Concurrency Verification

How to prove the correctness of concurrent programs?

Difficult because of the interleaving and state sharing.

Sometimes the programs could be too smart such that they are very difficult to understand.

Recall the algorithms you saw yesterday.

Owicki-Gries Method

Susan Owicki and David Gries, 1975

$$\frac{\{p \wedge b\}c\{q\}}{\{p\}\textbf{await}\ b\ \textbf{then}\ c\{q\}}$$

$$\frac{\{p_i\}c_i\{q_i\}\ \text{ for all } 1 \leq i \leq n \qquad \text{Non-Interference condition holds}}{\{p_1 \wedge \cdots \wedge p_n\}c_1 \parallel c_2 \parallel \cdots \parallel c_n\{q_1 \wedge \cdots \wedge q_n\}}$$

Non-Interference

Key idea: execution of a statement does not invalidate proofs of other code fragments that may run in parallel with the statement in question.

Given a proof {p} c {q}, and a command T whose precondition is pre(T), we say **T does not interfere with {p} c {q}** if

- $\{q \wedge \text{pre}(T)\}\ T\ \{q\}$; and
- for all c’ in c (but not in await),

  $\{\text{pre}(c') \wedge \text{pre}(T)\}\ T\ \{\text{pre}(c')\}$

Non-Interference

$\{p_1\}\ c_1\ \{q_1\}, \dots \{p_n\}\ c_n\ \{q_n\}$ are **interference-free** if, for any **await** or **primitive statement** T (not inside await) in c_i, and for all $j \neq i$, T does not interfere with $\{p_j\}\ c_j\ \{q_j\}$.

This method is not compositional!

Example 1

$$\{ x = 0 \}$$

$$\langle x := x + 1 \rangle \quad \| \quad \langle x := x + 2 \rangle$$

$$\{ x = 3 \}$$

Example 1

$$\{ x = 0 \}$$
$$\{ (x = 0 \lor x = 2) \land (x = 0 \lor x = 1) \}$$

$$\{ x = 0 \lor x = 2 \} \qquad \{ x = 0 \lor x = 1 \}$$
$$< x := x + 1 > \quad || \quad < x := x + 2 >$$
$$\{ x = 1 \lor x = 3 \} \qquad \{ x = 2 \lor x = 3 \}$$

$$\{ (x = 1 \lor x = 3) \land (x = 2 \lor x = 3) \}$$
$$\{ x = 3 \}$$

Example 1(b)

$\{ x = 0 \}$

$\langle x := x + 1 \rangle \quad || \quad \langle x := x + 1 \rangle$

$\{ x = 2 \}$

Example 1(b)

{ x = 0 }

y := 0 ; z := 0 ;

< x := x + 1 ; y := y + 1 >	||	< x := x + 1 ; z := z + 1 >

{ x = 2 }

Example 1(b)

$\{ x = 0 \}$

$y := 0 ; z := 0 ;$

$\{ (x = y + z \wedge y = 0) \wedge (x = y + z \wedge z = 0) \}$

$\{x = y + z \wedge y = 0\}$		$\{ x = y + z \wedge z = 0\}$
$< x := x + 1 ;$	$\|\|$	$< x := x + 1 ;$
$y := y + 1 >$		$z := z + 1 >$
$\{x = y + z \wedge y = 1\}$		$\{ x = y + z \wedge z = 1\}$

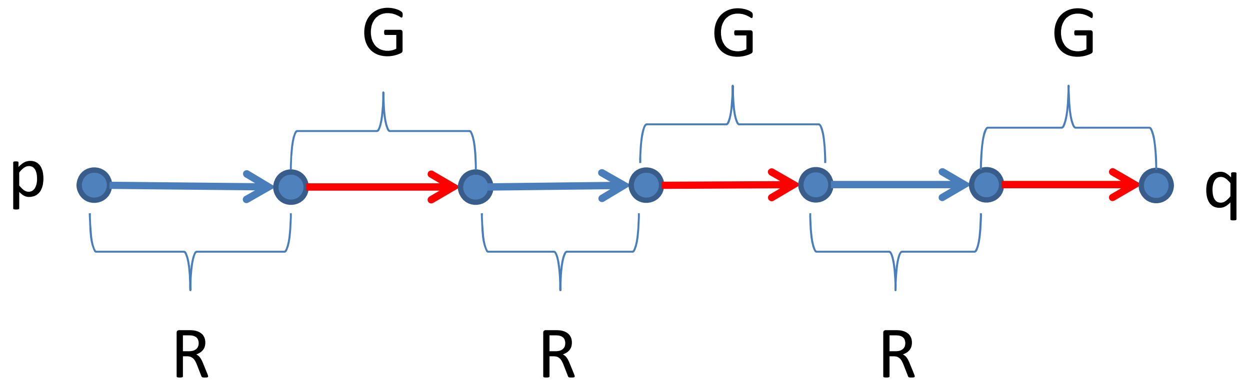
$\{ (x = y + z \wedge y = 1) \wedge (x = y + z \wedge z = 1) \}$

$\{ x = 2 \}$

Rely-Guarantee Reasoning

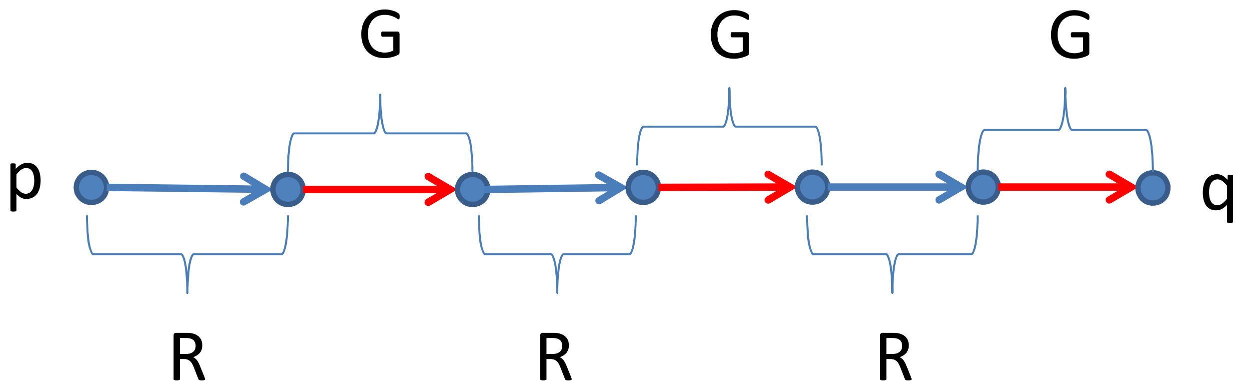
- Use rely (R) and guarantee (G) conditions to summarize the behaviors of environments and the thread itself.

$$R, G \vdash \{p\}c\{q\}$$



Rely-Guarantee Reasoning

- R and G: specification of state transitions
  example: $x' \geq x$



Inference Rules

$$\frac{\vdash \{p\}c\{q\} \quad \mathbf{stable}(p, R) \quad \mathbf{stable}(q, R) \quad (p; q) \Rightarrow G}{R, G \vdash \{p\}c\{q\}}$$

where $(\sigma, \sigma') \models (p; q) \stackrel{\text{def}}{=} \sigma \models p \wedge \sigma' \models q$

$$\mathbf{stable}(p, R) \stackrel{\text{def}}{=} \forall \sigma, \sigma'. (\sigma \models p) \wedge ((\sigma, \sigma') \models R) \Rightarrow \sigma' \models p$$

Inference Rules (2)

$$\frac{\begin{array}{c} R_1, G_1 \vdash \{p_1\} c_1 \{q_1\} \quad R_2, G_2 \vdash \{p_2\} c_2 \{q_2\} \\ R \Rightarrow R_1 \wedge R_2 \quad G_1 \vee G_2 \Rightarrow G \quad G_1 \Rightarrow R_2 \quad G_2 \Rightarrow R_1 \end{array}}{R, G \vdash \{p_1 \wedge p_2\}\, c_1 \parallel c_2\, \{q_1 \wedge q_2\}}$$

Example

$$\{x = 0\}$$

$$\langle\ x := x+1\ \rangle \quad || \quad \langle\ x := x+1\ \rangle$$

$$\{x = 2\}$$

$G1 \equiv y = 0 \wedge y' = 1 \wedge$
$x' = x+1 \wedge z' = z$

$G2 \equiv z = 0 \wedge z' = 1 \wedge$
$x' = x+1 \wedge y' = y$

$R1 \equiv G2$

$R2 \equiv G1$

$\{x = 0\}$

y := 0; z := 0;

$\{x = y+z \wedge y = 0 \wedge z = 0\}$

$\{x = y+z \wedge y = 0\}$		$\{x = y+z \wedge z = 0\}$
< x := x+1;	||	< x := x+1;
y := 1; >		z := 1; >
$\{x = y+z \wedge y = 1\}$		$\{x = y+z \wedge z = 1\}$

$\{x = y+z \wedge y = 1 \wedge z = 1\}$

$\{x = 2\}$

Example (2)

$$\{x \to M, N\}$$

```
⟨t11 := [x]⟩;
⟨t12 := [x+1]⟩;
while (t11 ≠ t12) do{
   if(t11 > t12) then {
      t11 := t11 − t12;
      ⟨[x] := t11⟩;
   }
   ⟨t12 := [x+1]⟩;
}
```

$\|$

```
⟨t21 := [x+1]⟩;
⟨t22 := [x]⟩;
while (t21 ≠ t22) do{
   if(t21 > t22) then {
      t21 := t21 − t22;
      ⟨[x+1] := t21⟩;
   }
   ⟨t22 := [x]⟩;
}
```

$$\{\exists O.\ x \to O, O \land O = GCD(M, N)\}$$

$$G1 \equiv [x+1] > [x'+1] \wedge GCD([x'], [x'+1]) = GCD(M, N)$$
$$\wedge \; ([x] > [x+1] \Rightarrow [x] > [x']$$
$$\wedge \; [x] \leq [x+1] \Rightarrow [x] = [x'])$$

$$G2 \equiv [x] > [x'] \wedge GCD([x'], [x'+1]) = GCD(M, N)$$
$$\wedge \; ([x+1] > [x] \Rightarrow [x+1] > [x'+1]$$
$$\wedge \; [x+1] \leq [x] \Rightarrow [x+1] = [x'+1])$$

$R1 \equiv G2$ $\qquad$ $R2 \equiv G1$

```
{GCD([x], [x+1]) = GCD(M, N)}
t11 := [x];
{GCD([x], [x+1]) = GCD(M, N) ∧ t11 = [x] }
t12 := [x+1];
{GCD(t11, t12) = GCD(M, N) ∧ t11 = [x]
            ∧ t12 ≥ [x+1] ∧ ([x] ≥ [x+1] ⇒ t12 = [x+1]) }
while(t11 ≠ t12) {
    if (t11 > t12){
            {GCD(t11, t12) = GCD(M, N) ∧ t11 = [x] ∧ t12 = [x+1] }
            t11 := t11 – t12;
            [x] := t11;
    }
    t12 := [x+1];
}
```

Example (3)

- A lock protecting variable x

```
Lock(L):

    tmp := 1;

    while(tmp ≠ 0)
        < tmp := [L];
          if (tmp = 0) then [L]:= tid>
```

```
Unlock(L):

    [L] := 0;
```

$$G(tid) \equiv (([L] = 0 \land [L'] = tid) \lor ([L] = tid \land [L] = 0))$$
$$\land ([L] \neq 0 \land [L] \neq tid \Rightarrow [L] = [L'] \land [x] = [x'])$$

$$R(tid) \equiv [L] = tid \Rightarrow ([L] = [L'] \land [x] = [x'])$$

{ true }

while(tmp $\neq$ 0)
< tmp := [L];
if (tmp = 0) then [L]:= tid>

{ [L] = tid }

{ [L] = tid }

[L] := 0;

{ true }

Soundness

$$R, G \vdash \{p\}c\{q\}$$

- Partial correctness
- Safety
- Preservation of R/G