COMP4161: Advanced Topics in Software Verification

$P || Q$

Gerwin Klein, June Andronick, Christine Rizkallah, Miki Tanaka
S2/2018

data61.csiro.au

Content

- Intro & motivation, getting started [1]
- Foundations & Principles
  - Lambda Calculus, natural deduction [1,2]
  - Higher Order Logic [3[a]]
  - Term rewriting [4]
- Proof & Specification Techniques
  - Inductively defined sets, rule induction [5]
  - Datatypes, recursion, induction [6, 7]
  - Hoare logic, proofs about programs, invariants [8[b],9]
  - (mid-semester break)
  - C verification [10]
  - CakeML, Isar [11[c]]
  - Concurrency [12]

[a] a1 due; [b] a2 due; [c] a3 due

Program verification so far

If the following true?

$\{x = 0\}$
$y := x;$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

YES!

 © Data61, CSIRO: provided under Creative Commons Attribution License

Program verification with concurrency

Is it still true?

$$\{x = 0\}$$
$$y := x; \qquad\qquad || \quad x := 4$$
$$x := x + 1;$$
$$\{x = 1 \wedge y = 0\}$$

NO!

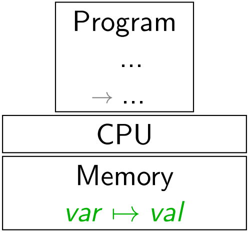
 © Data61, CSIRO: provided under Creative Commons Attribution License

Program verification so far

So far we have assumed **sequential execution**

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

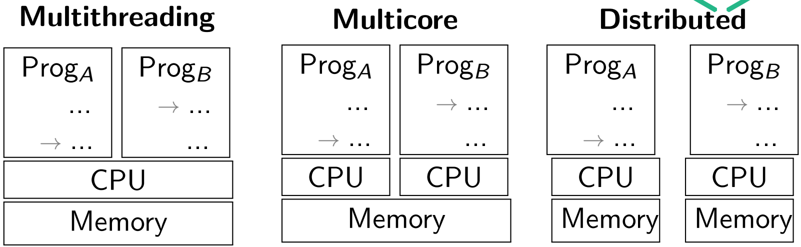
i.e. a single thread of execution accessing the memory state



This is not always the case!

© Data61, CSIRO: provided under Creative Commons Attribution License

Types of concurrency



All need communication and synchronisation mechanisms

Multithreading	Multicore	Distributed
Shared memory Interleaved execution	Shared memory Parallel execution	Message passing

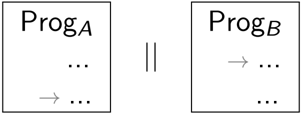
Here: we'll look at shared-memory concurrency

(and we'll ignore further complications such as caches, weak memory, ...)

Goal

We want to be able to reason about parallel composition of programs:

$\{$*precondition*$\}$



$\{$*postcondition*$\}$

2 kinds of properties:

Safety:
"something bad does not happen"
(no bad state can be reached)
e.g. $\{x = 0\}$

With concurrency: much harder!

Liveness:
"something good must happen"
(specific states must be reached)
e.g. the program terminates

With concurrency: new problems!

 © Data61, CSIRO: provided under Creative Commons Attribution License

Goal

We want to be able to reason about parallel composition of programs:

$\{\textit{precondition}\}$

Prog_A		Prog_B
$\dots$	$\|$	$\rightarrow \dots$
$\rightarrow \dots$		$\dots$

$\{\textit{postcondition}\}$

Here:

- ➜ We focus on **safety** properties: postcondition holds **if reached**
- ➜ We will define parallel composition ($\|$) as non-deterministic interleaving
- ➜ We go back to our minimal IMP language (forget about C and monads)

datatype com = SKIP

 © Data61, CSIRO: provided under Creative Commons Attribution License

Program verification so far

If the following true?

$\{x = 0\}$
$y := x;$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

YES!

© Data61, CSIRO: provided under Creative Commons Attribution License

Program verification with concurrency

Is it still true?

$\{x = 0\}$
$y := x; \qquad || \quad x := 4$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

NO!

What is going wrong?
What do we need to change?

- ➜ to make sure we don't prove wrong statements!
- ➜ to allow us to prove true statements about concurrent programs

© Data61, CSIRO: provided under Creative Commons Attribution License

Program verification so far

How would we have proved this?

$\{x = 0\} \implies \{x + 1 = 1 \wedge x = 0\}$
$y := x;\ \{x + 1 = 1 \wedge y = 0\}$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

Using Hoare logic rules!

$$\frac{\vdash \{P\}\ c_1\ \{R\} \quad \vdash \{R\}\ c_2\ \{Q\}}{\vdash \{P\}\ c_1; c_2\ \{Q\}}$$

$$\frac{}{\vdash \{P[x \mapsto e]\}\ x := e\ \{P\}}$$

Why does this make it true? What does it mean that it's true?

It means:

If the program "$y := x;\ x := x + 1$" is executed from a state satisfying $\{x = 0\}$ *then, if it terminates, the resulting state satisfied* $\{x = 1 \wedge y = 0\}$

 © Data61, CSIRO: provided under Creative Commons Attribution License

Program verification so far

How would we have proved this?

$\{x = 0\} \implies \{x + 1 = 1 \wedge x = 0\}$
$y := x;\ \{x + 1 = 1 \wedge y = 0\}$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

Using Hoare logic rules!

$$\frac{\vdash \{P\}\ c_1\ \{R\} \quad \vdash \{R\}\ c_2\ \{Q\}}{\vdash \{P\}\ c_1; c_2\ \{Q\}}$$

$$\frac{}{\vdash \{P[x \mapsto e]\}\ x := e\ \{P\}}$$

Why does this make it true? What does it mean that it's true?

It means:

$$\langle y := x;\ x := x + 1, \sigma\rangle \rightarrow \sigma' \ \wedge\ x\ \sigma = 0 \quad \longrightarrow \quad x\ \sigma' = 1 \ \wedge\ y\ \sigma' = 0$$

Where:

$$\frac{\langle c_1, \sigma\rangle \rightarrow \sigma' \quad \langle c_2, \sigma'\rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow \sigma''} \qquad \frac{e\ \sigma = v}{\langle \mathrm{x} := \mathrm{e}, \sigma\rangle \rightarrow \sigma[x \mapsto v]}$$

Program verification with concurrency

$\{x = 0\}$
$y := x;$ ~~$\{x + 1 = 1 \wedge y = 0\}$~~ $\quad || \quad x := 4$
$x := x + 1;$
$\{x = 1 \wedge y = 0\}$

- Execution is interleaved; Intermediate assertions can be interferred with

- Need a new reasoning framework!
- New syntax, new semantics,
  new proof rules (proved sound w.r.t semantics), new VCG

- (1969: Hoare Logic (Tony Hoare))
- 1976: Owicki-Gries (Susan Owicki and David Gries)
- 1981: Rely-Guarantee (Cliff Jones)

© Data61, CSIRO: provided under Creative Commons Attribution License

Owicki-Gries framework

Intuition:

- Syntax: our IMP language + Parallel operator + Await operator
- Semantics:
  - $P \,||\, Q$: pick one program and execute its current instruction
  - *AWAIT b DO c OD*: if guard is true execute c atomically
- Proof rules:
  - you prove *local correctness* (as before)
  - your prove *interference-freedom* (assertions not interfered with)

$$\begin{array}{l|l} \{is\_even\ x\} & \\ x := x + 1;\ \{is\_even\ x + 1\} & x := x + 2 \\ x := x + 1; & \\ \{is\_even\ x\} & \end{array}$$

➜ **Needs a fully annotated program!**

➜ **Needs a "small-step semantics"** $\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$

 © Data61, CSIRO: provided under Creative Commons Attribution License

Owicki-Gries framework

Formally:

- Syntax: our IMP language + Parallel operator + Await operator
- Semantics:

$$\frac{\langle c_1, \sigma\rangle \rightarrow \langle c_1', \sigma'\rangle}{\langle c_1 || c_2, \sigma\rangle \rightarrow \langle c_1' || c_2, \sigma'\rangle} \qquad \frac{\langle c_2, \sigma\rangle \rightarrow \langle c_2', \sigma'\rangle}{\langle c_1 || c_2, \sigma\rangle \rightarrow \langle c_1 || c_2', \sigma'\rangle}$$

- Hoare rules:

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \quad \{P_2\}\ c_2\ \{Q_2\} \quad \mathit{interfree}\ c_1\ c_2 \quad \mathit{interfree}\ c_2\ c_1}{\{P_1 \wedge P_2\}\ c_1 || c_2\ \{Q_1 \wedge Q_2\}}$$

Where

$\mathit{interfree}\ c_1\ c_2 \equiv$

$\forall p \in (\mathit{assertions}\ c_1).\ \forall (a, c) \in (\mathit{atomics}\ c_2).\ \{p \wedge a\} c \{p\}$

 © Data61, CSIRO: provided under Creative Commons Attribution License

Owicki-Gries framework

- **Quadratic explosion of proof obligations! (verification conditions)**
- **Not compositional**
- **Not complete:** sometimes need auxilliary/ghost variables

$$\{x = 0\}$$
$$x := x + 1; \parallel x := x + 1$$
$$\{x = 2\}$$

$$\{x = 0 \land a_1 = 0 \land a_2 = 0\}$$

$$\begin{array}{r|l}
\land a_1 = 0 & \land a_2 = \\
\{a_2 = 0 \land x = 0 \ \lor \ a_2 = 1 \land x = 1\} & \{a_1 = 0 \land x = 0 \ \lor \ a_1 = 1 \land x \\
< x := x + 1; a_1 := 1 > & < x := x + 1; a_2 := 1 > \\
\{a_2 = 0 \land x = 1 \ \lor \ a_2 = 1 \land x = 2\} & \{a_1 = 0 \land x = 1 \ \lor \ a_1 = 1 \land x \\
\land a_1 = 1 & \land a_2 =
\end{array}$$

 © Data61, CSIRO: provided under Creative Commons Attribution License

Demo

Rely-Guarantee?

Intuition:

- Syntax, semantics: as before (but no need for assertions)
- Proof rules:
  - each program is specified in isolation, **assuming a behavior of the "environment"** (other programs in parallel)
  - each program has: precondition, postcondition, **rely and guarantee**
  - rely and guarantee are **relations between 2 states**
  - **rely** expresses the maximum behavior of the environment (the interference that the program can tolerate)
  - **guarantee** expresses a maximum behavior promised to the environment

$$\underset{\{P, R, G, Q\}}{\text{c}} \;\Big\|\; \underset{\{P', R', G', Q'\}}{\text{c'}}$$

$$\sigma_0 \xrightarrow{c} \sigma_1 \xrightarrow{c} \sigma_2 \xrightarrow{c'} \sigma_3 \xrightarrow{c} \sigma_4 \xrightarrow{c'} \sigma_5 \xrightarrow{c'} \sigma_6 \xrightarrow{c} \sigma_7$$

Rely-Guarantee?

Formally:

- Syntax, semantics: as before (but no need for assertions)
- Proof rules (examples):

$$\frac{P \subseteq \{s.\ f\ s \in Q\} \quad \{(s,t).\ P\ s \wedge (t = f\ s \vee t = s)\} \subseteq G \quad \mathit{stable}\ P\ R \quad \mathit{stable}\ Q\ R}{\mathit{Basic}\ f \{P, R, G, Q\}}$$

$$\frac{c_1\{P_1, R_1, G_1, Q_1\} \quad c_2\{P_2, R_2, G_2, Q_2\} \quad G_1 \subseteq R_2 \quad G_2 \subseteq R_1}{c_1 || c_2 \{P_1 \cap P_2, R_1 \cap R_2, G_1 \cup G_2, Q_1 \cap Q_2\}}$$

Where $\mathit{stable}\ P\ R\ = \ \forall\ \sigma\ \sigma'.\ (P\sigma \wedge R(\sigma,\sigma')) \rightarrow P\sigma'$
(*doing an environment step before or after P should not make P invalid*)

Intuition: the guarantee of one program is the rely of the other program

© Data61, CSIRO: provided under Creative Commons Attribution License

Demo

We have seen today ...

- ➜ Need for new reasoning framework for parallel/concurrent programs
- ➜ Owicki-Gries
- ➜ Rely-Guarantee

© Data61, CSIRO: provided under Creative Commons Attribution License