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

- S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. Acta Informatica 6(4):319-340 (1976)

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.

$$\frac{P \Rightarrow Q[e/x]}{\{P\}\ x := e\ \{Q\}} \qquad \frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

Parallel composition (first attempt)

How about the following rule?

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 \wedge P_2\}\ C_1 \| C_2\ \{Q_1 \wedge Q_2\}}$$

Parallel composition (first attempt)

How about the following rule?

$$\frac{\{P_1\}\, C_1\, \{Q_1\} \qquad \{P_2\}\, C_2\, \{Q_2\}}{\{P_1 \wedge P_2\}\, C_1 \| C_2\, \{Q_1 \wedge Q_2\}}$$

This is ***unsound*** because of ***interference***.

$$\begin{array}{c}
\left\{x = 0\right\} \\
\begin{array}{l} \left\{x = 0\right\} \\ y := 1 \\ \left\{x = 0\right\} \end{array} \;\Bigg\|\; \begin{array}{l} \left\{\top\right\} \\ x := 1 \\ \left\{\top\right\} \end{array} \\
\left\{x = 0\right\}
\end{array}$$

Non-interference (second attempt)

Require that the two threads ***do not interfere***.

How about the following condition?

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

Non-interference (second attempt)

Require that the two threads ***do not interfere***.

How about the following condition?

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

Too restrictive: cannot verify simple programs.

$$\begin{gathered}\{x = 0\} \\ x := x + 1 \parallel x := x + 2 \\ \{x = 3\}\end{gathered}$$

Owicki-Gries method (1976)

OG = Hoare logic + rule for parallel composition

$$\frac{\{P_1\}\, c_1\, \{Q_1\} \qquad \{P_2\}\, c_2\, \{Q_2\} \qquad \text{the two proofs are } \textit{non-interfering}}{\{P_1 \wedge P_2\}\, c_1 \parallel c_2\, \{Q_1 \wedge Q_2\}}$$

Non-interference

$R \wedge P \vdash R\{u/x\}$ 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)

$$
\begin{array}{c}
\{x = 0\} \\
x := x + 1 \parallel x := x + 2 \\
\{x = 3\}
\end{array}
$$

Example: Parallel increment (easy case)

$$
\begin{array}{c}
\left\{x = 0\right\} \\
\begin{array}{l|l}
\left\{x = 0 \lor x = 2\right\} & \left\{x = 0 \lor x = 1\right\} \\
x := x + 1 & x := x + 2 \\
\left\{x = 1 \lor x = 3\right\} & \left\{x = 2 \lor x = 3\right\}
\end{array} \\
\left\{x = 3\right\}
\end{array}
$$

Example: Monotonic counter

$$\left\{x = 0\right\}$$

$$
\begin{array}{l||l}
x := 1 & a := x \\
x := 2 & b := x
\end{array}
$$

$$\left\{x = 2 \wedge b \geq a\right\}$$

Example: Monotonic counter

$$
\begin{array}{c}
\{x = 0\} \\
\begin{array}{l||l}
\{x = 0\} & \\
x := 1 & a := x \\
\{x = 1\} & \\
x := 2 & b := x \\
\{x = 2\} &
\end{array} \\
\{x = 2 \land b \geq a\}
\end{array}
$$

Example: Monotonic counter

$$\left\{x = 0\right\}$$

$$\begin{array}{l||l}
\left\{x = 0\right\} & \left\{\top\right\} \\
x := 1 & a := x \\
\left\{x = 1\right\} & \left\{x \geq a\right\} \\
x := 2 & b := x \\
\left\{x = 2\right\} & \left\{b \geq a\right\}
\end{array}$$

$$\left\{x = 2 \wedge b \geq a\right\}$$

Example: Parallel increment again

Can we prove the following triple?

$$\{x = 0\}$$

$$x := x + 1 \;\Big\|\; x := x + 1$$

$$\{x = 2\}$$

Example: Parallel increment again

We can certainly prove something weaker.

$$
\begin{array}{c}
\{x = 0\} \\
\begin{array}{l|l}
\{x = 0 \lor x = 1\} & \{x = 0 \lor x = 1\} \\
x := x + 1 & x := x + 1 \\
\{x = 1 \lor x = 2\} & \{x = 1 \lor x = 2\}
\end{array} \\
\{x = 1 \lor x = 2\}
\end{array}
$$

But how can we derive the postcondition $x = 2$?

Example: Parallel increment again

We can certainly prove something weaker.

$$
\begin{array}{c}
\{x = 0\} \\
\begin{array}{l|l}
\{x = 0 \lor x = 1\} & \{x = 0 \lor x = 1\} \\
x := x + 1 & x := x + 1 \\
\{x = 1 \lor x = 2\} & \{x = 1 \lor x = 2\}
\end{array} \\
\{x = 1 \lor x = 2\}
\end{array}
$$

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.

$$\left\{x = 0\right\}$$
$$(a, b) := (0, 0)$$

$$(x, a) := (x + 1, 1) \;\Big\|\; (x, b) := (x + 1, 1)$$

$$\left\{x = 2\right\}$$

$(x_1, x_2) := (e_1, e_2) \quad \leadsto \quad$ atomic parallel assignment.

Parallel increment with auxiliary variables

Add two auxiliary variables a and b:

Represent the contribution of each thread to x*.*

$$\{x = 0\}$$
$$(a, b) := (0, 0)$$
$$\{x = a + b \wedge a = 0 \wedge b = 0\}$$
$$\begin{array}{l|l} \{x = a + b \wedge a = 0\} & \{x = a + b \wedge b = 0\} \\ (x, a) := (x + 1, 1) & (x, b) := (x + 1, 1) \\ \{x = a + b \wedge a = 1\} & \{x = a + b \wedge b = 1\} \end{array}$$
$$\{x = 2\}$$

$(x_1, x_2) := (e_1, e_2) \quad \rightsquigarrow \quad$ atomic parallel assignment.

Rely-Guarantee reasoning

OG is ***not*** compositional.

- Recall the parallel composition rule:

$$\frac{\{P_1\}\, c_1\, \{Q_1\} \qquad \{P_2\}\, c_2\, \{Q_2\} \\ \text{the two proofs are } \textit{non-interfering}}{\{P_1 \wedge P_2\}\, c_1 \parallel c_2\, \{Q_1 \wedge Q_2\}}$$

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

Rely-guarantee $\rightsquigarrow$ 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 $P(\sigma) \wedge R(\sigma, \sigma') \Rightarrow P(\sigma')$ for all states σ and σ'.

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

$$\frac{\{P\}\, C\, \{Q \wedge G\} \qquad C \text{ is atomic} \qquad P \text{ stable under } R \qquad Q \text{ stable under } R}{C \text{ sat } (P, R, G, Q)}$$

Some rules

Weakening

$$\frac{C \text{ sat } (P, R, G, Q) \qquad P' \Rightarrow P \qquad R' \Rightarrow R \qquad G \Rightarrow G' \qquad Q \Rightarrow Q'}{C \text{ sat } (P', R', G', Q')}$$

Sequential composition

$$\frac{C_1 \text{ sat } (P, R, G, Q) \qquad C_2 \text{ sat } (Q, R, G, Q')}{C_1; C_2 \text{ sat } (P, R, G, Q')}$$

Parallel composition

Bottom-up rule:

$$\frac{\begin{array}{cc} C_1 \text{ sat } (P_1, R_1, G_1, Q_1) & G_1 \Rightarrow R_2 \\ C_2 \text{ sat } (P_2, R_2, G_2, Q_2) & G_2 \Rightarrow R_1 \end{array}}{C_1 \| C_2 \text{ sat } (P_1 \wedge P_2, R_1 \wedge R_2, G_1 \vee G_2, Q_1 \wedge Q_2)}$$

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

Top-down rule:

$$\frac{\begin{array}{cc} C_1 \text{ sat } (P, R \vee G_2, G_1, Q_1) & G_1 \vee G_2 \Rightarrow G \\ C_2 \text{ sat } (P, R \vee G_1, G_2, Q_2) & Q_1 \wedge Q_2 \Rightarrow Q \end{array}}{C_1 \| C_2 \text{ sat } (P, R, G, Q)}$$