Concurrent program logics

Viktor Vafeiadis
Max Planck Institute for Software Systems (MPI-SWS)
EPIT 2018, May 2018
Topics

- Owicki-Gries and rely-guarantee
- Concurrent separation logic
- Combining CSL and RG (RGSep, CAP, Iris)
- Reasoning under weak memory consistency
The Owicki-Gries method

Hoare logic

Hoare triples: \( \{P\} \ C \ \{Q\} \)

- **\( P \): precondition**
  (assertion describing initial state)
- **\( C \): program**
- **\( Q \): postcondition**
  (assertion describing final state if the program terminates)

Proof rules for reasoning about **sequential** programs.

\[
\frac{P \Rightarrow Q[e/x]}{\{P\} x := e \ \{Q\}} \quad \frac{\{P\} C_1 \ \{Q\} \quad \{Q\} C_2 \ \{R\}}{\{P\} C_1; \ C_2 \ \{R\}}
\]
Parallel composition (first attempt)

How about the following rule?

\[
\begin{align*}
\{P_1\} & \quad C_1 \quad \{Q_1\} & \quad & \{P_2\} & \quad C_2 \quad \{Q_2\} \\
\{P_1 \land P_2\} & \quad C_1 || \quad C_2 \quad \{Q_1 \land Q_2\}
\end{align*}
\]
Parallel composition (first attempt)

How about the following rule?

\[
\begin{align*}
\{P_1\} C_1 \{Q_1\} & \quad \{P_2\} C_2 \{Q_2\} \\
\{P_1 \land P_2\} C_1 || C_2 \{Q_1 \land Q_2\}
\end{align*}
\]

This is \textit{unsound} because of \textit{interference}.

\[
\begin{align*}
\{x = 0\} \\
\{x = 0\} || \{\top\} \\
y := 1 \quad x := 1 \\
\{x = 0\} || \{\top\} \\
\{x = 0\}
\end{align*}
\]
Non-interference (second attempt)

Require that the two threads *do not interfere*.

How about the following condition?

\[
\text{vars}(P_1) \cap \text{modified}(C_2) = \emptyset \quad \text{and} \\
\text{vars}(P_2) \cap \text{modified}(C_1) = \emptyset
\]
Non-interference (second attempt)

Require that the two threads do not interfere.

How about the following condition?

\[ \text{vars}(P_1) \cap \text{modified}(C_2) = \emptyset \text{ and } \text{vars}(P_2) \cap \text{modified}(C_1) = \emptyset \]

Too restrictive: cannot verify simple programs.

\[
\begin{align*}
\{x = 0\} \\
x &:= x + 1 \parallel x := x + 2 \\
\{x = 3\}
\end{align*}
\]
Owicki-Gries method (1976)

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

the two proofs are non-interfering

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

Non-interference

\( R \land P \vdash R\{u/x\} \) for every:

- assertion \( R \) in the proof outline of one thread
- assignment \( x := u \) with precondition \( P \) in the proof outline of the other thread
Example: Parallel increment (easy case)

\[
\begin{align*}
\{x = 0\} \\
\quad \quad \quad \quad \quad x := x + 1 \quad \quad \quad \quad \quad x := x + 2 \\
\{x = 3\}
\end{align*}
\]
Example: Parallel increment (easy case)

\[
\begin{align*}
\{x = 0\} & \quad \{x = 0\} \\
\{x = 0 \lor x = 2\} & \quad \{x = 0 \lor x = 1\} \\
x := x + 1 & \quad x := x + 2 \\
\{x = 1 \lor x = 3\} & \quad \{x = 2 \lor x = 3\} \\
\{x = 3\} & \quad \{x = 3\}
\end{align*}
\]
Example: Monotonic counter

\[
\begin{align*}
\{x = 0\} \\
x &:= 1 & a &:= x \\
x &:= 2 & b &:= x \\
\{x = 2 \land b \geq a\}
\end{align*}
\]
Example: Monotonic counter

\[
\begin{align*}
\{x = 0\} & \quad \{x = 0\} \\
\{x = 0\} & \quad a := x \\
x := 1 & \quad \{x = 1\} \\
x := 2 & \quad b := x \\
\{x = 2\} & \quad \{x = 2 \land b \geq a\}
\end{align*}
\]
Example: Monotonic counter

\[
\begin{align*}
\{x = 0\} & \quad \{x = 0\} \quad \top \\
\{x = 0\} & \quad \{x = 1\} \quad \{x \geq a\} \\
x := 1 & \quad a := x \\
\{x = 1\} & \quad \{x \geq a\} \\
x := 2 & \quad b := x \\
\{x = 2\} & \quad \{b \geq a\} \\
\{x = 2 \land b \geq a\} &
\end{align*}
\]
Example: Parallel increment again

Can we prove the following triple?

\[
\begin{align*}
\{ x = 0 \} & \quad \{ x = 2 \} \\
\xrightarrow{x := x + 1} & \quad \xrightarrow{x := x + 1}
\end{align*}
\]

But how can we derive the postcondition \( x = 2 \)?

We need auxiliary variables: i.e. variables that do not affect the program's control flow nor the data flow of the other variables, but record information useful for the proof.
Example: Parallel increment again

We can certainly prove something weaker.

\[
\begin{align*}
\{ & x = 0 \} \\
\{ & x = 0 \lor x = 1 \} & \parallel & \{ & x = 0 \lor x = 1 \} \\
& x := x + 1 & & x := x + 1 \\
\{ & x = 1 \lor x = 2 \} & \parallel & \{ & x = 1 \lor x = 2 \} \\
& \{ & x = 1 \lor x = 2 \} \\
\end{align*}
\]

But how can we derive the postcondition \( x = 2 \)?
Example: Parallel increment again

We can certainly prove something weaker.

\[
\begin{align*}
\{x = 0\} & \quad \{x = 0 \lor x = 1\} & \quad \{x = 0 \lor x = 1\} \\
x := x + 1 & \quad x := x + 1 \\
\{x = 1 \lor x = 2\} & \quad \{x = 1 \lor x = 2\} \\
\{x = 1 \lor x = 2\} & \quad \{x = 1 \lor x = 2\}
\end{align*}
\]

But how can we derive the postcondition \(x = 2\)?

We need *auxiliary* variables:  
*i.e. variables that do not affect the program’s control flow nor the data flow of the other variables, but record information useful for the proof.*
Parallel increment with auxiliary variables

Add two auxiliary variables $a$ and $b$:

*Represent the contribution of each thread to $x$.*

\[
\begin{align*}
\{ x = 0 \} \\
(a, b) &:= (0, 0) \\
\{ x = 2 \}
\end{align*}
\]

\[
\begin{align*}
(x, a) &:= (x + 1, 1) & (x, b) &:= (x + 1, 1) \\
\text{atomic parallel assignment.}
\end{align*}
\]
Parallel increment with auxiliary variables

Add two auxiliary variables \(a\) and \(b\):

Represent the contribution of each thread to \(x\).

\[
\left\{ \begin{array}{l}
  x = 0 \\
  (a, b) := (0, 0)
\end{array} \right\}
\]

\[
\left\{ \begin{array}{l}
  x = a + b \land a = 0 \land b = 0
\end{array} \right\}
\]

\[
\left\{ \begin{array}{l}
  x = a + b \land a = 0 \\
  (x, a) := (x + 1, 1)
\end{array} \right\} \quad \left| \quad \left\{ \begin{array}{l}
  x = a + b \land b = 0 \\
  (x, b) := (x + 1, 1)
\end{array} \right\}
\]

\[
\left\{ \begin{array}{l}
  x = a + b \land a = 1 \\
  (x, a) := (x + 1, 1)
\end{array} \right\} \quad \left| \quad \left\{ \begin{array}{l}
  x = a + b \land b = 1 \\
  (x, b) := (x + 1, 1)
\end{array} \right\}
\]

\[
\left\{ \begin{array}{l}
  x = 2
\end{array} \right\}
\]

\[
(x_1, x_2) := (e_1, e_2) \quad \leadsto \quad \text{atomic parallel assignment.}
\]
Rely-Guarantee reasoning
OG is not compositional.

- Recall the parallel composition rule:

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

the two proofs are non-interfering

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

- The specification of a program \(C\) is not just \(\{P\} \rightarrow \{Q\}\), but also all the intermediate assertions in the outline.

Rely-guarantee \(\sim\) compositional version of OG.
Rely-guarantee specifications

**RG judgment:** $C$ sat $(P, R, G, Q)$

- **$P$** : **precondition**
  (assertion describing initial state)

- **$R$** : **rely condition**
  (relation describing atomic steps of environment)

- **$G$** : **guarantee condition**
  (relation describing atomic steps of the program)

- **$Q$** : **postcondition**
  (assertion describing final state if the program terminates)

\[
\sigma_0 \xrightarrow{\text{env}} \sigma_1 \xrightarrow{\text{prog}} \sigma_2 \xrightarrow{\text{env}} \sigma_3 \xrightarrow{\text{prog}} \sigma_4 \xrightarrow{\text{prog}} \sigma_5 \xrightarrow{\text{env}} \sigma_6 \ldots \sigma_{n-1} \xrightarrow{\text{prog}} \sigma_n
\]

If $P(\sigma_0)$ and all $R(\sigma_i, \sigma_{i+1})$ for all $\sigma_i \xrightarrow{\text{env}} \sigma_{i+1}$,
then $G(\sigma_j, \sigma_{j+1})$ for all $\sigma_j \xrightarrow{\text{prog}} \sigma_{j+1}$, and $Q(\sigma_n)$ (the final state).
Stability

**Definition (Stability)**

An assertion \( P \) is *stable* under a relation \( R \) iff
\[
P(\sigma) \land R(\sigma,\sigma') \Rightarrow P(\sigma')
\]
for all states \( \sigma \) and \( \sigma' \).

**RG judgment:** \( C \ sat (P, R, G, Q) \)

We require that the \( P \) and \( Q \) are stable under \( R \).
- The environment can interfere at the beginning/end.

**Rule for atomic statements**

\[
\begin{align*}
\{P\} \ C \ \{Q \land G\} & \quad C \ is \ atomic \\
P \ stable \ under \ R & \quad Q \ stable \ under \ R \\
\hline
C \ sat (P, R, G, Q)
\end{align*}
\]
Some rules

<table>
<thead>
<tr>
<th>Weakening</th>
<th>Sequential composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P' \Rightarrow P$</td>
<td>$C_1 \text{ sat } (P, R, G, Q)$</td>
</tr>
<tr>
<td>$R' \Rightarrow R$</td>
<td>$C_2 \text{ sat } (Q, R, G, Q')$</td>
</tr>
<tr>
<td>$G \Rightarrow G'$</td>
<td>$C_1; C_2 \text{ sat } (P, R, G, Q')$</td>
</tr>
<tr>
<td>$Q \Rightarrow Q'$</td>
<td></td>
</tr>
<tr>
<td>$C \text{ sat } (P, R, G, Q)$</td>
<td>$C_1 \text{ sat } (P, R, G, Q)$</td>
</tr>
</tbody>
</table>
Parallel composition

**Bottom-up rule:**

\[
\begin{align*}
C_1 \text{ sat } (P_1, R_1, G_1, Q_1) & \quad G_1 \Rightarrow R_2 \\
C_2 \text{ sat } (P_2, R_2, G_2, Q_2) & \quad G_2 \Rightarrow R_1 \\
\hline
C_1 \| C_2 \text{ sat } (P_1 \land P_2, R_1 \land R_2, G_1 \lor G_2, Q_1 \land Q_2)
\end{align*}
\]

- Each thread’s guarantee must imply the other’s rely.

**Top-down rule:**

\[
\begin{align*}
C_1 \text{ sat } (P, R \lor G_2, G_1, Q_1) & \quad G_1 \lor G_2 \Rightarrow G \\
C_2 \text{ sat } (P, R \lor G_1, G_2, Q_2) & \quad Q_1 \land Q_2 \Rightarrow Q \\
\hline
C_1 \| C_2 \text{ sat } (P, R, G, Q)
\end{align*}
\]