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 \( \textbf{var} \ a: T \cdot S = \exists a, a': T \cdot S \)

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

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 \( t \)

\[
\text{ok} \quad = \quad a' = a \land b' = b \land t' = t
\]

\[
x' = x \land y' = y \quad \text{means} \quad x't = xt \land y't = yt
\]
Interactive Variables

**suppose** boundary \(a, b\); interactive \(x, y\); time \(t\)

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

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

$$(x := 2. \ x := x+y. \ x := x+y) \parallel (y := 3. \ y := x+y)$$

$x$ left, $y$ right, $a$ left, $b$ right

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

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

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

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

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

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

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

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

$$thermostat = (gas:= \bot || spark:= \bot). \text{ GasOff}$$

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

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

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

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

\[
\uparrow \quad \uparrow
\]

\[
rc \quad wc
\]
Input and Output

\[ c! \ e = \mathcal{M}_w = e \land \mathcal{T}_w = t \land (w := w + 1) \]

\[ c! = \mathcal{T}_w = t \land (w := w + 1) \]

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

\[ c = \mathcal{M}_{r-1} \]

\[ \sqrt{c} = \mathcal{T}_r \leq t \]

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

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

\[ \uparrow \quad \uparrow \]

\[ r \quad w \]
Input and Output

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

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

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

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

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

\[
\text{if } \sqrt{\text{key}} \\
\text{then } \text{key}?. \\
\text{if } \text{key} = \text{“y”} \\
\text{then } \text{screen}! \text{ “If you wish.”} \\
\text{else } \text{screen}! \text{ “Not if you don't want.” } \text{fi} \\
\text{else } \text{screen}! \text{ “Well?” } \text{fi}
\]
Input and Output

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

$$S = \forall n: \text{nat} \cdot M_{d_{wd+n}} = 2 \times M_{c_{rc+n}}$$

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

proof

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

$= rc := rc + 1 \cdot M_{d_{wd}} = 2 \times M_{c_{rc-1}} \land (wd := wd + 1) \cdot S$

$= M_{d_{wd}} = 2 \times M_{c_{rc}} \land \forall n: \text{nat} \cdot M_{d_{wd+1+n}} = 2 \times M_{c_{rc+1+n}}$

$= \forall n: \text{nat} \cdot M_{d_{wd+n}} = 2 \times M_{c_{rc+n}}$

$= S$
Communication Timing

**real time** need to know implementation

**transit time** input and output take time 0

communication transit takes time 1

\[ t := \max (T_{rc} + 1). \]

input \( c? \) becomes \( T_{rc} + 1 \leq t \)

check \( \sqrt{c} \) becomes \( T_{rc} + 1 \leq t \)
Communication Timing

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

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

proof

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

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

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

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

strongest solution

\[
\bot
\]

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

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

\[ dbl_0 = \top \]

\[ dbl_1 = c?. \ d! 2x \ c. \ t := t+1. \ dbl_0 \]
\[ = rc := rc+1. \ M_{wd} = 2 \times M_{rc-1} \land T_{wd} = t \land (wd := wd+1). \ t := t+1. \ \top \]
\[ = M_{wd} = 2 \times M_{rc} \land T_{wd} = t \]

\[ dbl_2 = c?. \ d! 2x \ c. \ t := t+1. \ dbl_1 \]
\[ = rc := rc+1. \ M_{wd} = 2 \times M_{rc-1} \land T_{wd} = t \land (wd := wd+1). \ t := t+1. \]
\[ M_{wd} = 2 \times M_{rc} \land T_{wd} = t \]
\[ = M_{wd} = 2 \times M_{rc} \land T_{wd} = t \land M_{wd+1} = 2 \times M_{rc+1} \land T_{wd+1} = t+1 \]

\[ dbl_\infty = \forall n: \text{nat}. \ M_{wd+n} = 2 \times M_{rc+n} \land T_{wd+n} = t+n \]
Monitor

\[
\text{monitor} = (\sqrt{x0in} \lor \mathcal{T}x0in_{rx0in} = m) \land (x0in\. x := x0in. x0ack!)
\lor (\sqrt{x1in} \lor \mathcal{T}x1in_{rx1in} = m) \land (x1in\. x := x1in. x1ack!)
\lor (\sqrt{x0req} \lor \mathcal{T}x0req_{rx0req} = m) \land (x0req\. x0out! x)
\lor (\sqrt{x1req} \lor \mathcal{T}x1req_{rx1req} = m) \land (x1req\. x1out! x).
\]

monitor
monitor \iff \sqrt{x0in} \then x0in?. \; x:= x0in. \; x0ack! \else \; ok \; \fi.

if \sqrt{x1in} \then x1in?. \; x:= x1in. \; x1ack! \else \; ok \; \fi.

if \sqrt{x0req} \then x0req?. \; x0out! x \else \; ok \; \fi.

if \sqrt{x1req} \then x1req?. \; x1out! x \else \; ok \; \fi.

\; t:= t+1. \; monitor
Communicating Processes

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

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

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

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

channel declaration

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

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

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

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

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

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

\[
= \quad x := 2
\]

including time

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

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

**chan** \( c : \text{int} \cdot \ t := \text{max} \ t (T_r + 1) \). \( c? \cdot \ c! \ 5 \)

\[
\begin{align*}
\mathcal{M}_0 & = 5 \land T_0 = \text{max} \ t (T_0 + 1) \land r' = 1 \land w' = 1 \land t' = \text{max} \ t (T_0 + 1) \\
\Rightarrow & \quad \exists M : \mathbb{N}^{\times} \cdot \exists T : \mathbb{N}^{\times} \cdot \exists r, w : \mathbb{N} \cdot r := 0 \land w := 0 \land t := \text{max} \ t (T_r + 1) \land r := r + 1 \\
\Rightarrow & \quad \exists M : \mathbb{N}^{\times} \cdot \exists T : \mathbb{N}^{\times} \cdot \exists r, w, w' : \mathbb{N} \cdot r := 0 \land w := 0 \land t := \text{max} \ t (T_r + 1) \land r := r + 1 \\
\Rightarrow & \quad \mathcal{M}_w = 5 \land T_w = t \land r' = r \land w' = w + 1 \land t' = t \\
\Rightarrow & \quad t' = \infty
\end{align*}
\]
Deadlock

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

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

$$= \exists M_c, M_d: \infty \cdot \text{int}. \ \exists T_c, T_d: \infty \cdot \text{xnat}. \ \exists r_c, r_c', w_c, w_c', r_d, r_d', w_d, w_d': \text{xnat}.$$

$$M_d_0 = 6 \land M_c_0 = 7 \land r_c' = w_c' = r_d' = w_d' = 1$$

$$\land T_c_0 = \max t (T_d_0 + 1) \land T_d_0 = \max t (T_c_0 + 1)$$

$$\land t' = \max (\max t (T_d_0 + 1)) (\max t (T_c_0 + 1))$$

$$= t' = \infty$$
**Power Series Multiplication**

Input on channel $a$ : $a_0 \ a_1 \ a_2 \ ...$

$A = a_0 + a_1x + a_2x^2 + ...$

Input on channel $b$ : $b_0 \ b_1 \ b_2 \ ...$

$B = b_0 + b_1x + b_2x^2 + ...$

Output on channel $c$ : $c_0 \ c_1 \ c_2 \ ...$

$C = c_0 + c_1x + c_2x^2 + ...$

$A_1 = a_1 + a_2x + a_3x^2 + ...$

$B_1 = b_1 + b_2x + b_3x^2 + ...$

$A_2 = a_2 + a_3x + a_4x^2 + ...$

$B_2 = b_2 + b_3x + b_4x^2 + ...$

$C = A \times B = a_0b_0 + (a_0b_1 + a_1b_0)x + (a_0B_2 + A_1B_1 + A_2b_0)x^2$

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

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

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

<table>
<thead>
<tr>
<th>Binary Theory</th>
<th>Number Theory</th>
<th>Character Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>laws</td>
<td>proof</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bunches</th>
<th>Sets</th>
<th>Strings</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>Quantifiers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>Refinement</td>
<td>exact precondition</td>
<td>exact postcondition</td>
</tr>
<tr>
<td>Program Development</td>
<td>Time Calculation</td>
<td>real time</td>
<td>recursive time</td>
</tr>
<tr>
<td>Space Calculation</td>
<td>maximum space</td>
<td>average space</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>variable declaration</td>
<td>frame</td>
<td></td>
</tr>
<tr>
<td>Data Structures</td>
<td>array element assignment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Structures</td>
<td><strong>while-loop</strong></td>
<td>loop with <strong>exit</strong></td>
<td><strong>for-loop</strong></td>
</tr>
</tbody>
</table>
Review

Time Dependence
Assertions
Subprograms
Probabilistic Programming
Functional Programming
Recursive Data Definition
Recursive Program Definition
Theory Design and Implementation
Data Transformation
Independent Composition
Interactive Variables

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 $P\parallel Q$ 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

\[ P. \; v' = v \]

\[ = \exists v'', w'', x''. \langle v', w', x' \rightarrow P \rangle \; v'' \; w'' \; x'' \; \land \; v' = v'' \]  
expand dependent composition

\[ = \exists w'', x''. \langle v', w', x' \rightarrow P \rangle \; v' \; w'' \; x'' \]  
one-point \( v'' \)

\[ = \exists w', x'. \langle v', w', x' \rightarrow P \rangle \; v' \; w' \; x' \]  
rename \( w'', x'' \) to \( w', x' \)

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

\[ Q. \; w' = w \]

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

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

\[( P \mid v \mid w \mid Q \text{ is implementable})\]  
\[= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q\]  
\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  
\[= \forall v, w, x \cdot (\exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  
\[= (\forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\exists v', w', x' \cdot Q)\]  
\[= ( P \text{ is implementable}) \land ( Q \text{ is implementable})\]
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

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

(b) Describe how $P \mid v | w \mid Q$ 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.
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.

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