Formal methods

Lecture 12

Concurrency and Communication with Shared Variables

Lecture is based on the book by

Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*

Non-deterministic programs

Value Expression	$e ::=$	$\mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Statement	$S ::=$	$\mathbf{skip} \mid x := e \mid S_1 ; S_2 \mid G \mid \star G$
Guarded Command	$G ::=$	$[[]_{i=1}^{n} b_i \rightarrow S_i]$

- Guarded command $[[]_{i=1}^{n} b_i \rightarrow S_i]$, also written as $[b_1 \rightarrow S_1 [] \ldots [] b_n \rightarrow S_n]$, terminates if none of the boolean guards b_i evaluate to true. Otherwise, non-deterministically select one of the b_i that evaluates to true and execute the corresponding statement S_i.
- Iteration $\star G$ indicates repeated execution of guarded command G as long as at least one of the guards evaluates to true. When none of the guards evaluate to true $\star G$ terminates.

We use shortened notation:

`if` $[]^{n}_{i=1} b_i \rightarrow S_i$ `fi`	➩	$[]^{n}_{i=1} b_i \rightarrow S_i$
`do` $[]^{n}_{i=1} b_i \rightarrow S_i$ `od`	➩	$\star([]^{n}_{i=1} b_i \rightarrow S_i)$

Recall: proof system for sequential programs

Axioms:

$\{p\}$ `skip` $\{p\}$

$\{q[e/x]\}\ \ x{:=}e\ \{q\}$

Consequence rule:

$$\frac{p \Rightarrow p_0,\ \{p_0\}\ S\ \{q_0\},\ q_0 \Rightarrow q}{\{p\}\ S\ \{q\}}$$

Derived rules:

$$\frac{p\ \Rightarrow\ q}{\{p\}\ \texttt{skip}\ \{q\}}$$

$$\frac{p \Rightarrow q[e/x]}{\{p\}\ x{:=}e\ \{q\}}$$

$$\frac{\{p\}\ S_1\ \{r\},\ \ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;S_2\ \{q\}}$$

$$\frac{\{p \wedge b_i\}\ S_i\ \{q_i\},\ \textit{for all } i \in \{1,\ldots,n\}}{\{p\}\ [[]_{i=1}^{n}\, b_i \rightarrow S_i]\ \{(p \wedge \neg b_G) \vee \bigvee_{i=1}^{n} q_i\}}$$

$$\frac{\{p \wedge b_G\}\ G\ \{p\}}{\{p\}\ \star G\ \{p \wedge \neg b_G\}}$$

Proof Outline

PO (Proof Outline) is Hoare triple (with annotated program) for which all the verification conditions (VC) are provable using given program annotations.

Example of PO:

$\{ x = a \}$
$x := x + 1;$
$\{ x = a + 1 \}$
skip
$\{ x + a = 2a + 1 \}$

If not assignment

Annotated commands:

$$AS ::= \quad \textbf{skip} \mid x := e \mid \textbf{await}\ b\ \textbf{then}\ S\ \textbf{end} \mid AS_1 ; \{p\} AS_2 \mid AG \mid \star \{p\} AG$$
$$AG ::= \quad [\,[]_{i=1}^{n} b_i \rightarrow \{p_i\} AS_i \{q_i\}\,]$$

Annotated commands

$$\begin{array}{ll} AS ::= & \textbf{skip} \mid x := e \mid \textbf{await } b \textbf{ then } S \textbf{ end} \mid AS_1; \boxed{\{p\}} AS_2 \mid AG \mid \star \boxed{\{p\}} AG \\ AG ::= & [\,[\!]_{i=1}^n b_i \to \boxed{\{p_i\}} AS_i \boxed{\{q_i\}}] \end{array}$$

- $\{p\}$ **skip** $\{q\}$ *is a proof outline iff* $p \leftrightarrow q$.
- $\{p\}\ x := e\ \{q\}$ *is a proof outline iff* $p \Rightarrow q[e/x]$.
- $\{p\}$ **await** b **then** S **end** $\{q\}$ *is a proof outline iff there exists a proof outline*
  $\{p \wedge b\}\ AS\ \{q\}$ *with* $Progr(AS) = S$.
- $\{p\}\ AS_1; \{r\}\ AS_2\ \{q\}$ *is a proof outline iff* $\{p\}\ AS_1\ \{r\}$ *and* $\{r\}\ AS_2\ \{q\}$ *are proof outlines.*
- $\{p\}\ [\,[\!]_{i=1}^n b_i \to \{p_i\}\ AS_i\ \{q_i\}\,]\ \{q\}$ *is a proof outline iff* $p \wedge b_i \Rightarrow p_i$ *and* $p \wedge \neg(\bigvee_{i=1}^n b_i) \Rightarrow q$ *hold,* $\{p_i\}\ AS_i\ \{q_i\}$ *are proof outlines, for* $i = 1, \ldots, n$, *and* $\bigvee_{i=1}^n q_i \Rightarrow q$.
- $\{p\}\ \star\{I\}\ AG\ \{q\}$ *is a proof outline iff* $p \Rightarrow I$, $\{I \wedge b_{AG}\}\ AG\ \{I\}$ *is a proof outline, and* $I \wedge \neg b_{AG} \Rightarrow q$.

Parallel programming language with shared variables

Expression	$e ::=$	$\vartheta \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Command	$S ::=$	$\textbf{skip} \mid x := e \mid \boxed{\textbf{await}\ b\ \textbf{then}\ S\ \textbf{end}} \mid$
		$S_1; S_2 \mid G \mid \star G$
Guarded Command	$G ::=$	$[\,[]_{i=1}^{n} b_i \rightarrow S_i]$
Program	$P ::=$	$S_1 \| \cdots \| S_n$

- **await** b **then** S **end** is blocked in any state in which b evaluates to false. If b evaluates to true then S can be executed atomically, i.e., without being interrupted by other components.
- $S_1 \| \cdots \| S_n$ indicates parallel execution of the commands $S_1, \ldots, S_n$. The components $S_1, \ldots, S_n$ of a parallel composition are often called *processes*.
- **await** $b \equiv$ **await** b **then skip end**
- $< S > \equiv$ **await** *true* **then** S **end**

The brackets $< \cdots >$ are sometimes called "Lamport brackets" and $< S >$ is also called a "bracketed section" or "atomic region".

Execution model: atomicity and interleaving

- An assignment $x := e$ is executed *atomically*, that is, during its execution other parallel processes may not change x or the variables occurring in e.
- For an await statement **await** b **then** S **end** we assume that S is executed atomically in a state where b holds.
- Concurrent processes proceed asynchronously. No assumptions are made about the relative speed at which processes execute their actions.

Interleaving semantics: only one atomic action of one of the processes that is not in the waiting state is executed at a time. It is called interleaving of atomic actions.

What is the value of x after the execution of the following program?

$$(x := 0; x := x + 2) \parallel (x := 1; x := x + 3)$$

It can be either 2, 4, 5 or 6.

Proof system for parallel programs

All the rules and axioms of the sequential (non-deterministic) programs apply, but we need new rules for new consturcts **await** and ||.

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

$$\frac{\text{There exist proof outlines } \{p_i\}AS_i\{q_i\},\ i = 1,\ldots,n.}{\{p_1 \wedge \ldots \wedge p_n\}\ Progr(AS_1) \parallel \ldots \parallel Progr(AS_n)\ \{q_1 \wedge \ldots \wedge q_n\}}$$

But the last rule about parallel composition is not valid. Consider:

$\{ x = 0\}\ \ x := x + 2\ \{ x = 2\}$ and $\{ x = 0\}\ \ y := x\ \{ y = 0\}$ are POs,
but $\{ x = 0\}\ \ x := x + 2 \parallel y := x\ \{ x = 2 \wedge y = 0\}$ is not valid

Interference freedom

- Annotation specifies the constraint what program variables have to satisfy when the execution has reached the place/state where annotation is written.
- It is difficult to locate the place for annotations in the parallel program because the global annotations should take into account all possible interleavings.
- It is not enough to prove the correcness of processes locally.
- Local annotations suffice only if we can prove that other processes do not interfere with the validity of assertions in this process.

Definition: The POs $\{p_i\}\ AS_i\ \{q_i\}\ i = 1, \ldots, n$ are ***interference free*** iff for all $i,j \in \{1, \ldots, n\}, i \neq j$, for every assertion r in $\{p_j\}\ AS_j\ \{q_j\}$ we have that if S_i is $x := e$ or `await` b `then` S_0 `end` occurs in $\{p_i\}\ AS_i\ \{q_i\}$ with precondition r_i then $\{r_i \wedge r\}\ S_i\ \{r\}$.

$$\frac{\textit{There exist proof outlines } \{p_i\}AS_i\{q_i\},\ i = 1, \ldots, n, \textit{ that are interference free}}{\{p_1 \wedge \ldots \wedge p_n\}\ Progr(AS_1) \parallel \ldots \parallel Progr(AS_n)\ \{q_1 \wedge \ldots \wedge q_n\}}$$

Proof rules for shared-variable parallel programs

Axioms:

$\{p\}$ **skip** $\{p\}$

$\{q[e/x]\}\ \ x{:=}e\ \{q\}$

Consequence rule:

$$\frac{p \Rightarrow p_0,\ \{p_0\}\ S\ \{q_0\},\ q_0 \Rightarrow q}{\{p\}\ S\ \{q\}}$$

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

$$\frac{\{p\}\ S_1\ \{r\},\ \ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;S_2\ \{q\}}$$

$$\frac{\{p \wedge b_i\}\ S_i\ \{q_i\},\ \mathit{for\ all}\ i \in \{1,\ldots,n\}}{\{p\}\ [[]_{i=1}^{n}\, b_i \rightarrow S_i]\ \{(p \wedge \neg b_G) \vee \bigvee_{i=1}^{n} q_i\}}$$

$$\frac{\{p \wedge b_G\}\ G\ \{p\}}{\{p\}\ \star G\ \{p \wedge \neg b_G\}}$$

$$\frac{\mathit{There\ exist\ proof\ outlines}\ \{p_i\}AS_i\{q_i\},\ i = 1,\ldots,n,\ \mathit{that\ are\ interference\ free}}{\{p_1 \wedge \ldots \wedge p_n\}\ \mathit{Progr}(AS_1)\ \|\ \ldots\ \|\ \mathit{Progr}(AS_n)\ \{q_1 \wedge \ldots \wedge q_n\}}$$

Example

Prove that $\{ x = 0 \}\ \ x := x + 1 \ \ \| \ \ x := x + 2 \ \ \{ x = 3 \}$

Add the annotations:

$\{x = 0 \vee x = 2\}$ $\{x = 1 \vee x = 3\}$

$$\{ x = 0 \}\ \ x := x + 1 \ \ \| \ \ x := x + 2 \ \ \{ x = 3 \}$$

$\{x = 0 \vee x = 1\}$ $\{x = 2 \vee x = 3\}$

The global precondition implies the local preconditions of the processes and the local postconditions imply the global postcondition:

$\vdash \ (x = 0) \Rightarrow \ (x = 0 \vee x = 2) \wedge (x = 0 \vee x = 1)$

$\vdash \ (x = 1 \vee x = 3) \wedge (x = 2 \vee x = 3) \Rightarrow (x = 3)$

Each processes are POs:

$\vdash \{x = 0 \vee x = 2\}\ x := x + 1\ \{x = 1 \vee x = 3\}$

$\vdash \{x = 0 \vee x = 1\}\ x := x + 2\ \{x = 2 \vee x = 3\}$

Example: interference test

$$\boxed{\{x = 0 \vee x = 2\}} \quad \boxed{\{x = 1 \vee x = 3\}}$$

$$\{\ x = 0\ \}\ \ x := x + 1 \ \ \|\ \ x := x + 2\ \ \{\ x = 3\ \}$$

$$\boxed{\{x = 0 \vee x = 1\}} \quad \boxed{\{x = 2 \vee x = 3\}}$$

P1 does not interfere to P2 local precondition

$$\vdash \{(x = 0 \vee x = 2) \wedge (x = 0 \vee x = 1)\}\ x := x + 1 \{\ x = 0 \vee x = 1\}$$

P1 does not interfere to P2 local postcondition

$$\vdash \{(x = 0 \vee x = 2) \wedge (x = 2 \vee x = 3)\}\ x := x + 1 \{\ x = 2 \vee x = 3\}$$

P2 does not interfere to P1 local precondition

$$\vdash \{(x = 0 \vee x = 1) \wedge (x = 0 \vee x = 2)\}\ x := x + 2 \{\ x = 0 \vee x = 2\}$$

P2 does not interfere to P1 local postcondition

$$\vdash \{(x = 0 \vee x = 1) \wedge (x = 1 \vee x = 3)\}\ x := x + 2 \{\ x = 1 \vee x = 3\}$$

A problem

We cannot prove that

$$\{\ x = 0\ \}\ \ x := x + 1 \ \ \|\ \ x := x + 1 \ \ \{\ x = 2\ \}$$

because POs

$$\{(x = 0 \vee x = 1)\}\ x := x + 1 \{\ x = 1 \vee x = 2\}$$

$$\{(x = 0 \vee x = 1)\}\ x := x + 1 \{\ x = 1 \vee x = 2\}$$

are not interference free:

$$\nvdash \quad \{(x = 0 \vee x = 1) \wedge (x = 0 \vee x = 1)\}\ x := x + 1 \{\ x = 0 \vee x = 1\}$$

and the conjunction of local postconditions does no imply postcondition

$$\nvdash \quad \{\ x = 1 \vee x = 2\} \wedge \{\ x = 1 \vee x = 2\} \Rightarrow \{\ x = 2\ \}$$

Conclusions

- Proving the properties of parallel programs is hard
- There is an exponential amount of verification conditions to check
  - one VC for every command in all processes for every assignment
- The presented proof method is not compositional for parallel composition
  - it is not possible to do parallel composition of processes knowing only the pre- and postcondition of the local process.
- If the specification is proved for the whole parallel program then it is possible to compose it sequentially to other programs looking only at pre- and postcondition
- A Rely-Guarantee method exists which has a compositional parallel composition property also

Exercise

Prove that

$$\{\neg done_1 \wedge \neg done_2 \wedge x = 0\}$$
$$< x := x + 1; done_1 := true > \| < x := x + 1; done_2 := true >$$
$$\{x = 2\}.$$

using PO:

$$\{\neg done_1 \wedge (\neg done_2 \Rightarrow x = 0) \wedge (done_2 \Rightarrow x = 1)\}$$
$$< x := x + 1; done_1 := true >$$
$$\{done_1 \wedge (\neg done_2 \Rightarrow x = 1) \wedge (done_2 \Rightarrow x = 2)\}$$

Exercise

1. Annotate and prove the correctness of the following triple

$$\{x \geq 0\}\ \ y := 1; \star[y \times y \leq x \rightarrow y := y + 1\,]; y := y - 1\ \ \{y^2 \leq x < (y+1)^2\}$$