Theory of concurrency

Lecture 13

Sequential Processes

Sequential Processes: **Assignment**

- Assignments, conditionals, and loops.

<u>Notations</u>

- *Assignment*:
  - $(x := e \,;\, P)$
    - x is a program variable, e is an expression, and P is a process
  - a process which *behaves like P*, except that
    - the initial value of x is the initial value of the expression e.
    - Initial values of all other variables are unchanged.
- Assignment by itself:
  - $(x := e) = (x := e \,;\, SKIP)$

Sequential Processes: Assignment

- Single assignment generalises easily to *multiple assignment*.
    - x stands for a list of distinct variables $x = x_0, x_1, \dots x_{n-1}$
    - e stands for a list of expressions $e = e_0, e_1, \dots e_{n-1}$
  - $x := e$
  - assigns the initial value of e_i to x_i, for all i
    - the lengths of the two lists are the same.

- *All* the e_i are evaluated before *any* of the assignments are made
  - if y occurs in g
    - $y := f\ ;\ z := g$ is quite different from $y, z := f, g$

Sequential Processes: **Assignment**

- $P \not< b \not> Q$ (P **if** b **else** Q)
    - P and Q are processes
    - b is an expression that evaluates to *true* or *false*.
  - a process which behaves like
    - P if the initial value of b is true, or
    - Q if the initial value of b is false.

- The loop:
  - $b * Q$ (**while** b **do** Q)
    - may be defined by recursion

D1. $b * Q = \mu X \bullet ((Q \,;\, X) \not< b \not> SKIP)$

$CT_0 = (up \rightarrow CT_1 \mid around \rightarrow CT_0)$
$CT_{n+1} = (up \rightarrow CT_{n+2} \mid down \rightarrow CT_n)$

Sequential Processes: **Assignment**

X1. A process that behaves like CT_n

$$X1 = \mu X \bullet (around \rightarrow X \mid up \rightarrow (n := 1 \,;\, X))$$
$$\not< n = 0 \not>$$
$$(up \rightarrow (n := n + 1 \,;\, X) \mid down \rightarrow (n := n - 1 \,;\, X))$$

- The current value of the count is recorded in the variable n

X2. A process that behaves like CT_0

$$n := 0 \,;\, X1$$

- The initial value of the count is set to zero.

$POS = (down \rightarrow SKIP \mid up \rightarrow (POS \,;\, POS))$

X3. A process that behaves like POS

$$n := 1 \,;\, (n > 0) * (up \rightarrow n := n + 1 \mid down \rightarrow n := n - 1)$$

- Recursion has been replaced by a conventional loop.

$VAR = left\ ?\ x \rightarrow VAR_x$

$VAR_x = (left\ ?\ y \rightarrow VAR_y \mid right\ !\ x \rightarrow VAR_x)$

$m : VAR\ /\!/\ Q$

Sequential Processes: Assignment

X4. A process divides a natural number x by a positive number y assigning the quotient to q and the remainder to r

$$QUOT = (q := x \div y\ ;\ r := x - q \times y)$$

X5. A process computes the quotient by the slow method of repeated subtraction

$$LONGQUOT = (q := 0\ ;\ r := x\ ;\ ((r \geq y) * (q := q + 1\ ;\ r := r - y)))$$

- Before we model the behaviour of a variable by
  - a subordinate process which communicates its value with the process which uses it.
- Now, we reject that technique, because
  - it does not have the properties which we would like:
    - $(m := 1\ ;\ m := 1) = (m := 1)$
      - but
        - $(m.left\ !\ 1 \rightarrow m.left\ !\ 1 \rightarrow SKIP) \neq (m.left\ !\ 1 \rightarrow SKIP)$

Sequential Processes: Assignment: **Laws**

- <u>Notation</u>
  - x and y stand for lists of distinct variables;
  - e, $f(x)$, $f(e)$ stand for lists of expressions, possible containing variables in x or y;
  - $f(e)$ contains e_i whenever $f(x)$ contains x_i for all indices i.
- We assume that all expressions always give a result, for any values of the variables.

L1. $(x := x) = SKIP$

L2. $(x := e \,;\, x := f(x)) = (x := f(e))$

L3. If x, y is a list of distinct variables $(x := e) = (x, y := e, y)$

L4. If x, y, z are of the same length as e, f, g respectively

$$(x, y, z := e, f, g) = (x, z, y := e, g, f)$$

- Using these laws, it is possible to transform every *sequence of assignments* into
  - *a single assignment* to a list of all the variables involved.

Sequential Processes: Assignment: **Laws**

- As a binary infix operator, $\not< b \not>$ possesses several familiar algebraic properties:

L5–L6. $\not< b \not>$ is idempotent, associative, and distributes through $\not< c \not>$

L5. $P \not< b \not> P = P$

L6. $P \not< b \not> (Q \not< b \not> R) = (P \not< b \not> Q) \not< b \not> R$

L6'. $P \not< b \not> (Q \not< c \not> R) = (P \not< b \not> Q) \not< c \not> (P \not< b \not> Q)$

L7. $P \not< true \not> Q = P$

L8. $P \not< false \not> Q = Q$

L9. $P \not< \neg b \not> Q = Q \not< b \not> P$

L10. $P \not< b \not> (Q \not< b \not> R) = P \not< b \not> R$

L11. $P \not< (a \not< b \not> c) \not> Q = (P \not< a \not> Q) \not< b \not> (P \not< c \not> Q)$

L12. $x := e\ ;\ (P \not< b(x) \not> Q) = (x := e\ ;\ P) \not< b(e) \not> (x := e\ ;\ Q)$

L13. $(P \not< b \not> Q)\ ;\ R = (P\ ;\ R) \not< b \not> (Q\ ;\ R)$

Sequential Processes: Assignment: **Laws**

- We impose a restriction that
  - no variable assigned in one concurrent process can ever be used in another.
    - to deal effectively with assignment in concurrent processes.
- Notation:
  - *var(P)* is the set of variables that may be assigned within P
  - *acc(P)* is the set of variables that may be accessed in expressions within P.
  - *acc(e)* is the set of variables appearing in e.
- All variables which may be changed may also be accessed:
  - $var(P) \subseteq acc(P) \subseteq \alpha P$
- If P and Q are joined by ||:
  - $var(P) \cap acc(Q) = var(Q) \cap acc(P) = \{\}$
- It does not matter whether an assignment takes place before a parallel split, or within one of its components after they are running concurrently:

L14. $((x := e \; ; P) \parallel Q) = (x := e \; ; (P \parallel Q))$ **if** $x \subseteq var(P) - acc(Q)$ **and** $acc(e) \cap var(Q) = \{\}$

L14. $((x := e \,;\, P) \parallel Q) = (x := e \,;\, (P \parallel Q))$ **if** $x \subseteq var(P) - acc(Q)$ **and** $acc(e) \cap var(Q) = \{\}$

Sequential Processes: Assignment: **Laws**

- A consequence:
- $(x := e \,;\, P) \parallel (y := f \,;\, Q) = (x, y := e, f \,;\, (P \parallel Q))$

  if $x \subseteq var(P) - acc(Q) - acc(f)$ **and** $y \subseteq var(Q) - acc(P) - acc(e)$
  - The alphabet restriction ensures that
    - assignments within one component process of a concurrent pair cannot *interfere* with assignments within the other.
      - In an implementation, sequences of assignments may be carried out
        - together or in any interleaving.

- Concurrent combination distributes through the conditional:

L15. $P \parallel (Q \lessdot b \gtrdot R) = (P \parallel Q) \lessdot b \gtrdot (P \parallel R)$ **if** $acc(b) \cap var(P) = \{\}$.

- It does not matter whether *b* is evaluated before or after the parallel split.

Sequential Processes: Assignment: **Laws**

- What if expressions are undefined for certain values of the variables they contain?
- $\mathcal{D}e$ is a Boolean expression which is true iff
  - all the operands of e are within the domains of their operators.
    - e is a list of expressions
- In natural number arithmetic,
  - $\mathcal{D}(x \div y) = (y > 0)$
  - $\mathcal{D}(y + 1, z + y) = true$
  - $\mathcal{D}(e + f) = \mathcal{D}e \wedge \mathcal{D}f$
  - $\mathcal{D}(r - y) = y \leq r$
- $\mathcal{D}e$ is always defined:
  - $\mathcal{D}(\mathcal{D}e) = true$

Sequential Processes: Assignment: **Laws**

- The result of an attempt to evaluate an undefined expression is unspecified:

L16'. $(x := e) = (x := e \not< \mathcal{D}e \not> CHAOS)$

L17'. $P \not< b \not> Q = ((P \not< b \not> Q) \not< \mathcal{D}b \not> CHAOS)$

- The slight modification of the laws L2, L4, and L12:

L2'. $(x := e; x := f(x)) = (x := f(e) \not< \mathcal{D}e \not> CHAOS)$

L5'. $(P \not< b \not> P) = (P \not< \mathcal{D}b \not> CHAOS)$

L12. $x := e\ ; (P \not< b(x) \not> Q) = (x := e\ ; P) \not< b(e) \not> (x := e\ ; Q)$?

L2. $(x := e\ ; x := f(x)) = (x := f(e))$

L4. If x, y, z are of the same length as e, f, g respectively

$$(x, y, z := e, f, g) = (x, z, y := e, g, f)$$

Sequential Processes: Assignment: Specifications

- A specification of a sequential process describes
  - the traces of the events which occur
  - the relationship between these traces
  - the initial and final values of the program variables.
    - The initial value of a program variable x – the variable name x.
    - The final value of a variable x – the superscripted name $x^{\checkmark}$.
      - The value of $x^{\checkmark}$ is not observable until the process is terminated
        - the last event of the trace is ✓.

$(x := e) = (x := e\ ;\ SKIP)$

Sequential Processes: Assignment: **Specifications**

X1. A process performs no action, but adds one to the value of *x*, and terminates successfully with the value of *y* unchanged

$$tr = \langle\rangle \lor (tr = \langle\checkmark\rangle \land x^{\checkmark} = x + 1 \land y^{\checkmark} = y)$$

X2. A process performs an event whose symbol is the initial value of the variable *x*, and then terminates successfully, leaving the final values of *x* and *y* equal to their initial values

$$tr = \langle\rangle \lor tr = \langle x\rangle \lor (tr = \langle x,\checkmark\rangle \land x^{\checkmark} = x \land y^{\checkmark} = y)$$

X3. A process stores the identity of its first event as the final value of *x*

$$\#tr \le 2 \land (\#tr = 2 \Rightarrow (tr = \langle x^{\checkmark},\checkmark\rangle \land y^{\checkmark} = y))$$

Sequential Processes: Assignment: Specifications

X4. A process divides a nonnegative x by a positive y, and assigns the quotient to q and the remainder to r

$$DIV = (y > 0 \Rightarrow tr = <> \vee \ (tr = <\surd> \wedge \\ q^{\surd} = (x \div y) \wedge r^{\surd} = x - (q^{\surd} \times y) \wedge y^{\surd} = y \wedge x^{\surd} = x))$$

- Without the precondition, this specification is impossible to meet in its full generality.

X5. Here are some more complex specifications which will be used later

$$DIVLOOP = (tr = <> \vee (tr = <\surd> \wedge \\ r = (q^{\surd} - q) \times y + r^{\surd} \wedge r^{\surd} < y \wedge x^{\surd} = x \wedge y^{\surd} = y))$$

- $T(n) = r < n \times y$
- All variables in these and subsequent specifications are evaluated by *natural numbers*
    - subtraction is *undefined* if the second operand is greater than the first.

Sequential Processes: Assignment: **Specifications**

- The laws for proving that a process satisfies its specification.
- *s(x, tr, x✓)* is a specification.
- *SKIP* satisfies this specification iff
  - this specification must be true when
    - the trace is empty;
    - the trace is ✓
    - the final values of all variables *x✓* are equal to their initial values.

L1. If $S(x, \langle\rangle, x^{\checkmark})$ **and** $S(x, \langle\checkmark\rangle, x^{\checkmark})$ **then** $SKIP$ sat $S(x, tr, x^{\checkmark})$

X6. The strongest specification satisfied by *SKIP* is

$$SKIP_A \text{ sat } (tr = \langle\rangle \lor (tr = \langle\checkmark\rangle \land \boldsymbol{x}^{\checkmark} = \boldsymbol{x}))$$

- where $\boldsymbol{x}$ is a list of all variables in *A* and $\boldsymbol{x}^{\checkmark}$ is a list of their ticked variants.
- X6 is an immediate consequence of L1 and *vice versa*.

$COND = (q := q + 1 \; ; \; r := r - y \; ; \; X) \not< r \geq y \not> SKIP$

$COND$ sat $(T(n + 1) \Rightarrow DIVLOOP)$

Sequential Processes: Assignment: Specifications

$x = q \times y + r$

X7. We can prove that

- $SKIP$ sat $(r < y \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$
- *Proof* :

(1) Replacing *tr* by <> in the specification gives

$$r < y \wedge T(n + 1) \Rightarrow \langle\rangle = \langle\rangle \vee \ldots$$

- which is a tautology.

(2) Replacing *tr* by <√> and final values by initial values gives

$$r < y \wedge T(n + 1) \Rightarrow (\langle\checkmark\rangle = \langle\rangle \vee (\langle\checkmark\rangle = \langle\checkmark\rangle \wedge x = x \wedge y = y \wedge r = ((q - q) \times y + r \wedge r < y)))$$

- which is also a trivial theorem.

$T(n) = r < n \times y$

$DIVLOOP = (tr = \langle\rangle \vee (tr = \langle\checkmark\rangle \wedge r = (q^{\checkmark} - q) \times y + r^{\checkmark} \wedge r^{\checkmark} < y \wedge x^{\checkmark} = x \wedge y^{\checkmark} = y))$

X6. $SKIP_A$ sat $(tr = <> \vee (tr = <\checkmark> \wedge x^{\checkmark} = x))$

L1. $SKIP \, ; P = P \, ; SKIP = P$

$(x := e) = (x := e \, ; SKIP)$

Sequential Processes: Assignment: Specifications

- A precondition of successful assignment $x := e$ is
  - the expressions e on the right-hand side should be defined.
- If P satisfies a specification $S(x)$,
  - $(x := e \, ; P)$ satisfies the same specification,
    - enriched the fact that the initial value of x is e.

L2. If P sat $S(x)$ **then** $(x := e; P)$ sat $(\mathcal{D}e \Rightarrow S(e))$

- The law for simple assignment follows from L2 on replacing P by $SKIP$, using X6 and L1:

L2A. $x_0 := e$ sat $(\mathcal{D}e \wedge tr \neq <> \Rightarrow tr = <\checkmark> \wedge x_0^{\checkmark} = e \wedge x_1^{\checkmark} = x_1 \wedge \ldots)$

- A consequence
  - for any P, the strongest fact one can prove about $(x := 1/0 \, ; P)$ is
    - $(x := 1/0 \, ; P)$ sat $true$
  - Whatever non-vacuous goal you may wish to achieve,
    - it cannot be achieved by starting with an illegal assignment.

$COND = (q := q + 1 ; r := r - y ; X) \not< r \geq y \not> SKIP$

$COND$ sat $(T(n + 1) \Rightarrow DIVLOOP)$

X6. $SKIP_A$ sat $(tr = \langle\rangle \vee (tr = \langle\surd\rangle \wedge x^{\surd} = x))$

L2A. $x_0 := e$ sat $(\mathcal{D}e \wedge tr \neq \langle\rangle \Rightarrow tr = \langle\surd\rangle \wedge x_0^{\surd} = e \wedge x_1^{\surd} = x_1 \wedge \ldots)$

Sequential Processes: Assignment: Specifications

X8.

$$SKIP \text{ sat } (tr \neq \langle\rangle \Rightarrow tr = \langle\surd\rangle \wedge q^{\surd} = q \wedge r^{\surd} = r \wedge y^{\surd} = y \wedge x^{\surd} = x)$$

- therefore

$$(r := x - q \times y \; ; SKIP) \text{ sat } (x \geq q \times y \wedge tr \neq \langle\rangle \Rightarrow$$
$$tr = \langle\surd\rangle \wedge q^{\surd} = q \wedge r^{\surd} = (x - q \times y) \wedge y^{\surd} = y \wedge x^{\surd} = x)$$

- therefore

$$(q := x \div y \; ; r := x - q \times y) \text{ sat } (y > 0 \wedge x \geq (x \div y) \times y \wedge tr \neq \langle\rangle \Rightarrow$$
$$tr = \langle\surd\rangle \wedge q^{\surd} = (x \div y) \wedge r^{\surd} = (x - (x \div y) \times y) \wedge y^{\surd} = y \wedge x^{\surd} = x)$$

- This specification is equivalent to *DIV*.

$T(n) = r < n \times y$

$DIV = (y > 0 \Rightarrow tr = \langle\rangle \vee \; (tr = \langle\surd\rangle \wedge q^{\surd} = (x \div y) \wedge r^{\surd} = x - (q^{\surd} \times y) \wedge y^{\surd} = y \wedge x^{\surd} = x))$

$COND = (q := q + 1 \; ; \; r := r - y \; ; \; X) \not< r \geq y \not> SKIP$

L2. If P sat $S(x)$ **then** $(x := e; P)$ sat $(\mathcal{D}e \Rightarrow S(e))$

$COND$ sat $(T(n + 1) \Rightarrow DIVLOOP)$

Sequential Processes: Assignment: **Specifications**

X9. Assume X sat $(T(n) \Rightarrow DIVLOOP)$

- therefore

$$(r := r - y \; ; \; X) \text{ sat } (y \leq r \Rightarrow (r - y < n \times y \Rightarrow (tr = \langle\rangle \vee tr = \langle\surd\rangle \wedge (r - y) = \ldots)))$$

- therefore $(q := q + 1 \; ; \; r := r - y \; ; \; X)$ sat $(y \leq r \Rightarrow (r < (n + 1) \times y \Rightarrow DIVLOOP'))$
- where $DIVLOOP' = (tr = \langle\rangle \vee (tr = \langle\surd\rangle \wedge (r - y) = (q^{\surd} - (q + 1)) \times y + r^{\surd} \wedge r^{\surd} < y \wedge x^{\surd} = x \wedge y^{\surd} = y))$
- By elementary algebra of natural numbers $y \leq r \Rightarrow (DIVLOOP' \equiv DIVLOOP)$
- therefore $(q := q + 1 \; ; \; r := r - y \; ; \; X)$ sat $(y \leq r \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$
- This result will be used in X10.

$T(n) = r < n \times y$

$DIVLOOP = (tr = \langle\rangle \vee (tr = \langle\surd\rangle \wedge r = (q^{\surd} - q) \times y + r^{\surd} \wedge r^{\surd} < y \wedge x^{\surd} = x \wedge y^{\surd} = y))$

Sequential Processes: Assignment: **Specifications**

- In general sequential composition
  - the traces of the components are sequentially composed
  - the initial state of the second component is identical to the final state of the first one.
  - the values of the variables in this intermediate state are not observable.

L3. If P sat $S(x, tr, x^{\checkmark})$ **and** Q sat $T(x, tr, x^{\checkmark})$ **and** P does not diverge **then**

$$(P \,;\, Q) \text{ sat } (\exists\, y, s, t \bullet tr = (s \,;\, t) \wedge S(x, s, y) \wedge T(y, t, x^{\checkmark}))$$

  - x is a list of all variables in the alphabet of P and Q,
  - $x^{\checkmark}$ is a list of their subscripted variants,
  - y a list of the same number of fresh variables.

L4A'. If P sat $(b \Rightarrow S)$ **and** Q sat $(\neg b \Rightarrow S)$ **then** $(P \not< b \not> Q)$ sat S

- The specification of a conditional:

L4. If P sat S **and** Q sat T **then** $(P \not< b \not> Q)$ sat $((b \wedge S) \vee (\neg b \wedge T))$

- An alternative form of this law:

L4A. If P sat $(b \Rightarrow S)$ **and** Q sat $(\neg b \Rightarrow T)$ **then** $(P \not< b \not> Q)$ sat $(S \vee T)$

$$DIVLOOP = (tr = \langle\rangle \lor (tr = \langle\surd\rangle \land r = (q^{\surd} - q) \times y + r^{\surd} \land r^{\surd} < y \land x^{\surd} = x \land y^{\surd} = y))$$

$$T(n) = r < n \times y$$

Sequential Processes: Assignment: **Specifications**

X10. Let $COND = (q := q + 1\ ;\ r := r - y\ ;\ X) \not< r \geq y \not> SKIP$

and

X sat $(T(n) \Rightarrow DIVLOOP)$

then

$COND$ sat $(T(n + 1) \Rightarrow DIVLOOP)$

- The two sufficient conditions for this conclusion have been proved in X7 and X9; the result follows by L4A.

X7. $SKIP$ sat $(r < y \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$

X9. $(q := q + 1\ ;\ r := r - y\ ;\ X)$ sat $(y \leq r \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$

L4A'. If P sat $(b \Rightarrow S)$ **and** Q sat $(\neg b \Rightarrow S)$ **then** $(P \not< b \not> Q)$ sat S

Sequential Processes: Assignment: **Specifications**

- Proofs of a loop use the recursive definition D1, and the law for unguarded recursion L8.
- If R is the intended specification of the loop,
  - we must find a specification $S(n)$ such that
    - $S(0)$ is always true, and
    - $(\forall n \bullet S(n)) \Rightarrow R$
- A general method to construct $S(n)$ is
  - to find a predicate $T(n, x)$, which describes
    - the conditions on the initial state x such that
      - the loop is certain to terminate in less than n repetitions.
  - Then define $S(n) = (T(n, x) \Rightarrow R)$
- If $T(n, x)$ is correctly defined then
  - $T(0, x)$ will be false, and consequently $S(0)$ will be true.
  - No loop can terminate in less than no repetitions.
- The result of the proof of the loop will be $\forall n \bullet S(n)$, i.e., $\forall n \bullet (T(n, x) \Rightarrow R)$

Sequential Processes: Assignment: Specifications

- The result of the proof of the loop will be $\forall n \bullet S(n)$, i.e. $\forall n \bullet (T(n, x) \Rightarrow R)$
- n is a variable which does not occur in R, hence
- $\forall n \bullet (T(n, x) \Rightarrow R) \equiv (\exists n \bullet T(n, x)) \Rightarrow R$
- No stronger specification can be met, since
  - $\exists n \bullet T(n, x)$ is
    - the precondition "the loop terminates in some finite number of iterations".
- Finally, we must prove that the body of the loop meets its specification.
- Since the recursive equation for a loop involves a conditional, this task splits into two.
- This reasoning is formalized by the general law

L5. If $\neg T(0, x)$ **and** $T(n, x) \Rightarrow \mathcal{D}b$ **and** $SKIP$ sat $(\neg b \Rightarrow (T(n, x) \Rightarrow R))$

and $(X$ sat $T(n, x) \Rightarrow R)) \Rightarrow ((Q \,;\, X)$ sat $(b \Rightarrow (T(n + 1, x) \Rightarrow R)))$

then $(b * Q)$ sat $((\exists n \bullet T(n, x)) \Rightarrow R)$

L5. If $\neg T(0, x)$ **and** $T(n, x) \Rightarrow \mathcal{D}b$ **and** $SKIP$ sat $(\neg b \Rightarrow (T(n, x) \Rightarrow R))$
and $(X$ sat $T(n, x) \Rightarrow R)) \Rightarrow ((Q ; X)$ sat $(b \Rightarrow (T(n + 1, x) \Rightarrow R)))$
then $(b * Q)$ sat $((\exists\, n \bullet T(n, x)) \Rightarrow R)$

Sequential Processes: Assignment: Specifications

$$DIV = (y > 0 \Rightarrow tr = \langle\rangle \vee (tr = \langle\checkmark\rangle \wedge q^{\checkmark} = (x \div y) \wedge r^{\checkmark} = x - (q^{\checkmark} \times y) \wedge y^{\checkmark} = y \wedge x^{\checkmark} = x))$$

X11. Prove that the program for long division by repeated subtraction meets

- its specification *DIV*.

X5. $LONGQUOT = (q := 0 ; r := x ; ((r \geq y) * (q := q + 1 ; r := r - y)))$

- The task splits naturally in two.
- The second part is to prove that the loop meets specification

$$(r \geq y) * (q := q + 1 ; r := r - y) \text{ sat } (y > 0 \Rightarrow DIVLOOP)$$

- The condition under which the loop terminates in less than n iterations:
  - $T(n) = r < n \times y$
  - $T(0)$ is false;
  - $\exists\, n \bullet T(n)$ is equivalent to $y > 0$ (the precondition for the loop termination).
- The remaining steps are taken in X7 and X9.

X5. $DIVLOOP = (tr = \langle\rangle \vee (tr = \langle\checkmark\rangle \wedge r = (q^{\checkmark} - q) \times y + r^{\checkmark} \wedge r^{\checkmark} < y \wedge x^{\checkmark} = x \wedge y^{\checkmark} = y))$

X7. $SKIP$ sat $(r < y \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$

X9. $(q := q + 1 ; r := r - y ; X)$ sat $(y \leq r \Rightarrow (T(n + 1) \Rightarrow DIVLOOP))$

$T(n) = r < n \times y$

Sequential Processes: Assignment: **Specifications**

- The laws for sequential programs are used to prove *total* correctness for
  - programs, which contain no input or output.
- Q sat $(S(x) \wedge tr \neq \langle\rangle \Rightarrow tr = \langle\checkmark\rangle \wedge R(x, x^{\checkmark}))$ (1)
- For sequential program Q, a proof of this specification established that
  - if $S(x)$ is true of the initial values of the variables when Q is started, then
    - Q will terminate and
      - $R(x, x^{\checkmark})$ describes the relation between initial values x and final values $x^{\checkmark}$.
- $(S(x), R(x, x^{\checkmark}))$ form a *precondition / postcondition pair*.

- In the special case of noncommunicating programs, the proof methods are mathematically equivalent to ones that are already familiar.

Shared Resources

Shared Resources: **Introduction**

- The sole task of subordinate process $(m : R)$ is
  - to meet the needs of a single main process S:
    - $(m : R \mathbin{/\!/} S)$
- Let S consist of two concurrent processes $(P \parallel Q)$, and
  - *both* P and Q require the services of the same subordinate process $(m : R)$.
- It is not possible for P and Q both to communicate with $(m : R)$ along the same channels:
  - these channels would have to be in the alphabet of both P and Q;
  - the definition of $\parallel$ requires that
    - communications with $(m : R)$ take place only when
      - both P and Q communicate the same message simultaneously.
- We need *interleaving the communications*
  - between P and $(m : R)$ with those between Q and $(m : R)$.
- $(m : R)$ serves as *a resource shared* between P and Q;
  - each of them uses it independently, their interactions with it are interleaved.

Shared Resources: **Introduction**

- Each sharing process uses a *different* set of channels to the shared resource.
  - if the *identity* of all the sharing processes is known in advance
    - The dining philosophers: shared forks and the footman.
    - X6: a shared buffer, Q uses only the left channel, P uses only the right channel.
- Multiple labelling for a general method of sharing
  - creates enough separate channels for independent communication with processes.
  - individual communications along these channels are arbitrarily interleaved.
  - requires that the names of all the sharing processes are known *in advance*
    - *not* for a subordinate process serving a main process with
      - an *arbitrary number* of concurrent subprocesses.
- We need techniques for sharing a resource among many processes
  - even when their number and identities are not known in advance.
    - operating systems,
    - a cloud.

Shared Resources: **Sharing by interleaving**

- The interleaving form of concurrency $(P \parallel\!\mid Q)$
  - P and Q have the same alphabet
  - their communications with external (shared) processes are arbitrarily interleaved.
  - no *direct* communication between P and Q
  - *indirect* communication through the services of a shared subordinate process.

Shared Resources: **Sharing by interleaving**

X1. (Shared subroutine) $doub : DOUBLE \mathbin{/\!/} (P \mathbin{|||} Q)$

- Both P and Q contain calls on the subordinate process

$$(doub.left\ !\ v \rightarrow doub.right\ ?\ x \rightarrow SKIP)$$

- Correctness: no situation that
  - some processes accidentally obtains an answer intended for the other.
- All subprocesses of the main process must strictly *alternate* communications
  - on the left channel and on the right channel of the shared subordinate.
- Introduce a specialised notation
  - ≈ a traditional procedure call in a high-level language

$$doub\ !\ x\ ?\ y = (doub.left\ !\ x \rightarrow doub.right\ ?\ y \rightarrow SKIP)$$

L7. If $P = (x : A \to P(x))$ **and** $Q = (y : B \to P(y))$
then $P \mid\mid\mid Q = (x : A \to (P(x) \mid\mid\mid Q) \;\square\; y : B \to (P \mid\mid\mid Q(y)))$

Shared Resources: **Sharing by interleaving**

- Two sharing processes simultaneously use the shared subroutine
  - matched pairs of communications are taken in arbitrary order
    - *the components of a pair of communications* with one process are
      - *never separated* by a communication with another.

- The abbreviations

$$\left.\begin{array}{l} d\,!\,v \text{ for } d.left\,!\,v \\ d\,?\,x \text{ for } d.right\,?\,x \end{array}\right\} \quad \text{within a main process}$$

$$\left.\begin{array}{l} !\,v \text{ for } right\,!\,v \\ ?\,x \text{ for } left\,?\,x \end{array}\right\} \quad \text{within a subordinate process}$$

- Let

$$\begin{array}{ll} D = ?\,x \to !(x + x) \to D & P = d\,!\,3 \to d\,?\,y \to P(y) \\ R = (d : D \,/\!/\, (P \mid\mid\mid Q)) & Q = d\,!\,4 \to d\,?\,z \to Q(z) \end{array}$$

- then

$$P \mid\mid\mid Q = d\,!\,3 \to ((d\,?\,y \to P(y)) \mid\mid\mid Q) \;\square\; d\,!\,4 \to (P \mid\mid\mid (d\,?\,z \to Q(z))) \quad \text{[by } \mathbf{L7}\text{]}$$

$$P \parallel\!\mid Q = d\,!\,3 \rightarrow ((d\,?\,y \rightarrow P(y)) \parallel\!\mid Q) \,\square\, d\,!\,4 \rightarrow (P \parallel\!\mid (d\,?\,z \rightarrow Q(z)))$$

$$D = ?\,x \rightarrow !(x + x) \rightarrow D$$

Shared Resources: **Sharing by interleaving**

- The shared process accepts either input
  - after hiding the choice becomes nondeterministic:

$$\begin{aligned}(d : D \,/\!/\, (P \parallel\!\mid Q)) &= ((d : (!3 + 3 \rightarrow D)) \,/\!/\, ((d\,?\,y \rightarrow P(y)) \parallel\!\mid Q)) \sqcap \\ &\quad ((d : (!4 + 4 \rightarrow D)) \,/\!/\, (P \parallel\!\mid (d\,?\,z \rightarrow Q(z)))) \\ &= (d : D \,/\!/\, (P(6) \parallel\!\mid Q)) \sqcap (d : D \,/\!/\, (P \parallel\!\mid Q(8)))\end{aligned}$$

- The shared process offers its result to
  - *whichever* of the sharing processes is ready to take it.
    - The process provided the argument gets the result
      - The other process is still waiting for output.
- Strict alternation of output and input is important in calling a shared subroutine.

Shared Resources: **Sharing by interleaving**

X8. $SET = left\ ?\ x \rightarrow right\ !\ NO \rightarrow (rest : SET \mathbin{/\!/} LOOP(x))$
$LOOP(x) = \mu X \bullet left\ ?\ y \rightarrow (\textbf{if}\ y = x\ \textbf{then}\ right\ !\ YES \rightarrow X$
$\textbf{else}\ (rest.left\ !\ y \rightarrow rest.right\ ?\ z \rightarrow right\ !\ z \rightarrow X))$

X2. (Shared data structure)

- In an airline flight reservation system, bookings are made by
  - many reservation clerks, whose actions are interleaved.
- Each reservation adds a passenger to the flight list, and
  - returns an indication whether that passenger was already booked or not.
- The set X8 serves as a shared subordinate process, named by the flight number:

$$SU2584 : SET \mathbin{/\!/} (\ldots (CLERK \mathbin{|||} CLERK \mathbin{|||} \ldots)\ldots)$$

- Each *CLERK* books a passenger by the call $SU2584\ !\ pass\_n\ ?\ x$
- which stands for $(SU2584.left\ !\ pass\_n \rightarrow SU2584.right\ ?\ x \rightarrow SKIP)$

Shared Resources: **Sharing by interleaving**

- In X1 and X2, each occasion of use of the shared resource involves
  - *exactly two* communications,
    - one to send the parameters and the other to receive the results;
  - after each pair of communications, the subordinate process returns to a state in which
    - it is ready to serve another process, or the same one again.

- *Series of communications* without interference by another processes.
  - A single *output device* is shared by several concurrent processes.
    - On each occasion of use, a *number of lines* must be output consecutively
      - without any danger of interleaving of lines sent by another process.
  - The output of a file must be preceded by an *acquire*
    - which obtains exclusive use of the resource;
      - and on completion, the resource must be made available again by a *release*.

Shared Resources: **Sharing by interleaving**

X3. (Shared line printer)

$$LP = acquire \rightarrow \mu X \bullet (left\ ?\ s \rightarrow h\ !\ s \rightarrow X \mid release \rightarrow LP)$$

- h is the channel which connects LP to the hardware of the line printer.
- After acquisition,
  - the process LP copies successive lines from its left channel to its hardware,
    - until a release signal returns it to its original state, in which
      - it is available for use by any other processes.
- This process is used as a shared resource

$$lp.acquire \rightarrow \ldots\ lp.left\ !\ \text{“A. JONES”} \rightarrow \ldots\ lp.left\ !\ nextline \rightarrow \ldots\ lp.release \rightarrow$$

- The use of the signals *acquire* and *release* prevent
  - arbitrary interleaving of lines from distinct files without the danger of deadlock.

$LP = acquire \to \mu X \bullet (left\ ?\ s \to h\ !\ s \to X \mid release \to LP)$

$lp.acquire \to \ldots\ lp.left\ !$ "A. JONES" $\to \ldots\ lp.left\ !\ nextline \to \ldots\ lp.release \to$

Shared Resources: **Sharing by interleaving**

- If more than one resource is to be shared in *acquire* | *release* fashion,
  - the risk of deadlock cannot be ignored.

X5. (Deadlock, by E. W. Dijkstra) Ann and Mary are good friends and good cooks;

- they share a pot and a pan, which they *acquire*, use and *release* as they need them

$$UTENSIL = (acquire \to use \to use \to \ldots \to release \to UTENSIL)$$
$$pot : UTENSIL \mathbin{/\!/} pan : UTENSIL \mathbin{/\!/} (ANN \mathbin{|||} MARY)$$

- Ann cooks in accordance with a recipe which requires a pot first and then a pan,
- Mary needs a pan first, then a pot

$$ANN = \ldots\ pot.acquire \to \ldots\ pan.acquire \to \ldots$$
$$MARY = \ldots\ pan.acquire \to \ldots\ pot.acquire \to \ldots$$

- They decide to prepare a meal at about the same time.
  - Each of them acquires her first utensil;
  - When she needs her second utensil,
    - she finds that she cannot have it, because it is being used by the other.

Shared Resources: **Sharing by interleaving**

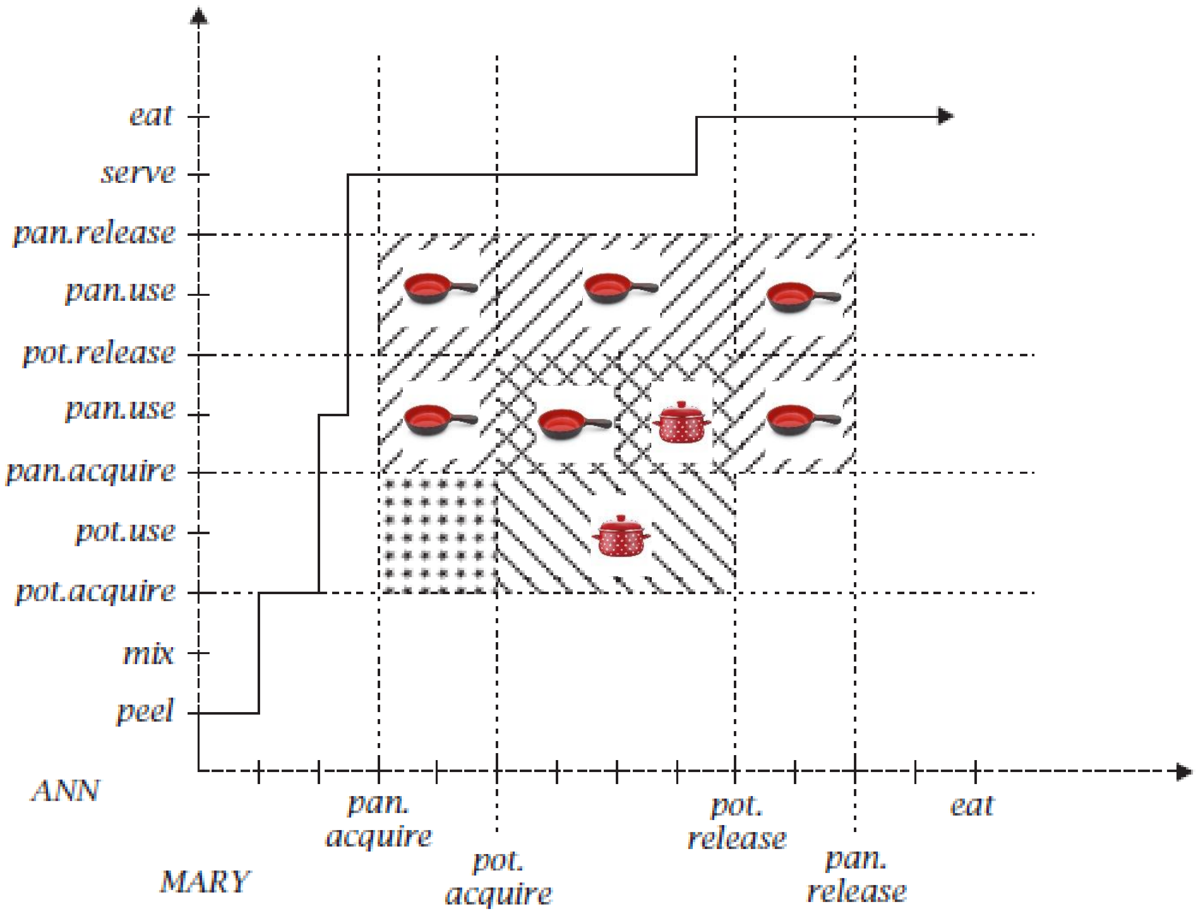
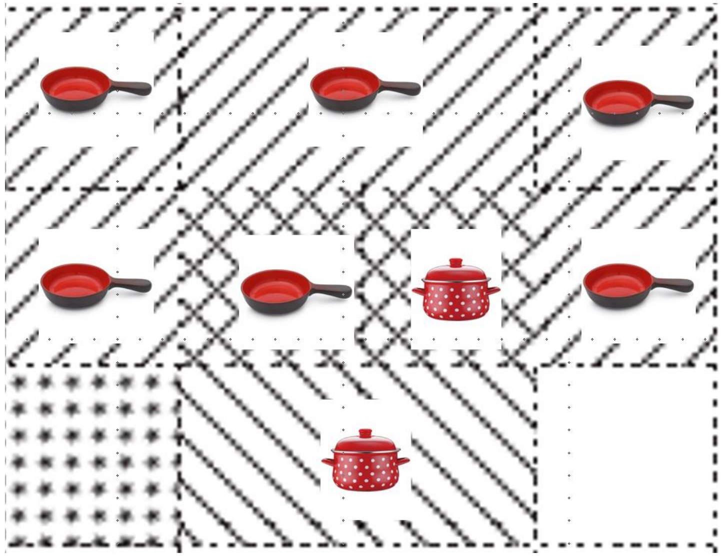


Figure 6.1

Shared Resources: **Sharing by interleaving**



- Consider the zone marked with dots.
- If ever the trajectory enters this zone,
  - it will inevitably end in deadlock.
- It is reasonable to extend the forbidden region
  - *to cover* the danger zone.
- To introduce an *additional artificial resource* which
  - must be acquired before either utensil, and
  - must not be released until both utensils have been released.
    - The footman in the story of the dining philosophers
      - permission to sit down is a kind of resource
        - only four instances of the permission are shared by five philosophers.
- An easier solution is to insist that
  - any cook who is going to want both utensils must acquire the pan first. 

Shared Resources: **Sharing by interleaving**

- -solution generalises to
  - any number of users, and any number of resources.
- A fixed order for acquiring the resources provides absence of deadlock.

- Users release the resources as soon as they have finished with them;
  - the order of release does not matter.
    - Users may even acquire resources out of order, if
      - at the time of acquisition they have already released all resources
        - which are *later* in the standard ordering.
- Such fixed order can often be checked by
  - a visual scan of the text of the user processes.