Is the following true?

\{x = 0\}
y := x;
x := x + 1;
\{x = 1 \land y = 0\}

YES!
Is the following still true?

\[
\{x = 0\} \\
y := x; \\
x := x + 1; \\
{x = 1 \land y = 0}\]

\[
\parallel \\
x := 5; \\
x := 5; \\
\parallel \\
\]

NO!

The parallel process may interfere with the intermediate assertions.
Is the following still true?

\[
\{x = 0\} \\
y := x; \\
x := x + 1; \\
\{x = 1 \land y = 0\}
\]

• NO!

\[
x := 5;
\]
Is the following still true?

\[
\begin{align*}
\{x = 0\} & \\
y := x; \quad \parallel \quad x := 5; \\
\{x + 1 = 1 \land y = 0\} & \\
x := x + 1; \\
\{x = 1 \land y = 0\} & 
\end{align*}
\]

NO!
Is the following still true?

\[
\{x = 0\}
\]
\[
y := x; \quad | \quad x := 5;
\]
\[
\{x + 1 = 1 \land y = 0\}
\]
\[
x := x + 1;
\]
\[
\{x = 1 \land y = 0\}
\]

NO!

The parallel process may interfere with the intermediate assertions.
Parallel Composition

• Extend IMP language of previous lectures with parallel composition

\[
e ::= \ n \ | \ x \ | \ e_1 + e_2 \ | \ e_1 = e_2
\]

\[
c ::= \ x := e \ | \ \text{if } e \ \text{then } c_1 \ \text{else } c_2 \ | \\
\quad \text{while } e \ \text{do } c \ | \ \text{skip} \ | \ c_1 ; c_2 \ | \\
\quad c_1 \parallel c_2
\]
Can we derive a Hoare triple for parallel composition from the triples of each command?

First Attempt:

\[ \vdash \{ P_1 \} \ c_1 \ \{ Q_1 \} \quad \vdash \{ P_2 \} \ c_2 \ \{ Q_2 \} \]
\[ \vdash \{ P_1 \land P_2 \} \ \parallel \ \{ Q_1 \land Q_2 \} \]

**Intuition:** if we satisfy the preconditions of \( c_1 \) and \( c_2 \), their postconditions will be satisfied too.
Unsoundness of First Attempt

\[ \vdash \{ P_1 \} c_1 \{ Q_1 \} \quad \vdash \{ P_2 \} c_2 \{ Q_2 \} \]
\[ \vdash \{ P_1 \land P_2 \} c_1 \parallel c_2 \{ Q_1 \land Q_2 \} \]

- This rule is not always sound, consider:

\( \{ x = 1 \} \ y := 0 \ \{ x = 1 \} \quad \{ \text{true} \} \ x := 10 \ \{ \text{true} \} \)

- It does not hold that

\( \{ x = 1 \land \text{true} \} \ y := 0 \parallel x := 10 \ \{ x = 1 \land \text{true} \} \)
Second Attempt

\[
\begin{align*}
\vdash \{P_1\} c_1 \{Q_1\} & \quad \vdash \{P_2\} c_2 \{Q_2\} \\
\vdash \{P_1 \land P_2\} c_1 \parallel c_2 \{Q_1 \land Q_2\}
\end{align*}
\]

- If \(c_1\) and \(c_2\) do not read and write the same variables, and all the pres- and post- conditions talk about different variables
- What’s wrong with this?
• If \( c_1 \) and \( c_2 \) do not read and write the same variables, and all the pres- and post- conditions talk about different variables

• What’s wrong with this?

• No way to prove some program

• The rule is \textbf{incomplete}
Let $\text{UPD}(c)$ be the set of variables that are updated (modified) in $c$

$$\vdash \{P_1\} \ c_1 \ \{Q_1\} \quad \vdash \{P_2\} \ c_2 \ \{Q_2\}$$

$$\vdash \{P_1 \land P_2\} \ c_1 \ || \ c_2 \ \{Q_1 \land Q_2\}$$

If $\text{UPD}(c_1) \cap (\text{FV}(P_2) \cup \text{FV}(Q_2)) = \emptyset$ and $\text{UPD}(c_2) \cap (\text{FV}(P_1) \cup \text{FV}(Q_1)) = \emptyset$
Let $\text{UPD}(c)$ be the set of variables that are updated (modified) in $c$

$\vdash \{P_1\} c_1 \{Q_1\} \quad \vdash \{P_2\} c_2 \{Q_2\}$

$\vdash \{P_1 \land P_2\} c_1 \parallel c_2 \{Q_1 \land Q_2\}$

If $\text{UPD}(c_1) \cap (\text{FV}(P_2) \cup \text{FV}(Q_2)) = \emptyset$ and $\text{UPD}(c_2) \cap (\text{FV}(P_1) \cup \text{FV}(Q_1)) = \emptyset$

Still unsound. Consider:

$\{x = 0\} \ y := x; z := y \ \{z = 0\}$

$\{\text{true}\} \ y := 10 \ \{\text{true}\}$

It does not hold that

$\{x = 0 \land \text{true}\} \ y := x; z := y \parallel y := 10 \ \{z = 0 \land \text{true}\}$

**Diagnose:** $y := 10$ interferes with the proof of

$\{x = 0\} \ y := x; z := y \ \{z = 0\}$

$\uparrow$

$y = 0$
Susan Owicki,
“Axiomatic proof techniques for parallel programs”,
Cornell University, Ithaca, NY, 1975
  • Under supervision of Prof. David Gries
• First complete logic for partial correctness of concurrent programs that communicate using shared variables
Interference Freedom

- **Interference Freedom**: every assertion used in the local verification is not invalidated by the execution of the other process.

\[
P_1: \{ p_1 \} \quad c_1 \quad \{ p_2 \} \quad c_2 \quad \ldots
\]

\[
P_2: \{ q_1 \} \quad a_1 \quad \{ q_2 \} \quad a_2 \quad \ldots
\]

We say that they are interference free iff

\[
\forall p_i \in \text{assertions of } P_1 \land \forall a_j \in \text{atomic actions of } P_2, \quad \{ p_i \land \text{pre } a_j \} \\
\quad a_j \\
\quad \{ p_i \} \\
(\text{and vice versa})
\]

- If \( P_1 \) has \( n \) statements and \( P_2 \) has \( m \) statements, proving interference freedom requires proving \( O(n \times m) \) correctness formulas.
These two proof outlines are correct but not interference free.

For example, the assertion $x = 0$ is not preserved against the atomic action $x := x + 2$:

$$
\begin{align*}
\{x = 0\} & \quad \| \quad \{\text{true}\} & \quad \{x = 0 \land x = 0\} \\
 x := x + 2; & \quad \| \quad x := 0; & \quad x := x + 2; \\
\{x = 2\} & \quad \| \quad \{x = 0\} & \quad \{x = 0\}
\end{align*}
$$

By weakening the postconditions we obtain both correct and interference free proof outlines:

$$
\begin{align*}
\{x = 0\} & \quad \| \quad \{\text{true}\} & \quad \{(x = 0 \lor x = 2) \land x = 0\} & \quad \{x = 0\} \\
 x := x + 2; & \quad \| \quad x := 0; & \quad x := x + 2; & \quad \Rightarrow & \quad x := x + 2; \\
\{x = 0 \lor x = 2\} & \quad \| \quad \{x = 0 \lor x = 2\} & \quad \{x = 0 \lor x = 2\} & \quad \Rightarrow & \quad \{x = 0 \lor x = 2\}
\end{align*}
$$
Rule for Parallel Composition

\[ \vdash \{ P_1 \} c_1 \{ Q_1 \} \quad \vdash \{ P_2 \} c_2 \{ Q_2 \} \quad \text{interference freedom} \]

\[ \vdash \{ P_1 \land P_2 \} c_1 \parallel c_2 \{ Q_1 \land Q_2 \} \]

• This rule is not compositional
• A change in one of the components may affect the proof, not only of the modified component, but also of all the others
- **Rely-Guarantee** is a well-known compositional method for proving Hoare logic properties of concurrent programs.
- Rough idea: instead of trying to write interference-free proofs, explicitly account for the allowed interference.
- No additional interference checks required.
Rely-Guarantee

\[ R, G \vdash \{ P \} \ c \ \{ Q \} \]

If

1. program \( c \) is executed in a state which satisfies \( P \)
2. every state change by another process satisfies \( R \)

then

1. every final state of \( c \) satisfies \( G \)
2. if the execution terminates, the final state will satisfy \( Q \)