**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 \( \text{ivar } x: T \cdot S = \exists x: \text{time} \to T \cdot S \)

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

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

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

$a := e = a' = e \land b' = b \land t' = t$

$x := e = a' = a \land b' = b \land x' = e \land (\forall t''. \ t \leq t'' \leq t' \Rightarrow y'' = y) \land t' = t + \text{(the time required to evaluate and store } e \text{)}$

$P \cdot Q = \exists a'', b'', t'' \cdot (\text{substitute } a'', b'', t'' \text{ for } a', b', t' \text{ in } P) \land (\text{substitute } a'', b'', t'' \text{ for } a, b, t \text{ in } Q)$

$P || Q = \exists t_P, t_Q \cdot (\text{substitute } t_P \text{ for } t' \text{ in } P) \land (\text{substitute } t_Q \text{ for } t' \text{ in } Q) \land t' = \max t_P t_Q \land (\forall t''. \ t_P \leq t'' \leq t' \Rightarrow x t'' = x(t_P)) \land (\forall t''. \ t_Q \leq t'' \leq t' \Rightarrow y t'' = y(t_Q)) \land \text{interactive variables of } P \land \text{interactive variables of } Q$
Interactive Variables

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

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

$$= (a'=a \land xt'=2 \land t'=t+1. \ a'=a \land xt'= xt+yt \land t'=t+1. \ a'=a \land xt'= xt+yt \land t'=t+1)$$

$$\| (b'=b \land yt'=3 \land t'=t+1. \ b'=b \land yt'= xt+yt \land t'=t+1)$$

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

$$\| (b'=b \land y(t+1)=3 \land y(t+2)= x(t+1)+y(t+1) \land t'=t+2)$$

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

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

$$\land a'=a \land b'=b \land t'=t+3$$

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

thermometer || control || thermostat || 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 $T$ turns the gas on and $\perp$ turns the gas off.
• binary spark; assigning it $T$ 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.

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

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

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

Channel $c$ is described by

- message script $\mathcal{M}_c$ string constant
- time script $\mathcal{T}_c$ string constant
- read cursor $rc$ extended natural variable
- write cursor $wc$ extended natural variable

\[
\mathcal{M} = \{6, 4, 7, 1, 0, 3, 8, 9, 2, 5, \ldots\}
\]
\[
\mathcal{T} = \{3, 5, 5, 20, 25, 28, 31, 31, 45, 48, \ldots\}
\]
↑       ↑
  r      w
Input and Output

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

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

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

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

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

\[ M = 6 ; 4 ; 7 ; 1 ; 0 ; 3 ; 8 ; 9 ; 2 ; 5 ; \ldots \]

\[ T = 3 ; 5 ; 5 ; 20 ; 25 ; 28 ; 31 ; 31 ; 45 ; 48 ; \ldots \]

\[ \uparrow \quad \uparrow \]

\[ r \quad w \]
Input and Output

\[ c! e = \mathcal{M}_w e \land T_w = t \land (w := w+1) \]
\[ c! = T_w = t \land (w := w+1) \]
\[ c? = r := r+1 \]
\[ c = \mathcal{M}_{r-1} \]
\[ \sqrt{c} = T_r \leq t \]

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 $d$.

$$S = \forall n: \text{nat} \cdot M_{d \cdot w + n} = 2 \times M_{c \cdot r + n}$$

$$S \iff c? \cdot d! 2 \times c \cdot S$$

**proof**

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

$$= rc := rc + 1. \ M_{d \cdot w} = 2 \times M_{c \cdot r - 1} \land (wd := wd + 1). \ S$$

$$= M_{d \cdot w} = 2 \times M_{c \cdot r} \land \forall n: \text{nat} \cdot M_{d \cdot w + 1 + n} = 2 \times M_{c \cdot r + 1 + n}$$

$$= \forall n: \text{nat} \cdot M_{d \cdot w + n} = 2 \times M_{c \cdot r + n}$$

$$= S$$
**Communication Timing**

**real time** need to know implementation

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

input $c?$ becomes $t := max \left( Tc_{rc} + 1 \right)$. $c$?

check $\sqrt{c}$ becomes $Tc_{rc} + 1 \leq t$
Communication Timing

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

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

**proof**

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

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

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

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

\[ = W \]
Recursive Communication

\[ \text{dbl} = c?. \ d! 2 \times c. \ t := t+1. \ \text{dbl} \]

**weakest solution**

\[ \forall n: \text{nat} \cdot M d_{wd+n} = 2 \times M c_{rc+n} \land T d_{wd+n} = t+n \]

**strongest implementable solution**

\[ (\forall n: \text{nat} \cdot M d_{wd+n} = 2 \times M c_{rc+n} \land T d_{wd+n} = t+n) \land rc' = wd' = t' = \infty \land wc' = wc \land rd' = rd \]

**strongest solution**

\[ \bot \]

\[ \forall n: \text{nat} \cdot M d_{wd+n} = 2 \times M c_{rc+n} \land T d_{wd+n} = t+n \iff \text{dbl} \]

\[ \text{dbl} \iff c?. \ d! 2 \times c. \ t := t+1. \ \text{dbl} \]
Recursive Construction

\[ \text{dbl}_0 \quad = \quad T \]

\[ \text{dbl}_1 \quad = \quad c? \cdot d! \cdot 2 \times c \cdot t := t + 1 \cdot \text{dbl}_0 \]
\[ = \quad rc := rc + 1 \cdot \text{Md}_{wd} = 2 \times M_{rc-1} \land Td_{wd} = t \land (wd := wd + 1) \cdot t := t + 1 \cdot T \]
\[ = \quad \text{Md}_{wd} = 2 \times M_{rc} \land Td_{wd} = t \]

\[ \text{dbl}_2 \quad = \quad c? \cdot d! \cdot 2 \times c \cdot t := t + 1 \cdot \text{dbl}_1 \]
\[ = \quad rc := rc + 1 \cdot \text{Md}_{wd} = 2 \times M_{rc-1} \land Td_{wd} = t \land (wd := wd + 1) \cdot t := t + 1. \]
\[ \text{Md}_{wd} = 2 \times M_{rc} \land Td_{wd} = t \]
\[ = \quad \text{Md}_{wd} = 2 \times M_{rc} \land Td_{wd} = t \land \text{Md}_{wd+1} = 2 \times M_{rc+1} \land Td_{wd+1} = t + 1 \]

\[ \text{dbl}_\infty \quad = \quad \forall n: \text{nat} \cdot \text{Md}_{wd+n} = 2 \times M_{rc+n} \land Td_{wd+n} = t + n \]
Monitor

\[
\text{monitor} = (\sqrt{x0in} \lor T x0in \text{ }_{rx0in} = m) \land (x0in? \cdot x := x0in \cdot x0ack!)
\]

\[
\lor (\sqrt{x1in} \lor T x1in \text{ }_{rx1in} = m) \land (x1in? \cdot x := x1in \cdot x1ack!)
\]

\[
\lor (\sqrt{x0req} \lor T x0req \text{ }_{rx0req} = m) \land (x0req? \cdot x0out! x)
\]

\[
\lor (\sqrt{x1req} \lor T x1req \text{ }_{rx1req} = m) \land (x1req? \cdot x1out! x).
\]
monitor \iffalse \textit{x}0\in. \textit{x}:= \textit{x}0\in. \textit{x}0\texttt{ack}! \else \textit{ok} \fi.

\textit{if} \textit{x}1\in \then \textit{x}1\in?. \textit{x}:= \textit{x}1\in. \textit{x}1\texttt{ack}! \else \textit{ok} \fi.

\textit{if} \textit{x}0\texttt{req} \then \textit{x}0\texttt{req}?.. \textit{x}0\texttt{out}! \textit{x} \else \textit{ok} \fi.

\textit{if} \textit{x}1\texttt{req} \then \textit{x}1\texttt{req}?.. \textit{x}1\texttt{out}! \textit{x} \else \textit{ok} \fi.

t:= t+1. \textit{monitor}
Communicating Processes

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

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

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

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

channel declaration

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

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

```plaintext
chan c: int · c! 2 || (c?. x:= c)

= ∃M: ∞*int · ∃T: ∞*xnat · var r, w: xnat := 0·
    x' = M_r · M_w = 2 · r' = r+1 · w' = w+1 · (other variables unchanged)

= ∃M: ∞*int · ∃T: ∞*xnat · var r, w: xnat ·
    x' = M_0 · M_0 = 2 · r'=1 · w'=1 · (other variables unchanged)

= x'=2 · (other variables unchanged)
= x:= 2
```

including time

```plaintext
chan c: int · c! 2 || (t:= max t (T_r + 1). c?. x:= c)

= x'=2 · t' = t+1 · (other variables unchanged)
```
Deadlock

\textbf{Deadlock}

\begin{align*}
\text{chan } c: \text{int} & \quad t := \text{max } t (T_r + 1). \quad c?: \quad c! 5 \\
= & \quad \exists M: \infty \ast \text{int} \quad \exists T: \infty \ast \text{xnat} \quad \text{var } r, w : \text{xnat} := 0. \\
& \quad t := \text{max } t (T_r + 1). \quad r := r+1. \quad M_w \ = \ 5 \quad \land \quad T_w = t \quad \land \quad (w := w+1) \\
= & \quad \exists M: \infty \ast \text{int} \quad \exists T: \infty \ast \text{xnat} \quad \exists r, r', w, w' : \text{xnat}. \\
& \quad r := 0. \quad w := 0. \quad t := \text{max } t (T_r + 1). \quad r := r+1. \\
& \quad M_w = 5 \quad \land \quad T_w = t \quad \land \quad r' = r \quad \land \quad w' = w+1 \quad \land \quad t' = t \\
= & \quad \exists M: \infty \ast \text{int} \quad \exists T: \infty \ast \text{xnat} \quad \exists r, r', w, w' : \text{xnat}. \\
& \quad M_0 = 5 \quad \land \quad T_0 = \text{max } t (T_0 + 1) \quad \land \quad r' = 1 \quad \land \quad w' = 1 \quad \land \quad t' = \text{max } t (T_0 + 1) \\
= & \quad t' = \infty
\end{align*}
Deadlock

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

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

\[ = \exists M_c, M_d: \mathbb{N}^\infty \cdot \exists T_c, T_d: \mathbb{N}^\infty \cdot \exists r_c, r_c', w_c, w_c', r_d, r_d', w_d, w_d': xnat \cdot \]

\[ M_d 0 = 6 \land M_c 0 = 7 \land r_c = w_c' = r_d' = w_d' = 1 \]

\[ \land T_c 0 = \max t (T_d 0 + 1) \land T_d 0 = \max t (T_c 0 + 1) \]

\[ \land t' = \max (\max t (T_d 0 + 1)) (\max t (T_c 0 + 1)) \]

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

Input on channel \( a : a_0 a_1 a_2 \ldots \)
\[
A = a_0 + a_1 x + a_2 x^2 + \ldots
\]

Input on channel \( b : b_0 b_1 b_2 \ldots \)
\[
B = b_0 + b_1 x + b_2 x^2 + \ldots
\]

Output on channel \( c : c_0 c_1 c_2 \ldots \)
\[
C = c_0 + c_1 x + c_2 x^2 + \ldots
\]

\[
\begin{align*}
A_1 &= a_1 + a_2 x + a_3 x^2 + \ldots \\
A_2 &= a_2 + a_3 x + a_4 x^2 + \ldots \\
B_1 &= b_1 + b_2 x + b_3 x^2 + \ldots \\
B_2 &= b_2 + b_3 x + b_4 x^2 + \ldots \\
\end{align*}
\]

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

\[
\langle \text{!c: rat } ightarrow \text{ C = A\timesB} \rangle \ c \iff (a? \parallel b?). \ c! a\times b.
\]

\[
\begin{align*}
\text{var a0: rat := a;} & \quad \text{var b0: rat := b;} \quad \text{chan d: rat;} \\
\langle \text{!c: rat } ightarrow \text{ C = A\timesB} \rangle \ d \\
\mid & \quad ((a? \parallel b?). \ c! a0\times b + a\times b0). \ C = a0\times B + D + A\times b0
\end{align*}
\]

\[
C = a0\times B + D + A\times b0 \iff (a? \parallel b? \parallel d?). \ c! a0\times b + d + a\times b0. \ C = a0\times B + D + A\times b0
\]
## Review

<table>
<thead>
<tr>
<th>Category</th>
<th>Concepts</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>while-loop, loop with exit, for-loop</td>
</tr>
</tbody>
</table>
## Review

<table>
<thead>
<tr>
<th>Time Dependence</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertions</td>
<td>checking backtracking</td>
</tr>
<tr>
<td>Subprograms</td>
<td>function procedure</td>
</tr>
<tr>
<td>Probabilistic Programming</td>
<td>random number generator</td>
</tr>
<tr>
<td>Functional Programming</td>
<td>refinement timing</td>
</tr>
<tr>
<td>Recursive Data Definition</td>
<td>construction induction</td>
</tr>
<tr>
<td>Recursive Program Definition</td>
<td>construction induction</td>
</tr>
<tr>
<td>Theory Design and Implementation</td>
<td>data theory program theory</td>
</tr>
<tr>
<td>Data Transformation</td>
<td></td>
</tr>
<tr>
<td>Independent Composition</td>
<td>sequential to parallel transformation</td>
</tr>
<tr>
<td>Interactive Variables</td>
<td>Communication Channels</td>
</tr>
</tbody>
</table>
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

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

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

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

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

Disjoint Composition

\[ P. \ v' = v \]  
\[ = \exists v'', w'', x'' \cdot \langle v', w', x' \rightarrow P \rangle \ v'' w'' x'' \land v' = v'' \]  
\[ = \exists w'', x'' \cdot \langle v', w', x' \rightarrow P \rangle \ v' w'' x'' \]  
\[ = \exists w', x' \cdot \langle v', w', x' \rightarrow P \rangle \ v' w' x' \]  
\[ = \exists w', x' \cdot P \]  

\[ Q. \ w' = w \]  
\[ = \exists v', x' \cdot Q \]  

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

\((P \mid v \mid w \mid Q \text{ is implementable})\)

\(= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q\)

\(= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\)

\(= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\)

\(= \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\)

\(= \forall v, w, x \cdot (\exists v' \cdot \exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\)

\(= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\exists v', w', x' \cdot Q)\)

\(= (\forall v, w, x \cdot \exists v', w', x' \cdot P) \land (\forall v, w, x \cdot \exists v', w', x' \cdot Q)\)

\(= (P \text{ is implementable}) \land (Q \text{ is implementable})\)
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

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

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

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.
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

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

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

$$P |v|w|Q \iff \text{var cv:=v \ var cw:=w \ var cx:=x}.
(P \parallel \langle v, w, x, v', w', x' \rightarrow Q \rangle \ cv \ cw \ cx \ cv' \ cw' \ cx'). \ w:= cw$$