Interaction

**shared variables**
- can be read and written by any process (most interaction)
- difficult to implement
- difficult to reason about

**interactive variables**
- can be read by any process, written by only one process (some interaction)
- easier to implement
- easier to reason about

**boundary variables**
- can be read and written by only one process (least interaction)
  - but initial value can be seen by all processes
- easiest to implement
- easiest to reason about
**Interactive Variables**

boundary variable \( \text{var } a: T \cdot S = \exists a, a': T \cdot S \)

interactive variable \( \text{ivar } x: T \cdot S = \exists x: time \rightarrow T \cdot S \)

The value of variable \( x \) at time \( t \) is \( x_t \)

But sometimes we write \( x \) for \( x_t \), \( x' \) for \( x_{t'} \), \( x'' \) for \( x_{t''} \), ...

\[ a := a + x \]

is really

\[ a := a + x_t \]

Most laws still work but not the Substitution Law
Interactive Variables

suppose boundary $a$, $b$; interactive $x$, $y$; time $t$

$$ok = a' = a \land b' = b \land t' = t$$

$$x' = x \land y' = y$$ means $$x' t' = x t \land y' t' = y t$$
Interactive Variables

suppose boundary $a, b$; interactive $x, y$; time $t$

\[
ok = a' = a \land b' = b \land t' = t
\]
\[
a := e = a' = e \land b' = b \land t' = t
\]
\[
x := e = a' = a \land b' = b \land x' = e \land (\forall t'' : t \leq t'' \leq t' \Rightarrow y'' = y)
\]
\[
\land t' = t + \text{(the time required to evaluate and store } e \text{)}
\]
\[
P \cdot Q = \exists a'', b'', t'' : \text{(substitute } a'', b'', t'' \text{ for } a', b', t' \text{ in } P)
\]
\[
\land (\text{substitute } a'', b'', t'' \text{ for } a, b, t \text{ in } Q)
\]
\[
P \parallel Q = \exists t_P, t_Q : \text{(substitute } t_P \text{ for } t' \text{ in } P)
\]
\[
\land (\text{substitute } t_Q \text{ for } t' \text{ in } Q)
\]
\[
\land t' = \max t_P t_Q
\]
\[
\land (\forall t'' : t_P \leq t'' \leq t' \Rightarrow x'' = x(t_P)) \quad \text{interactive variables of } P
\]
\[
\land (\forall t'' : t_Q \leq t'' \leq t' \Rightarrow y'' = y(t_Q)) \quad \text{interactive variables of } Q
\]
Interactive Variables

**example** boundary $a$, $b$; interactive $x$, $y$; extended integer time $t$

\[
(x := 2. x := x + y. x := x + y) \parallel (y := 3. y := x + y) \quad x \text{ left, } y \text{ right, } a \text{ left, } b \text{ right}
\]

\[
= (a' = a \land xt' = 2 \land t' = t + 1. \quad a' = a \land xt' = xt + yt \land t' = t + 1. \quad a' = a \land xt' = xt + yt \land t' = t + 1)
\]

\[
\parallel (b' = b \land yt' = 3 \land t' = t + 1. \quad b' = b \land yt' = xt + yt \land t' = t + 1)
\]

\[
= (a' = a \land x(t + 1) = 2 \land x(t + 2) = x(t + 1) + y(t + 1) \land x(t + 3) = x(t + 2) + y(t + 2) \land t' = t + 3)
\]

\[
\parallel (b' = b \land y(t + 1) = 3 \land y(t + 2) = x(t + 1) + y(t + 1) \land t' = t + 2)
\]

\[
= x(t + 1) = 2 \land x(t + 2) = x(t + 1) + y(t + 1) \land x(t + 3) = x(t + 2) + y(t + 2)
\]

\[
\land y(t + 1) = 3 \land y(t + 2) = x(t + 1) + y(t + 1) \land y(t + 3) = y(t + 2)
\]

\[
\land a' = a \land b' = b \land t' = t + 3
\]

\[
= x(t + 1) = 2 \land x(t + 2) = 5 \land x(t + 3) = 10 \land y(t + 1) = 3 \land y(t + 2) = y(t + 3) = 5 \land a' = a \land b' = b \land t' = t + 3
\]
Thermostat

thermometer \parallel control \parallel thermostat \parallel burner

inputs to the thermostat:

• real temperature, which comes from the thermometer and indicates the actual temperature.
• real desired, which comes from the control and indicates the desired temperature.
• binary flame, which comes from a flame sensor in the burner and indicates whether there is a flame.

outputs of the thermostat:

• binary gas; assigning it \( \top \) turns the gas on and \( \bot \) turns the gas off.
• binary spark; assigning it \( \top \) causes sparks for the purpose of igniting the gas.
Heat is wanted when the actual temperature falls $\varepsilon$ below the desired temperature, and not wanted when the actual temperature rises $\varepsilon$ above the desired temperature, where $\varepsilon$ is small enough to be unnoticeable, but large enough to prevent rapid oscillation. To obtain heat, the spark should be applied to the gas for at least 1 second to give it a chance to ignite and to allow the flame to become stable. But a safety regulation states that the gas must not remain on and unlit for more than 3 seconds. Another regulation says that when the gas is shut off, it must not be turned on again for at least 20 seconds to allow any accumulated gas to clear. And finally, the gas burner must respond to its inputs within 1 second.

\[ \text{thermostat} = (\text{gas}:= \bot \parallel \text{spark}:= \bot). \ \text{GasOff} \]

\[ \text{GasOff} = \begin{cases} \text{if temperature} < \text{desired} - \varepsilon \\
& \text{then } (\text{gas}:= T \parallel \text{spark}:= T \parallel t' \geq t+1) \land t' \leq t+3. \ \text{spark}:= \bot. \ \text{GasOn} \\
& \text{else } ((\text{frame gas, spark} \cdot \text{ok}) \parallel t' \geq t) \land t' \leq t+1. \ \text{GasOff fi} \end{cases} \]

\[ \text{GasOn} = \begin{cases} \text{if temperature} < \text{desired} + \varepsilon \land \text{flame} \\
& \text{then } ((\text{frame gas, spark} \cdot \text{ok}) \parallel t' \geq t) \land t' \leq t+1. \ \text{GasOn} \\
& \text{else } (\text{gas}:= \bot \parallel (\text{frame spark} \cdot \text{ok}) \parallel t' \geq t+20) \land t' \leq t+21. \ \text{GasOff fi} \end{cases} \]
Communication Channels

Channel $c$ is described by

- message script $M_c$ string constant
- time script $T_c$ string constant
- read cursor $r_c$ extended natural variable
- write cursor $w_c$ extended natural variable

$$M = 6 ; 4 ; 7 ; 1 ; 0 ; 3 ; 8 ; 9 ; 2 ; 5 ; ...$$

$$T = 3 ; 5 ; 5 ; 20 ; 25 ; 28 ; 31 ; 31 ; 45 ; 48 ; ...$$

↑  ↑

$r$  $w$
\[ c! \ e = M_w = e \land T_w = t \land (w:=w+1) \]
\[ c! = T_w = t \land (w:=w+1) \]
\[ c? = r := r+1 \]
\[ c = M_{r-1} \]
\[ \sqrt{c} = T_r \leq t \]

\[ M = 6 ; 4 ; 7 ; 1 ; 0 ; 3 ; 8 ; 9 ; 2 ; 5 ; ... \]
\[ T = 3 ; 5 ; 5 ; 20 ; 25 ; 28 ; 31 ; 31 ; 45 ; 48 ; ... \]
\[ \uparrow \quad \uparrow \]
\[ r \quad w \]
Input and Output

\[ c! e = M_w = e \land T_w = t \land (w := w+1) \]

\[ c! = T_w = t \land (w := w+1) \]

\[ c? = r := r+1 \]

\[ c = M_{r-1} \]

\[ \sqrt{c} = T_r \leq t \]

**if** \( \sqrt{key} \)

**then** \( key?. \)

**if** \( key=“y” \)

**then** screen! “If you wish.”

**else** screen! “Not if you don't want.”  **fi**

**else** screen! “Well?”  **fi**
Input and Output

Repeatedly input numbers from channel $c$, and output their doubles on channel $d$.

$$S = \forall n: \text{nat} \cdot M_d w_d + n = 2 \times M_c r_c + n$$

$$S \Leftarrow c? . d! 2 \times c . S$$

**proof**

$$c? . d! 2 \times c . S$$

$$= rc := rc + 1 . M_d w_d = 2 \times M_c r_c - 1 \land (wd := wd + 1) \cdot S$$

$$= M_d w_d = 2 \times M_c r_c \land \forall n: \text{nat} \cdot M_d w_d + 1 + n = 2 \times M_c r_c + 1 + n$$

$$= \forall n: \text{nat} \cdot M_d w_d + n = 2 \times M_c r_c + n$$

$$= S$$
Communication Timing

**real time** need to know implementation

**transit time** input and output take time 0
communication transit takes time 1

\[
\text{input } c? \quad \text{becomes} \quad t := \max t (\mathcal{T}_r c + 1). \quad c?
\]

\[
\text{check } \sqrt{c} \quad \text{becomes} \quad \mathcal{T}_r c + 1 \leq t
\]
Communication Timing

\[ W = t := max \ t (T_r + 1). \ c? \]
\[ = \text{wait (if necessary) for input and then read it} \]

\[ W \leftarrow \text{if } \sqrt{c} \text{ then } c? \text{ else } t := t+1. \ W \text{ fi} \]

**proof**

\[ \text{if } \sqrt{c} \text{ then } c? \text{ else } t := t+1. \ W \text{ fi} \]
\[ = \text{if } T_r + 1 \leq t \text{ then } c? \text{ else } t := t+1. \ t := max \ t (T_r + 1). \ c? \text{ fi} \]
\[ = \text{if } T_r + 1 \leq t \text{ then } t := t. \ c? \text{ else } t := max (t+1) (T_r + 1). \ c? \text{ fi} \]
\[ = \text{if } T_r + 1 \leq t \text{ then } t := max \ t (T_r + 1). \ c? \text{ else } t := max \ t (T_r + 1). \ c? \text{ fi} \]
\[ = W \]
Recursive Communication

dbl = c?. d! 2×c. t:= t+1. dbl

weakest solution

∀n: nat· Md_{wd+n} = 2 × Mc_{rc+n} ∧ Td_{wd+n} = t+n

strongest implementable solution

(∀n: nat· Md_{wd+n} = 2 × Mc_{rc+n} ∧ Td_{wd+n} = t+n)
∧ rc'=wd'=t'=∞ ∧ wc'=wc ∧ rd'=rd

strongest solution

⊥

∀n: nat· Md_{wd+n} = 2 × Mc_{rc+n} ∧ Td_{wd+n} = t+n ⇐ dbl

dbl ⇐ c?. d! 2×c. t:= t+1. dbl
Recursive Construction

\[dbl_0 = T\]

\[dbl_1 = c?.\ d! 2\times c.\ t:= t+1.\ dbl_0\]
\[= rc:= rc+1.\ Mdw = 2 \times Mc_{rc-1} \land Td_{wd} = t \land (wd:= wd+1).\ t:= t+1.\ T\]
\[= Md_{wd} = 2 \times Mc_{rc} \land Td_{wd} = t\]

\[dbl_2 = c?.\ d! 2\times c.\ t:= t+1.\ dbl_1\]
\[= rc:= rc+1.\ Mdw = 2 \times Mc_{rc-1} \land Td_{wd} = t \land (wd:= wd+1).\ t:= t+1.\]
\[Mdw = 2 \times Mc_{rc} \land Td_{wd} = t\]
\[= Md_{wd} = 2 \times Mc_{rc} \land Td_{wd} = t \land Md_{wd+1} = 2 \times Mc_{rc+1} \land Td_{wd+1} = t+1\]

\[dbl_\infty = \forall n: nat.\ Md_{wd+n} = 2 \times Mc_{rc+n} \land Td_{wd+n} = t+n\]
monitor = 

(\sqrt{x0in} \lor Tx0in_{rx0in} = m) \land (x0in?. x := x0in. x0ack!)
\lor (\sqrt{x1in} \lor Tx1in_{rx1in} = m) \land (x1in?. x := x1in. x1ack!)
\lor (\sqrt{x0req} \lor Tx0req_{rx0req} = m) \land (x0req?. x0out! x)
\lor (\sqrt{x1req} \lor Tx1req_{rx1req} = m) \land (x1req?. x1out! x).

monitor
Monitor

\[
\text{monitor } \iff \begin{array}{l}
\text{if } \sqrt{x0in} \text{ then } x0in? \cdot x := x0in \cdot x0ack! \text{ else ok fi.} \\
\text{if } \sqrt{x1in} \text{ then } x1in? \cdot x := x1in \cdot x1ack! \text{ else ok fi.} \\
\text{if } \sqrt{x0req} \text{ then } x0req? \cdot x0out! x \text{ else ok fi.} \\
\text{if } \sqrt{x1req} \text{ then } x1req? \cdot x1out! x \text{ else ok fi.} \\
\end{array}
\]

\[t := t + 1. \text{ monitor}\]
Communicating Processes

\[ c! 2 \parallel (c?. x := c) \]

\[ = \quad M_w = 2 \land (w := w+1) \parallel (r := r+1. x := M_{r-1}) \]

\[ = \quad M_w = 2 \land w' = w+1 \land r' = r+1 \land x' = M_r \]

\[ c! 1. (c! 2 \parallel (c?. x := c)). c? \]

**channel declaration**

\[ \text{chan } c: T \cdot P \]

\[ = \quad \exists M_c: \infty T . \exists T_c: \infty xnat . \text{var } rc , wc: xnat := 0 \cdot P \]
ignoring time

\[\text{chan } c: \text{int} \!\cdot\! c! 2 \ || \ (c?\cdot\ x:= c)\]

\[= \ \exists M: \infty\text{*int}\ \exists T: \infty\text{*xnat} \ \text{var } r, w: \text{xnat} := 0\cdot\]
\[\ x' = M_r \land M_w = 2 \land r' = r+1 \land w' = w+1 \land \text{(other variables unchanged)}\]

\[= \ \exists M: \infty\text{*int}\ \exists T: \infty\text{*xnat} \ \text{var } r, w: \text{xnat} \]
\[\ x' = M_0 \land M_0 = 2 \land r' = 1 \land w' = 1 \land \text{(other variables unchanged)}\]

\[= \ x' = 2 \land \text{(other variables unchanged)}\]

\[= \ x := 2\]

including time

\[\text{chan } c: \text{int} \!\cdot\! c! 2 \ || \ (t:= \max t (T_r + 1). \ c?\cdot\ x:= c)\]

\[= \ x' = 2 \land t' = t+1 \land \text{(other variables unchanged)}\]
Deadlock

\textbf{chan} c:: int. \quad t:= max \ t (\mathcal{T}_r + 1). \quad c?:. \quad c! 5

\begin{align*}
&= \exists M. \ \forall^* \text{int}. \ \exists T. \ \forall^* \text{xnat}. \ \text{var} \ r, w: \text{xnat} := 0. \\
&\quad t:= max \ t (\mathcal{T}_r + 1). \quad r:= r+1. \quad \mathcal{M}_w = 5 \ \land \ T_w = t \ \land \ (w:= w+1)
\end{align*}

\begin{align*}
&= \exists M. \ \forall^* \text{int}. \ \exists T. \ \forall^* \text{xnat}. \ \exists r, r', w, w': \text{xnat}. \\
&\quad r:= 0. \quad w:= 0. \quad t:= max \ t (\mathcal{T}_r + 1). \quad r:= r+1. \\
&\quad \mathcal{M}_w = 5 \ \land \ T_w = t \ \land \ r' = r \ \land \ w' = w+1 \ \land \ t' = t
\end{align*}

\begin{align*}
&= \exists M. \ \forall^* \text{int}. \ \exists T. \ \forall^* \text{xnat}. \ \exists r, r', w, w': \text{xnat}. \\
&\quad M_0 = 5 \ \land \ T_0 = max \ t (\mathcal{T}_0 + 1) \ \land \ r' = 1 \ \land \ w' = 1 \ \land \ t' = max \ t (\mathcal{T}_0 + 1)
\end{align*}

\begin{align*}
&= \ t' = \infty
\end{align*}
Deadlock

\[\textbf{chan} \ c, d: \text{int}\cdot (c?\cdot d! 6) \parallel (d?\cdot c! 7)\]

\[\textbf{chan} \ c, d: \text{int}\cdot (t:= \text{max} \ t (T_{c \cdot rc} + 1). \ c?\cdot d! 6) \parallel (t:= \text{max} \ t (T_{d \cdot rd} + 1). \ d?\cdot c! 7)\]

\[= \exists M_c, M_d: \text{\infty*int}\cdot \exists T_c, T_d: \text{\infty*xnat}\cdot \exists rc, rc', wc, wc', rd, rd', wd, wd': xnat\cdot \]
\[M_d 0 = 6 \land M_c 0 = 7 \land rc' = wc' = rd' = wd' = 1\]
\[\land T_c 0 = \text{max} \ t (T_{d 0} + 1) \land T_d 0 = \text{max} \ t (T_{c 0} + 1)\]
\[\land t' = \text{max} (\text{max} \ t (T_{d 0} + 1)) (\text{max} \ t (T_{c 0} + 1))\]

\[= t' = \infty\]
Power Series Multiplication

Input on channel $a$ : $a_0 \ a_1 \ a_2 \ ...$  \hspace{1cm} $A = a_0 + a_1 x + a_2 x^2 + ...$

Input on channel $b$ : $b_0 \ b_1 \ b_2 \ ...$  \hspace{1cm} $B = b_0 + b_1 x + b_2 x^2 + ...$

Output on channel $c$ : $c_0 \ c_1 \ c_2 \ ...$  \hspace{1cm} $C = c_0 + c_1 x + c_2 x^2 + ...$

$A_1 = a_1 + a_2 x + a_3 x^2 + ...$  \hspace{1cm} $B_1 = b_1 + b_2 x + b_3 x^2 + ...$

$A_2 = a_2 + a_3 x + a_4 x^2 + ...$  \hspace{1cm} $B_2 = b_2 + b_3 x + b_4 x^2 + ...$

$C = A \times B = a_0 \times b_0 + (a_0 \times b_1 + a_1 \times b_0) x + (a_0 \times B_2 + A_1 \times B_1 + A_2 \times b_0) \times x^2$

$\langle \text{chan} \ c : \text{rat} \rightarrow C = A \times B \rangle \ c \ \leftarrow \ (a? \parallel b?). \ c! \ a \times b.$

$\var a0: \text{rat} := a \cdot \var b0: \text{rat} := b \cdot \text{chan} \ d: \text{rat}$

$\langle \text{chan} \ c: \text{rat} \rightarrow C = A \times B \rangle \ d$

$\parallel (a? \parallel b?). \ c! a0 \times b + a \times b0. \ C = a0 \times B + D + A \times b0$

$C = a0 \times B + D + A \times b0$  \hspace{1cm} $\leftarrow \ (a? \parallel b? \parallel d?). \ c! a0 \times b + d + a \times b0. \ C = a0 \times B + D + A \times b0$

22/29
## Review

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Theory</td>
<td>laws, proof</td>
</tr>
<tr>
<td>Number Theory</td>
<td>Character Theory</td>
</tr>
<tr>
<td>Bunches</td>
<td>Sets, Strings, Lists</td>
</tr>
<tr>
<td>Functions</td>
<td>Quantifiers</td>
</tr>
<tr>
<td>Specification</td>
<td>Refinement, exact precondition, exact postcondition</td>
</tr>
<tr>
<td>Program Development</td>
<td>Time Calculation, real time, recursive time</td>
</tr>
<tr>
<td>Space Calculation</td>
<td>maximum space, average space</td>
</tr>
<tr>
<td>Scope</td>
<td>variable declaration, frame</td>
</tr>
<tr>
<td>Data Structures</td>
<td>array element assignment</td>
</tr>
<tr>
<td>Control Structures</td>
<td><strong>while-loop</strong>, loop with <strong>exit</strong>, <strong>for-loop</strong></td>
</tr>
<tr>
<td>Review</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Time Dependence</td>
<td></td>
</tr>
<tr>
<td>Assertions</td>
<td></td>
</tr>
<tr>
<td>Subprograms</td>
<td></td>
</tr>
<tr>
<td>Probabilistic Programming</td>
<td></td>
</tr>
<tr>
<td>Functional Programming</td>
<td></td>
</tr>
<tr>
<td>Recursive Data Definition</td>
<td></td>
</tr>
<tr>
<td>Recursive Program Definition</td>
<td></td>
</tr>
<tr>
<td>Theory Design and Implementation</td>
<td></td>
</tr>
<tr>
<td>Data Transformation</td>
<td></td>
</tr>
<tr>
<td>Independent Composition</td>
<td></td>
</tr>
<tr>
<td>Interactive Variables</td>
<td></td>
</tr>
</tbody>
</table>

| wait                                      |
| checking                                  |
| backtracking                              |
| function                                  |
| procedure                                 |
| random number generator                   |
| refinement                                |
| timing                                    |
| construction                              |
| induction                                 |
| data theory                               |
| program theory                            |
| sequential to parallel transformation     |
| Communication Channels                    |
Disjoint Composition

Independent composition $P\|Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \|v\|w\|Q = (P. \ v' = v) \land (Q. \ w' = w)$$

(a) Prove that if $P$ and $Q$ are implementable specifications, then $P \|v\|w\|Q$ is implementable.

Application Law $\langle v\to b \rangle a = \text{(substitute } a \text{ for } v \text{ in } b \text{)}$

Let the remaining variables (if any) be $x$. 


**Disjoint Composition**

\[ P. \ v' = v \]

\[ = \exists v'', w'', x'' : \langle v', w', x' \to P \rangle \cdot v'' w'' x'' \land v' = v'' \] expand dependent composition

\[ = \exists w'', x'' : \langle v', w', x' \to P \rangle \cdot v' w'' x'' \] one-point \( v'' \)

\[ = \exists w', x' : \langle v', w', x' \to P \rangle \cdot v' w' x' \] rename \( w'', x'' \) to \( w', x' \)

\[ = \exists w', x' : P \] apply

\[ Q. \ w' = w \]

\[ = \exists v', x' : Q \]

\[ P \mid v \mid w \mid Q = (P. \ v' = v) \land (Q. \ w' = w) = (\exists w', x' : P) \land (\exists v', x' : Q) \]
Disjoint Composition

\((P \mid v \mid w \mid Q \text{ is implementable})\) \hspace{1cm} \text{definition of implementable}

\[= \quad \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \hspace{1cm} \text{use previous result}\]

\[= \quad \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \hspace{1cm} \text{identity for } x'\]

\[= \quad \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \hspace{1cm} \text{distribution (factoring)}\]

\[= \quad \forall v, w, x \cdot (\exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P)) \land (\exists v', x' \cdot Q) \hspace{1cm} \text{distribution (factoring)}\]

\[= \quad \forall v, w, x \cdot (\exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P)) \land (\exists v', \exists w' \cdot x' \cdot Q) \hspace{1cm} \text{splitting law}\]

\[= \quad (\forall v, w, x \cdot \exists v', w', x' \cdot P) \land (\forall v, w, x \cdot \exists v', w', x' \cdot Q) \hspace{1cm} \text{definition of implementable}\]

\[= \quad (P \text{ is implementable}) \land (Q \text{ is implementable})\]
Independent composition \( P \parallel Q \) requires that \( P \) and \( Q \) have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both \( P \) and \( Q \) to use all the variables with no restrictions, and then to choose disjoint sets of variables \( v \) and \( w \) and define

\[
P | v | w | Q = (P. \; v' = v) \land (Q. \; w' = w)
\]

(b) Describe how \( P | v | w | Q \) can be executed.

Make a copy of all variables. Execute \( P \) using the original set of variables and in parallel execute \( Q \) using the copies. Then copy back from the copy \( w \) to the original \( w \). Then throw away the copies.
Disjoint Composition

Independent composition $P \parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \mid v \mid w \mid Q = (P. \; v' = v) \land (Q. \; w' = w)$$

(b) Describe how $P \mid v \mid w \mid Q$ can be executed.

$$P \mid v \mid w \mid Q \iff \text{var } cv := v \cdot \text{var } cw := w \cdot \text{var } cx := x \cdot$$

$$(P \parallel \langle v, w, x, v', w', x' \rightarrow Q \rangle \; cv \; cw \; cx \; cv' \; cw' \; cx'). \; w := cw$$