separación

- ¿Cómo compartir recursos más interesantes?
- Invariantes de recursos
  - Representan ciertos recursos que están presentes globalmente, y que todos los hilos pueden acceder (mientras sea de forma segura).
  - E.g. variable global, o mutex.
- Sólo pueden *abrirse* durante la ejecución de una instrucción *atómica*
  - Esto es crucial para no invalidar el conocimiento de otros hilos
- Los invariantes casi siempre implican que la lógica es *afín* (i.e. se pueden perder recursos).
- (En la práctica es algo más engorroso.)

$$\textsc{IAlloc}\quad \frac{}{\{P\}\ \mathbf{ialloc}\ P\ \{\lambda i.\ \mathbf{inv}\ i\ P\}}$$

$$\textsc{WithInv}\quad \frac{\{P \star I\}\ C\ \{Q \star I\} \qquad C \text{ es atómico}}{\{P \star \mathbf{inv}\ i\ P\}\ \mathbf{with}\ i\ C\ \{Q \star \mathbf{inv}\ i\ P\}}$$

Ejemplo: spinlocks

(demo)

Tareas

- En breve (sí) subo el archivo + submódulo de Pulse
- Voy a intentar que la semana que viene tengamos clase invitada
- Ya prácticamente terminamos