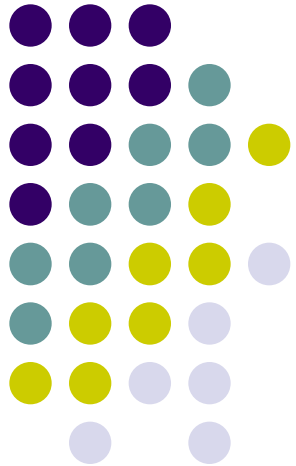
Formal methods

Lecture 13



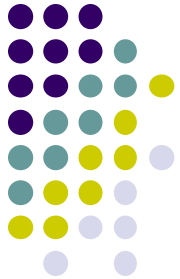
Parallel programs with message passing

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*

Different programs



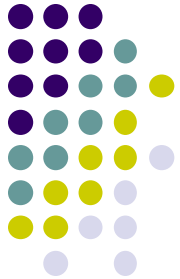
We have studied the formal (syntactic) verification of

- deterministic and non-deterministic programs - an abstraction of sequential programs;
- parallel programs with shared variables - an abstraction of multi-threaded programs (in multiprocessor computer).

We will look next

- parallel programs with message passing - an abstraction of distributed (networked) programs
  - where syntax inspired by language Occam.

Communication primitives of parallel programs



- We have communication primitives $C!e$ and $C?x$ sending a value to channel C and reading the value from C.

Notations

$C \in CHANNEL$;

e – arithmetic expression on the local variables of the process;

x – local variable;

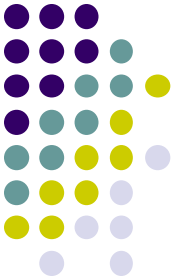
$C!e$ – the value of an expression e is sent to channel C;

$C?x$ – a value is read from channel C and assigned to variable x.

Synchrony!

Commands $C!e$ and $C?x$ are executed synchronously.

Recall: non-deterministic programs



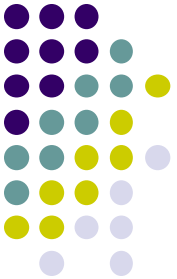
Value Expression	$e ::=$	$\mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Command	$S ::=$	$\textbf{skip} \mid x := e \mid S_1; S_2 \mid G \mid \star G$
Guarded Command	$G ::=$	$[[]_{i=1}^{n} b_i \rightarrow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

- 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 command 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.

`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` ⇨ $\star([]^n_{i=1}\, b_i \rightarrow S_i)$

Parallel programs with message passing



Expression	$e ::=$	$\mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Command	$S ::=$	$\textbf{skip} \mid x := e \mid c!e \mid c?x \mid$
		$S_1; S_2 \mid G \mid \star G$
Guarded Command	$G ::=$	$[]_{i=1}^{n} b_i \rightarrow S_i] \mid []_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i]$
Program	$P ::=$	$S_1 \| \cdots \| S_n$

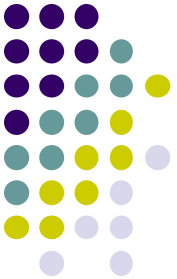
- Output statement $c!e$ is used to send the value of expression e on channel c as soon as a corresponding input command is available. Since we assume synchronous communication, such an output command is suspended until a parallel process executes an input command $c?x$.
- Input command $c?x$ is used to receive a value via channel c and assign this value to the variable x. As for the output command , such an input statement has to wait for a corresponding partner before a (synchronous) communication can take place.

Parallel programs with message passing

- Guarded command $[]_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i]$. A guard (the part before the arrow) is *open* if its boolean part evaluates to true. If none of the guards is open, the guarded command terminates after evaluation of the booleans. Otherwise, wait until the communication of one of the open guards can be performed and continue with the corresponding S_i.
- $S_1 \| \cdots \| S_n$ 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*.

For a guarded command $G \equiv []_{i=1}^{n} b_i \rightarrow S_i]$ or $G \equiv []_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i]$, we define $b_G \equiv b_1 \vee \ldots \vee b_n$. An io-guard $b_i; c_i?x_i$ is often shortened to $c_i?x_i$ if $b_i \equiv true$.

Syntactic restrictions

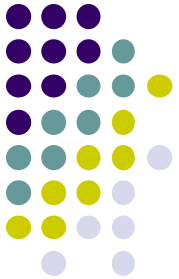


- For $S_1; S_2$ we require that if S_1 contains $c!e$ then S_2 does not contain $c?x$, and if S_1 contains $c?x$ then S_2 does not contain $c!e$.
- For $[[]_{i=1}^{n} b_i \rightarrow S_i]$ we require that, for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, if S_i contains $c!e$ then S_j does not contain $c?x$.
- For $[[]_{i=1}^{n} b_i; c_i?x_i \rightarrow S_i]$ we require that, for all $i, j \in \{1, \ldots, n\}$, S_j does not contain $c_i!e$, and if S_i contains $c!e$ then, for $j \neq i$, S_j does not contain $c?x$.
- For $S_1 \| \cdots \| S_n$ we require that, for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, if S_i contains $c!e_1$ then S_j does not contain $c!e_2$, and if S_i contains $c?x_1$ then S_j does not contain $c?x_2$.

Furthermore, parallel processes do not share program variables

- For $S_1 \| \cdots \| S_n$, we require, for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, that $var(S_i) \cap var(S_j) = \emptyset$.

Recall: proof method - proof outline



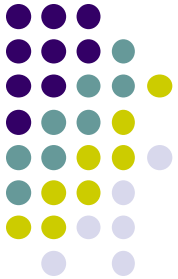
PO (Proof Outline) is Hoare triple (with annotated program) for which all the verification conditions are provable using given program annotations.

$\{ x = a \}$
$x := x + 1;$
$\{ x = a + 1 \}$
$x := x + 2;$
$\{ x = a + 3 \}$

Annotated commands:

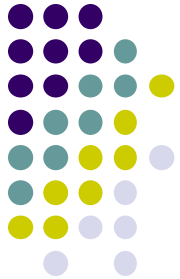
$$AS ::= \textbf{skip} \mid x := e \mid \textbf{await } b \textbf{ then } S \textbf{ end} \mid AS_1 ; \{p\} AS_2 \mid AG \mid \star \{p\} AG$$
$$AG ::= [] _{i=1}^{n} b_i \rightarrow \{p_i\} AS_i \{q_i\}$$

Proof method - proof outline



- $\{p\}$ **skip** $\{q\}$ *is a proof outline iff* $p \leftrightarrow q$.
- $\{p\}\ x := e\ \{q\}$ *is a proof outline iff* $p \Rightarrow q[e/x]$.
- $\{p\}\ AS_1; \{r\}\ AS_2\ \{q\}$ *is a proof outline iff* $\{p\}\ AS_1\ \{r\}$ *and* $\{r\}\ AS_2\ \{q\}$ *are proof outlines.*
- $\{p\}\ [[]_{i=1}^{n} b_i \rightarrow \{p_i\}\ AS_i\ \{q_i\}]\ \{q\}$ *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* $I \wedge \neg b_{AG} \Rightarrow q$.
- $\{p\}\ c!e\ \{q\}$ *is a proof outline,*
- $\{p\}\ c?x\ \{q\}$ *is a proof outline,*
- $\{p\}\ [[]_{i=1}^{n} b_i; c_i?x_i \rightarrow \{p_i\}\ AS_i\ \{q_i\}]\ \{q\}$ *is a proof outline iff* $p \wedge \neg(\bigvee_{i=1}^{n} b_i) \Rightarrow q$, $\{p_i\}\ AS_i\ \{q_i\}$ *are proof outlines, for* $i = 1, \ldots, n$, *and* $\bigvee_{i=1}^{n} q_i \Rightarrow q$.

Recall: proof method for parallel programs with shared variables



- The method of Owicki and Gries
  - First, a *local* correcness proof is given for each of processes by means of proof outlines. Proved by proof outlines.
  - In the second, *global*, stage a consitency check is applied to the local proof outlines. This is the *interference* test which verifies that assertions in the proof outline of one process remain valid under actions of other processes.

- The similar two-stage method applies to the parallel processes with message passing also, but the *cooperation* tests are verified instead of *interference* tests

Cooperation

The proof outlines $\{p_i\}\ AS_i\ \{q_i\}$, *for* $i = 1, \ldots, n$, cooperate *iff*

(i) *assertions occurring in* $\{p_i\}\ AS_i\ \{q_i\}$ *do not refer to variables of* $Progr(AS_j)$, *for* $j \neq i$,

(ii) *for any channel* c, *if* $\{r_1^i\}\ c!e\ \{r_2^i\}$ *occurs in* $\{p_i\}\ AS_i\ \{q_i\}$ *and* $\{r_1^j\}\ c?x\ \{r_2^j\}$ *occurs in* $\{p_j\}\ AS_j\ \{q_j\}$ *then* $\{r_1^i \wedge r_1^j\}\ x := e\ \{r_2^i \wedge r_2^j\}$.

The formula $\{r_1^i \wedge r_1^j\}\ x := e\ \{r_2^i \wedge r_2^j\}$ can be proved by means of proof outlines or using PS_{seq}.

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

Example

$$\{y=3\}\ \ (c?x;\, x:=x+1;\, d!(x+2))\ \|\ (c!y;\, d?y;\, y:=y+2)\ \ \{x=4\wedge y=8\}.$$

$\{true\}\ c?x;\ \{x=3\}\ x:=x+1;\ \{x=4\}\ d!(x+2)\ \{x=4\}$, *and*
$\{y=3\}\ c!y;\ \{y=3\}\ d?y;\ \{y=6\}\ y:=y+2\ \{y=8\}$.

local annotations

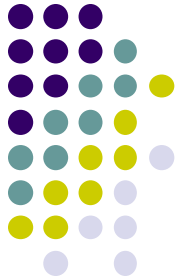
cooperation tests

- *for the pair* $\{true\}\ c?x;\ \{x=3\}$ *and* $\{y=3\}\ c!y;\ \{y=3\}$ *we have to prove* $\{true\wedge y=3\}\ x:=y\ \{x=3\wedge y=3\}$, *and*
- *for the pair* $\{x=4\}\ d!(x+2)\ \{x=4\}$ *and* $\{y=3\}\ d?y;\ \{y=6\}$ *we have to show* $\{x=4\wedge y=3\}\ y:=x+2\ \{x=4\wedge y=6\}$.

parallel composition

$$\{y=3\}\ \ (c?x;\, x:=x+1;\, d!(x+2))\ \|\ (c!y;\, d?y;\, y:=y+2)\ \ \{x=4\wedge y=8\}.$$

Completeness and compositionality



- It comes out that the proof system is not complete.

  Due to possible interleaving of communication actions via channel c it is not possible to prove that

  $$\{true\}\ (c!1;c!2)\|(c?x;c?x)\ \{x = 2\}$$

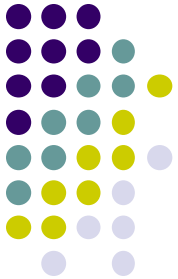
  i.e. it is not possible to find POs of the form

  $$\{p_1\}\ c!1;\ \{r_1\}\ c!2\ \{q_1\} \text{ and } \{p_2\}\ c?x;\ \{r_2\}\ c?x\ \{q_2\}$$

Solution (Levin and Gries)

- using auxiliary variables for indexing channel usage
- taking $<c!e;S>$ as an additional statement, where S can be a an assignment to auxiliary variable, e.g. $k := k+1$
- accepting $\{p\}\ <c!e;S>\ \{q\}$ as an additional PO

Example of using auxiliary variables (for identifying matching pairs)



$\{true\}\ (c!1;c!2)\|(c?x;c?x)\ \{x=2\}$

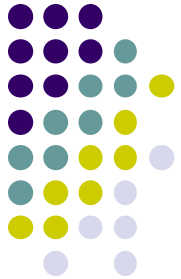
$\{k=0\}\ <c!1;k:=k+1>;\ \{k=1\}\ <c!2;k:=k+1>\ \{k=2\}$

$\{k=0\}\ c?x;\ \{k=1\}\ c?x\ \{k=2\wedge x=2\}$

- *for* $\{k=0\}\ <c!1;k:=k+1>\ \{k=1\}$ *and* $\{k=0\}\ c?x\ \{k=1\}$ *we can prove* $\{k=0\}\ x:=1;k:=k+1\ \{k=1\}$, *and*
- *for* $\{k=1\}\ <c!2;k:=k+1>\ \{k=2\}$ *and* $\{k=1\}\ c?x\ \{k=2\wedge x=2\}$ *we have* $\{k=1\}\ x:=2;k:=k+1\ \{k=2\wedge x=2\}$.

- *for* $\{k=0\}\ <c!1;k:=k+1>\ \{k=1\}$ *and* $\{k=1\}\ c?x\ \{k=2\wedge x=2\}$, *we have* $\{k=0\wedge k=1\}\ x:=1;k:=k+1\ \{k=1\wedge k=2\wedge x=2\}$, *and*
- *for* $\{k=1\}\ <c!2;k:=k+1>\ \{k=2\}$ *and* $\{k=0\}\ c?x\ \{k=1\}$ *we can prove* $\{k=1\wedge k=0\}\ x:=2;k:=k+1\ \{k=2\wedge k=1\}$.

Completeness and compositionality



- The proof system with auxiliary variables is not compositional any more, because the auxiliary variables are shared between the communicating processes
- You have to show that the proof outlines of the local processes are *interference free*.
- There exists a complete and compositional proof method for the parallel programs with message passing using one standard auxiliary variable, recording the history of communication actions.

Assignment

Show that

$$\{true\}\ S_1 \parallel S_2 \parallel S_3\ \{x = u\},$$

where

$S_1 \equiv C!x,$

$S_2 \equiv C?y;\ D!y$

$S_3 \equiv D?u$

Assignment

Let R and T be nonempty sets of natural numbers. Consider the following partitioning algorithm $S_1 \| S_2$, where

$$S_1 \equiv max := max(R); c?mn; d!max;$$
$$\star[max > mn \rightarrow R := (R \setminus \{max\}) \cup \{mn\}; max := max(R);$$
$$c?mn; d!max]$$

$$S_2 \equiv min := min(T); c!min; d?mx;$$
$$\star[mx > min \rightarrow T := (T \setminus \{min\}) \cup \{mx\}; min := min(T);$$
$$c!min; d?mx]$$

Prove, by means of the method of Levin & Gries,

$$\{R = R_0 \neq \emptyset \wedge T = T_0 \neq \emptyset \wedge R \cap T = \emptyset\} \quad S_1 \| S_2$$
$$\{|R| = |R_0| \wedge |T| = |T_0| \wedge R \cup T = R_0 \cup T_0 \wedge max(R) < min(T)\}$$

where, for a set A, $|A|$ denotes the number of elements of A, and R_0 and T_0 are logical variables denoting a finite set of natural numbers.