Theory of concurrency

Lecture 10

Communication

Communication: **Introduction**

- An event is an action without duration
  - its occurrence may require simultaneous participation by more than one independent process.
- A *communications* form a special class of events.
- A *communication* is an event
  - described by a pair $c.v$
    - c is the name of the channel on which the communication takes place,
    - v is the value of the message which passes.
- Examples:
- The set of all messages which P can communicate on channel c:
  - $ac(P) = \{v \mid c.v \in aP\}$
- Functions which extract channel and message components of a communication:
  - $channel(c.v) = c, \quad message(c.v) = v$

Communication: **Input and output**

- Let v be any member of $ac(P)$.
- A process first outputs v on the channel c and then behaves like P:
  - $(c!v \rightarrow P) = (c.v \rightarrow P)$
  - This process is initially prepared to engage in the communication event $c.v$.
- A process first inputs any value x from the channel c and then behaves like $P(x)$:
  - $(c?x \rightarrow P(x)) = (y : \{ y \mid channel(y) = c \} \rightarrow P(message(y)))$

X1. *COPYBIT* revisited:

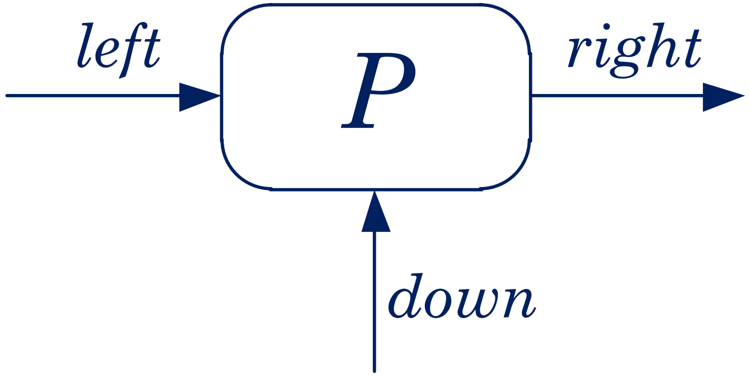
- $ain(COPYBIT) = aout(COPYBIT) = \{0, 1\}$

$$COPYBIT = \mu\, X \bullet (in?x \rightarrow (out!x \rightarrow X))$$

$$COPYBIT = \mu X \bullet (in.0 \rightarrow out.0 \rightarrow X \mid in.1 \rightarrow out.1 \rightarrow X)$$

Communication: **Input and output**

- Channels are used for communication
  - in only one direction and
  - between only two processes.
- An *output channel* of a process is the channel which is used only for output by the process.
- An *input channel* of a process is the channel which is used only for input by this process.
- The channel name is a member of the alphabet of the process.
- In a connection diagram of a process,
  - the channels are arrows in the appropriate direction,
    - labelled with the name of the channel.



Communication: **Input and output**

- In concurrent system $(P \parallel Q)$
  - P and Q are processes, c is an output channel of P and an input channel of Q.
  - Communication will occur on channel c on each occasion that
    - P outputs a message and
    - Q *simultaneously* inputs that message.
  - An outputting process specifies a *unique* value for the message
  - The inputting process is prepared to accept *any* communicable value.
  - The event that will *actually* occur is the communication $c.v$,
    - v is the value specified by the outputting process.
- The channel c must have the same alphabet at both ends:
  - $ac(P) = ac(Q)$
    - we write ac for $ac(P)$.
- In general, the value to be output by a process is specified by means of an expression containing variables to which a value has been assigned by some previous input.

Communication: **Input and output**

X1. A process immediately copies every message it has input from the left by outputting it to the right:

- $aleft(COPY) = aright(COPY)$ $\qquad COPY = \mu X \bullet (left\,?\,x \rightarrow right\,!\,x \rightarrow X)$
- If $aleft = \{0, 1\}$, $COPY$ is almost identical to $COPYBIT$.

X2. A process like $COPY$, except that every number input is doubled before it is output:

- $aleft = aright = \mathbb{N}$ $\qquad DOUBLE = \mu X \bullet (left\,?\,x \rightarrow right\,!\,(x + x) \rightarrow X)$

X3. The value of a punched card is a sequence of eighty characters, which may be read as a single value along the left channel.

- A process which reads cards and outputs their characters one at a time:
- $aleft = \{s \mid s \in aright^* \wedge \#s = 80\}$

$$UNPACK = P_{<>}$$
$$P_{<>} = left\,?\,s \rightarrow P_s$$
$$P_{<x>} = right\,!\,x \rightarrow P_{<>}$$
$$P_{<x>^\frown s} = right\,!\,x \rightarrow P_s$$

Communication: **Input and output**

X4. A process inputs characters one at a time from the left, and assembles them into lines of 125 characters' length.

- Each completed line is output on the right as a single array-valued message

$$
\begin{aligned}
&PACK = P_{<>} \\
&\quad P_s = right\,!\,s \rightarrow P_{<>} \quad \text{if } \#s = 125 \\
&\quad P_s = left\,?\,x \rightarrow P_{s^\wedge<x>} \quad \text{if } \#s < 125
\end{aligned}
$$

- $aright = \{s \mid s \in aleft^{*} \wedge \#s = 125\}$
- P_s is the process which has input and packed the characters in the sequence s;
  - they are waiting to be output when the line is long enough.

X5 A process copies from left to right, and each pair of consecutive asterisks is replaced by a single "↑":

$$
\begin{aligned}
&SQUASH = \mu X \bullet left\,?\,x \rightarrow \\
&\quad \textbf{if } x \neq \text{“*”} \textbf{ then } (right\,!\,x \rightarrow X) \\
&\quad \textbf{else } left\,?\,y \rightarrow \textbf{if } y = \text{“*”} \textbf{ then } (right\,!\,\text{“}\uparrow\text{”} \rightarrow X) \\
&\qquad\qquad \textbf{else } (right\,!\,\text{“*”} \rightarrow right\,!\,y \rightarrow X))
\end{aligned}
$$

- $aleft = aright - \{\text{“}\uparrow\text{”}\}$

Communication: **Input and output**

- A process may be prepared initially to *communicate on any one of a set of channels*, leaving the choice between them to the other processes with which it is connected.
- Communication choice:
  - $(c\ ?\ x \rightarrow P(x)\ |\ d\ ?\ y \rightarrow Q(y))$
    - c and d are distinct channel names,
    - the process which initially
      - inputs x on channel c and then behaves like $P(x)$, or
      - inputs y on channel d and then behaves like $Q(y)$.
- The choice is determined by whichever of the corresponding outputs is ready first.
  - The actions of these two processes are arbitrarily interleaved.
    - If one process is making progress towards an output on c, and
    - the other is making progress towards an output on d,
      - it is not determined which of them reaches its output first.
- The choice protects against the deadlock:
  - if the second output cannot occur, or if it can occur only after the first output

Communication: **Input and output**

X6. A process accepts input on either of the two channels *left1* or *left2*, and immediately outputs the message to the right:

- $\alpha left1 = \alpha left2 = \alpha right$

$$MERGE = (left1\,?\,x \rightarrow right\,!\,x \rightarrow MERGE \mid left2\,?\,x \rightarrow right\,!\,x \rightarrow MERGE)$$

- The output of this process is an interleaving of the messages input from *left1* and *left2*.

X7. A process is always prepared to input a value on the left, or to output to the right a value which it has most recently input

- $\alpha left = \alpha right$

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

- VAR_x behaves like a program variable with current value x.
- New values are assigned to it by communication on the left channel, and its current value is obtained by communication on the right channel.
- If $\alpha left = \{0, 1\}$ the behaviour of VAR is almost identical to that of $BOOL$.

Communication: **Input and output**

X8. A process inputs from *up* and *left,* outputs to *down* a function of what it has input, before repeating

$$NODE(v) = \mu X \bullet (up \,?\, sum \rightarrow left \,?\, prod \rightarrow down \,!\, (sum + v \times prod) \rightarrow X)$$

X9. A process is at all times ready

- to input a message on the left, and
- to output on its right the first message which it has input but not yet output

$$BUFFER = P_{<>}$$

$$P_{<>} = left \,?\, x \rightarrow P_{<x>}$$

$$P_{<x>\frown s} = (left \,?\, y \rightarrow P_{<x>\frown s \frown <y>} \mid right \,!\, x \rightarrow P_s)$$

- *BUFFER* behaves like a *queue*;
  - messages *join* the right-hand end of the queue and *leave* it from the left end,
    - in the same order as they joined,
    - but after a possible delay, during which later messages may join the queue.

$$BUFFER = P_{<>}$$
$$P_{<>} = left\ ?\ x \rightarrow P_{<x>}$$
$$P_{<x>\frown s} = (left\ ?\ y \rightarrow P_{<x>\frown s\frown <y>} \mid right\ !\ x \rightarrow P_s)$$

Communication: **Input and output**

X10. A process which behaves like a *stack* of messages.

- When empty, it responds to the signal *empty*.

- Always it is ready to input a new message from the left and put it on top of the stack;
- If nonempty, it is prepared to output and remove the top element of the stack

$$STACK = P_{<>}$$
$$P_{<>} = (empty \rightarrow P_{<>} \mid left\ ?\ x \rightarrow P_{<x>})$$
$$P_{<x>\frown s} = (right\ !\ x \rightarrow P_s \mid left\ ?\ y \rightarrow P_{<y>\frown <x>\frown s})$$

- The difference the *STACK* and the *BUFFER*:
  - If empty the *STACK* participates in the *empty* event.
  - If y is the just arrived input message, and x is the message ready for output,
    - the *STACK* stores $<y>\frown<x>\frown s$
    - the *BUFFER* stores $<x>\frown s\frown<y>$.

Communication: Input and output: **Specifications**

- In specifications, we describe
  - separately *the sequences of messages* that pass along each of the channels.
    - $tr \downarrow c = message^*(tr \restriction \{e \mid channel(e) = c\})$
      - c is a channel name
        - We will omit the $tr\downarrow$
          - write $right \leq left$
          - instead of $tr\downarrow right \leq tr \downarrow left$.

- A lower bound on the length of a prefix:
  - $s \leq^n t = (s \leq t \wedge \#t \leq \#s + n)$
    - s is a prefix of t, with not more than n items removed.
  - $s \leq^0 t \equiv (s = t)$
  - $s \leq^n t \wedge t \leq^m u \Rightarrow s \leq^{n+m} u$
  - $s \leq t \equiv \exists n \bullet s \leq^n t$

Communication: Input and output: **Specifications**

X1. $COPY$ sat $right \leq^1 left$

$COPY = \mu X \bullet (left\ ?\ x \rightarrow right\ !\ x \rightarrow X)$

X2. $DOUBLE$ sat $right \leq^1 double^*(left)$

$DOUBLE = \mu X \bullet (left\ ?\ x \rightarrow right\ !\ (x + x) \rightarrow X)$

X3. $UNPACK$ sat $right \leq\ \hat{}/\ left$

- where $\hat{}/s_0, s_1,\ldots, s_{n-1} = s_0\ s_1 \ldots\ s_{n-1}$

- The output on the right is obtained by
  - flattening the sequence of sequences input on the left.

$UNPACK = P_{<>}$
$P = left\ ?\ s \rightarrow P_s$
$P_{<x>} = right\ !\ x \rightarrow P_{<>}$
$P_{<x>\hat{}s} = right\ !\ x \rightarrow P_s$

X4 $PACK$ sat $((\hat{}/\ right \leq^{125} left) \wedge (\#^*\ right \in \{125\}^*))$

- Each element output on the right is
  - a sequence of length 125, and
- the catenation of all these sequences is
  - an initial subsequence of what has been input on the left.

$PACK = P_{<>}$
$P_s = right\ !\ s \rightarrow P_{<>}$ if $\#s = 125$
$P_s = left\ ?\ x \rightarrow P_{s\hat{}<x>}$ if $\#s < 125$

Communication: Input and output: **Specifications**

- If $\oplus$ is some binary operator,
  - we may apply it distributively to the corresponding elements of two sequences.

- The length of the resulting sequence is equal to that of the shorter operand:
- $s \oplus t = \langle\rangle$ **if** $s = \langle\rangle$ **or** $t = \langle\rangle$

  $= s_0 \oplus t_0 \wedge (s' \oplus t')$ **otherwise**

- $(s \oplus t)[i] = s[i] \oplus t[i]$ for $i \leq \min(\#s, \#t)$.

- $s \leq^n t \Rightarrow (s \oplus u \leq^n t \oplus u) \land (u \oplus s \leq^n u \oplus t)$

Communication: Input and output: Specifications

X5. The Fibonacci sequence *1, 1, 2, 3, 5, 8,…* is defined by the recurrence relation

$$fib[0] = fib[1] = 1 \qquad fib[i+2] = fib[i+1] + fib[i]$$

- The second line is rewritten using the $'$ operator to left-shift the sequence by one place

$$fib'' = fib' + fib$$

- The original definition of the Fibonacci sequence may be recovered from this more cryptic form by subscripting both sides of the equation

$$\begin{array}{ll} 1, 1, 2, 3, 5, \ldots & fib \\ 1, 2, 3, 5, \ldots & +\, fib' \\ 2, 3, 5, \ldots & =\, fib'' \end{array} \qquad \begin{array}{rlll} & fib''[i] = (fib' & + fib)[i] & \\ \Rightarrow & fib'[i+1] = fib'[i] & + fib[i] & [L1] \\ \Rightarrow & fib[i+2] = fib[i+1] & + fib[i] & \end{array}$$

- If s is a finite initial subsequence of fib (with $\#s \geq 2$) then
  - instead of the equation we get the inequality $s'' \leq s' + s$
- Specification of a process *FIB* which outputs the Fibonacci sequence to the right:
  - FIB sat $(right \leq \langle 1, 1\rangle \vee (\langle 1, 1\rangle \leq right \wedge right'' \leq right' + right))$

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

Communication: Input and output: Specifications

X6. A variable with value *x* outputs on the

- right the value most recently input on the left, or
- *x*, if there is no such input.

- If the most recent action was an output, then
  - the value which was output is equal to the last item in the sequence $\langle x \rangle$^*left*
- VAR_x sat $(channel(revers(tr)_0) = right \Rightarrow revers(right_0) = revers(\langle x \rangle \frown left)_0$
  - $revers(s_0)$ is the last element of *s*.
- This process cannot be adequately specified solely in
  - terms of *the sequence of messages* on its separate channels.
    - It is also necessary to know *the order* in which the communications on separate channels are interleaved
      - the latest communication is on the right.
- This extra complexity will be necessary for processes which use the choice operator.

Communication: Input and output: Specifications

$MERGE = (left1 \,?\, x \rightarrow right \,!\, x \rightarrow MERGE \mid left2 \,?\, x \rightarrow right \,!\, x \rightarrow MERGE)$

X7. The *MERGE* process produces an interleaving of the two sequences input on *left1* and *left2*, buffering up to one message

- *MERGE* sat $\exists\, r \bullet right \leq^1 r \wedge r$ *interleaves* $(left1, left2)$

$$BUFFER = P_{<>}$$
$$P_{<>} = left \,?\, x \rightarrow P_{<x>}$$
$$P_{<x>^\frown s} = (left \,?\, y \rightarrow P_{<x>^\frown s^\frown <y>} \mid right \,!\, x \rightarrow P_s)$$

X8. *BUFFER* sat $right \leq left$

- This is the behaviour of a transparent communications protocol
  - the guarantee of delivering on the right
    - only those messages which have been submitted on the left, and
    - in the same order.
- The protocol achieves this in spite of the facts that
  - the place where the messages are submitted is separated from the place where they are received, and
  - the communications medium connecting the two places is somewhat unreliable.

Communication: **Communications**

- Let P and Q be processes, and let c be a channel used for output by P and for input by Q.
- All communication events $c.v$ are in $aP \cap aQ$.
- In concurrent system $(P \parallel Q)$, a communication $c.v$ can occur only when
  - both processes engage simultaneously in that event
    - whenever P outputs a value v on the channel c, and
    - Q simultaneously inputs the same value.
- The outputting process determines which actual message value is transmitted
  - An inputting process is prepared to accept *any* communicable value.
- Thus output may be regarded as a specialised case of the prefix operator, and input a special case of choice:

L1. $(c\,!\,v \rightarrow P) \parallel (c\,?\,x \rightarrow Q(x)) = c\,!\,v \rightarrow (P \parallel Q(v))$

- $c!v$ on the right-hand side is an observable action in the behaviour of the system.

Communication: **Communications**

- Concealment of internal communications:

L2. $((c\,!\,v \rightarrow P) \parallel (c\,?\,x \rightarrow Q(x))) \setminus C = (P \parallel Q(v)) \setminus C,$ where $C = \{c.v \mid v \in \alpha c\}$

- The specification of the parallel composition of communicating processes
  - may use channel names for the sequences of messages passing on them.
- Let c be the name of a channel along which P and Q communicate.
  - In the specification of P,
    - c stands for the sequence of messages communicated by P on c.
  - In the specification of Q,
    - c stands for the sequence of messages communicated by Q.

Communication: **Communications**

- We consider that when P and Q communicate on c, the sequences of messages sent and received must at all times be *identical*.
- This sequence must satisfy *both* the specification of P and the specification of Q.
  - The same is true for all channels in the intersection of their alphabets.
- Consider a channel d in the alphabet of P but *not* of Q.
  - This channel cannot be mentioned in the specification of Q, only in the specification of P.
- A specification of the behaviour of $(P \parallel Q)$ can be formed as
  - the logical conjunction of the specification of P with that of Q.
  - This simplification is valid only when
    - the specifications of P and Q are expressed wholly
      - in terms of *the channel names*, which is not always possible.

Communication: **Communications**

X1. Let

- $P = (left\ ?\ x \rightarrow mid\ !\ (x \times x) \rightarrow P)$
- $Q = (mid\ ?\ y \rightarrow right\ !\ (173 \times y) \rightarrow Q)$

- Clearly
  - P sat $(mid \leq^1 square^*(left))$
  - Q sat $(right \leq^1 173 \times mid)$
    - where $(173 \times mid)$ multiples each message of mid by 173.

- It follows that
  - $(P \parallel Q)$ sat $(right \leq^1 173 \times mid) \wedge (mid \leq^1 square^*(left))$

- The specification here implies
  - $right \leq 173 \times square^*(left)$
    - which was presumably the original intention.

Communication: **Communications**

- A physical implementation of concurrent processes with $\|$
  - electronic components are connected by channels (wires) for communication.
- A desirable feature of such an implementation is to
  - increase the speed with which useful results can be produced.
  - When the same calculation must be performed on each member of a stream of input data, and
  - the results must be output at the same rate as the input, but possibly after a delay.
    - *Data flow networks*.
- A picture of communicating processes represents their physical realisation.
- An output channel of one process is connected to a like-named input channel of the other process, but channels in the alphabet of only one process are left free:

Communication: **Communications**

X2. Two streams of numbers are to be input from *left1* and *left2*.

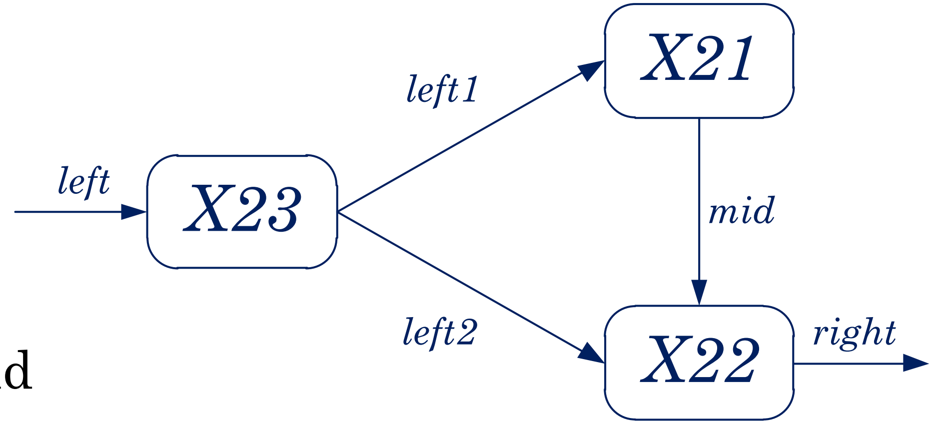
- For each x read from *left1* and each y from *left2*,
  - the number $(a \times x + b \times y)$ is to be output on the right.
- The speed requirement dictates that the multiplications must proceed concurrently.
- We therefore define two processes, and compose them

$$X21 = (left1\ ?\ x \rightarrow mid\ !\ (a \times x) \rightarrow X21)$$
$$X22 = (left2\ ?\ y \rightarrow mid\ ?\ z \rightarrow right\ !\ (z + b \times y) \rightarrow X22)$$
$$X2 = (X21 \parallel X22)$$

- Clearly,
- $X2$ sat $(mid \leq^1 a \times left1 \wedge right \leq^1 mid + b \times left2)$

$$\Rightarrow (right \leq^1 a \times left1 + b \times left2)$$

Communication: **Communications**



X3. A stream of numbers is to be input on the left, and

- on the right is output a weighted sum of consecutive pairs of input numbers,
  - with weights a and b.
- We require that $right \leq a \times left + b \times left$
- The solution can be constructed by adding a new process $X23$ to the solution of X2

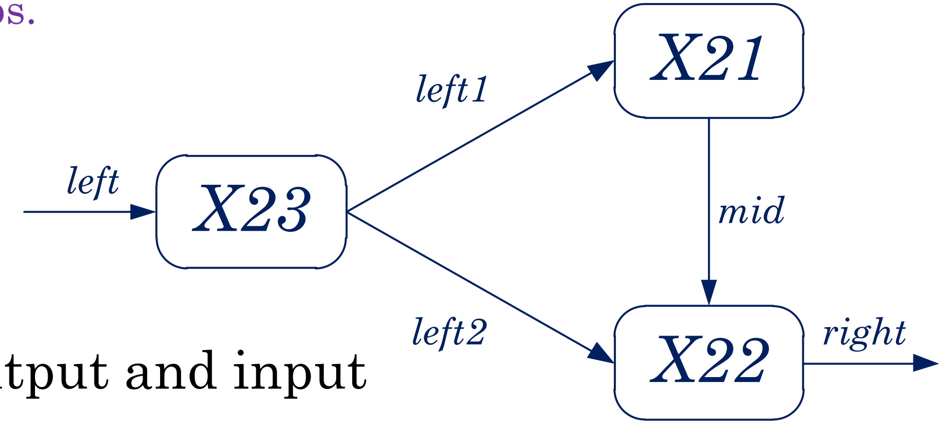
$$X3 = (X2 \parallel X23)$$
$$X23 \text{ sat } (left1 \leq^{1} left \wedge left2 \leq^{1} left)$$
$$X23 = (left\ ?\ x \rightarrow left1\ !\ x \rightarrow (\mu\ X \bullet left\ ?\ x \rightarrow left2\ !\ x \rightarrow left1\ !\ x \rightarrow X))$$

- It copies from *left* to both *left1* and *left2*, but omits the first element in the case of *left2*.

$$X21 = (left1\ ?\ x \rightarrow mid\ !\ (a \times x) \rightarrow X21)$$
$$X22 = (left2\ ?\ y \rightarrow mid\ ?\ z \rightarrow right\ !\ (z + b \times y) \rightarrow X22)$$
$$X2 = (X21 \parallel X22)$$

L2. If P and Q never stop **and** $|\alpha P \cap \alpha Q| \leq 1$, **then** $(P \parallel Q)$ never stops.

Communication: **Communications**



- When two concurrent processes communicate by output and input
  - *only on a single channel*, they cannot deadlock.
- Any network of nonstopping processes which is free of cycles *cannot deadlock*,
  - an acyclic graph can be decomposed into subgraphs connected only by a single arrow.
- The network of X3 contains an undirected cycle
  - cyclic networks cannot be decomposed into subnetworks except with connections on two or more channels.
  - In this case absence of deadlock cannot so easily be assured.
    - If in the loop of X3, we reverse the two outputs: $left1\,!\,x \rightarrow left2\,!\,x \rightarrow \ldots$
      - deadlock occurs rapidly.

$X21 = (left1\,?\,x \rightarrow mid\,!\,(a \times x) \rightarrow X21)$

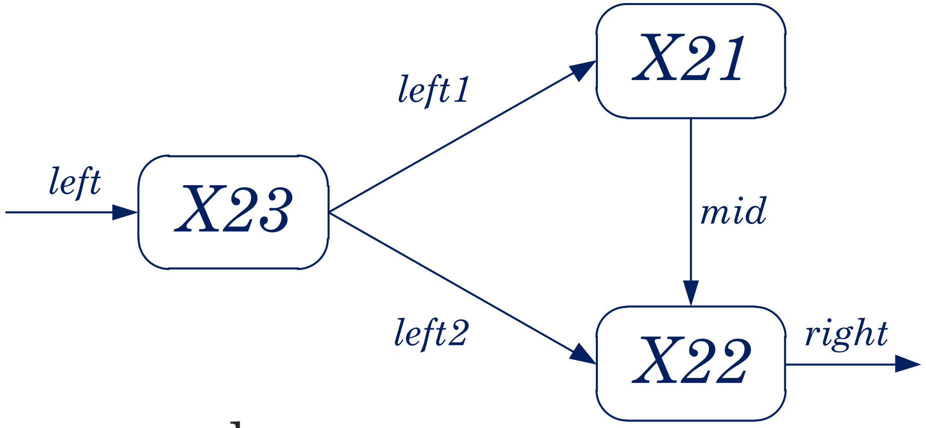
$X22 = (left2\,?\,y \rightarrow mid\,?\,z \rightarrow right\,!\,(z + b \times y) \rightarrow X22)$

$X2 = (X21 \parallel X22)$

$X3 = (X2 \parallel X23)$

$X23 = (left\,?\,x \rightarrow left1\,!\,x \rightarrow (\mu\, X \bullet left\,?\,x \rightarrow left2\,!\,x \rightarrow left1\,!\,x \rightarrow X))$

Communication: **Communications**



- In proving the absence of deadlock
  - it is often possible to *ignore the content* of the messages, and
  - regard each communication on channel *c* as *a single event named c*.
- Communications on unconnected channels can be ignored.
- X3 can be written in terms of these events

$$
\begin{aligned}
&(\mu\, X \bullet left1 \rightarrow mid \rightarrow X) \\
&\quad \parallel (\mu\, Y \bullet left2 \rightarrow mid \rightarrow Y) \\
&\quad \parallel (left1 \rightarrow (\mu\, Z \bullet left2 \rightarrow left1 \rightarrow Z)) \\
&= \mu\, X3 \bullet (left1 \rightarrow left2 \rightarrow mid \rightarrow X3)
\end{aligned}
$$

- This proves that X3 cannot deadlock, using algebraic methods.

$$X21 = (left1\ ?\ x \rightarrow mid\ !\ (a \times x) \rightarrow X21)$$
$$X22 = (left2\ ?\ y \rightarrow mid\ ?\ z \rightarrow right\ !\ (z + b \times y) \rightarrow X22)$$
$$X2 = (X21 \parallel X22)$$
$$X3 = (X2 \parallel X23)$$
$$X23 = (left\ ?\ x \rightarrow left1\ !\ x \rightarrow (\mu\, X \bullet left\ ?\ x \rightarrow left2\ !\ x \rightarrow left1\ !\ x \rightarrow X))$$

Communication: **Communications**

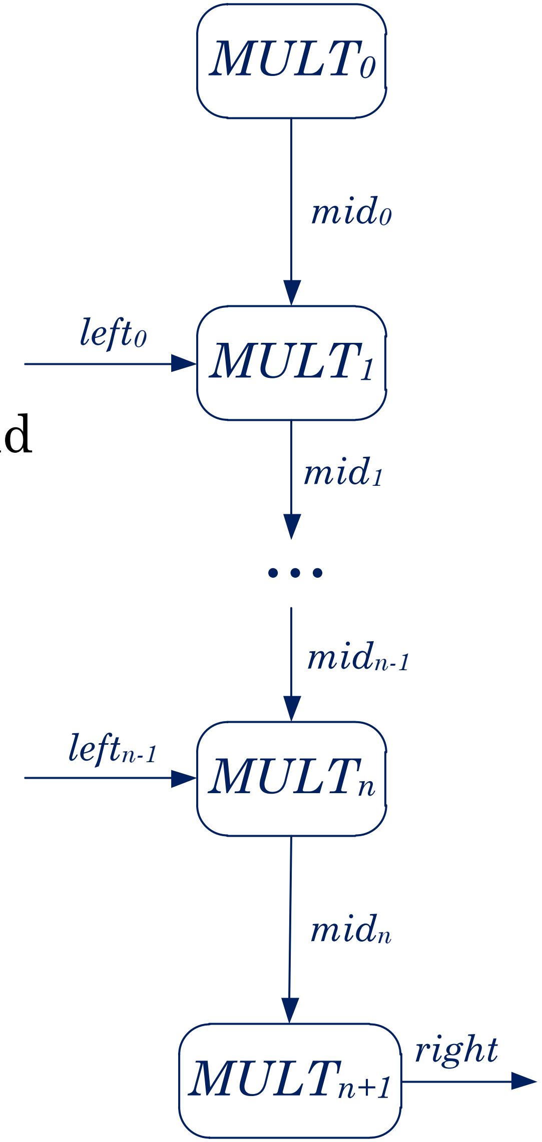
- Data flow networks can be set up
  - to compute one or more streams of results
  - from one or more streams of input data.
- The shape of the network corresponds to
  - the structure of the operands and
  - operators appearing in the expressions to be computed.
- An iterated notation for concurrent combination with subscripted names for channels:
  - $\|_{i<n} P(i) = (P(0) \| P(1) \| \ldots \| P(n-1))$
- An *iterative array* is regular network of this kind.
- A *systolic array* is a network which the connection diagram has no directed cycles.
  - Data passes through the system like blood through the chambers of the heart.

Communication: **Communications**

X4. The channels $\{\, left_j \mid j < n \,\}$ are used to input

- the coordinates of successive points in n-dimensional space.

- Each coordinate set is multiplied by a fixed vector V of length n, and
  - the resulting scalar product is output to the right:
    - $right \leq \Sigma_{j=0}^{n-1} V_j \times left_j$
      - It is specified that in each time unit
        - the n coordinates of one point are to be input and
        - one scalar product is to be output.
- For each individual processor, it takes nearly one time unit to do
  - an input, a multiplication, an addition and an output.
- At least n processors is required to operate concurrently.
- The solution to the problem:
  - an iterative array with at least n elements.



Communication: **Communications**

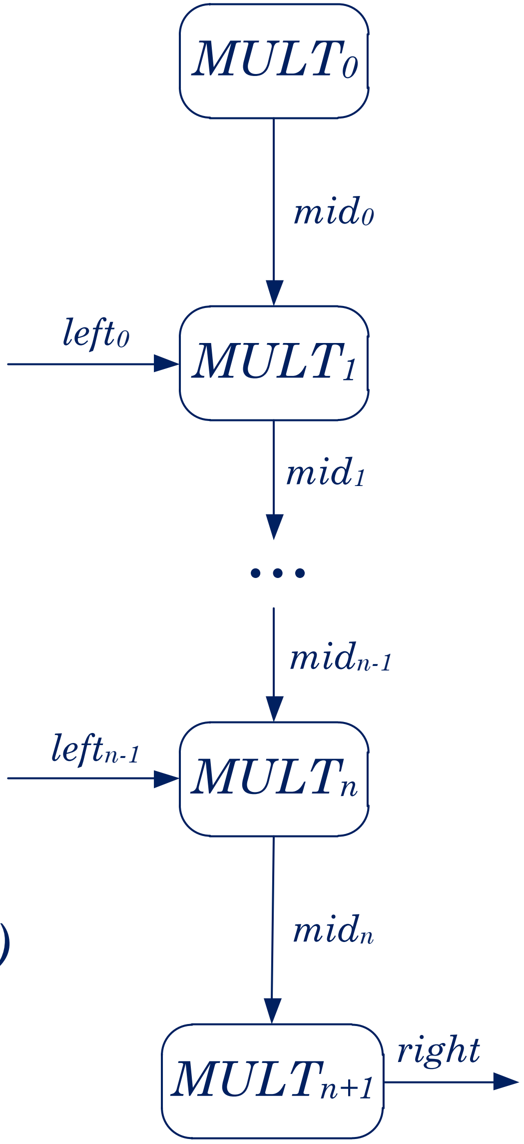
- Replace the Σ in the specification by its inductive definition:

$$mid_0 = 0^*$$
$$mid_{j+1} = V_j \times left_j + mid_j \qquad \text{for } j < n$$
$$right = mid_n$$

- The specification is split into
  - a conjunction of $n + 1$ component equations,
  - each containing at most one multiplication.

- A process for each equation (for $j < n$):

$$MULT_0 = (\mu\, X \bullet mid_0\, !\, 0 \to X)$$
$$MULT_{j+1} = (\mu\, X \bullet left_j\, ?\, x \to mid_j\, ?y \to mid_{j+1}\, !\, (V_j \times x + y) \to X)$$
$$MULT_{n+1} = (\mu\, X \bullet mid_n\, ?\, x \to right\, !\, x \to X)$$
$$NETWORK = \|_{j<n+2}\, MULT_j$$



SX4: $right \leq \Sigma_{j=0}^{n-1} V_j \times left_j$

SX4: $right \leq \Sigma_{j=0}{}^{n-1}\ V_j \times left_j$

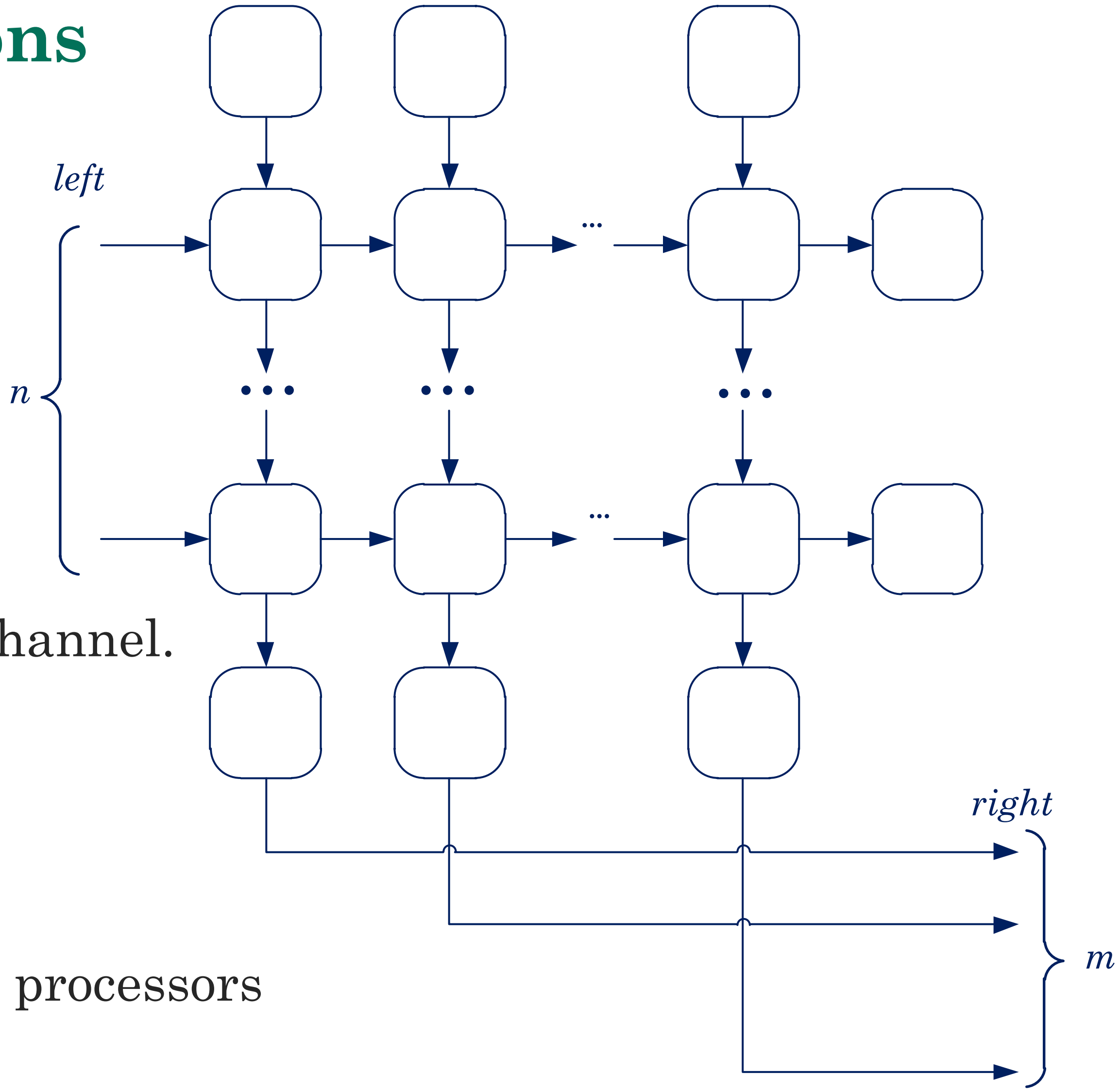
Communication: **Communications**

X5. Like X4, but m different scalar products of the same coordinate sets are required.

- The channel $left_j$ (for $j < n$) is used to input the j^{th} column of an infinite array
  - this is to be multiplied by the $(n \times m)$ matrix M, and
    - the i^{th} column of the result is to be output on $right_i$, for $i < m$.
      - $right_i = \Sigma_{j<n} M_{ij} \times left_j$
- The coordinates of the result are required as rapidly as before
  - at least $m \times n$ processes are required.
- Practical application in a graphics display device
  - automatically transforms or even rotates
    - a two-dimensional representation of a three-dimensional object.
- The shape is defined by a series of points in absolute space;
  - *the iterative array applies linear transformations* to compute
    - the deflection on the x and y plates of the cathode ray tube;
  - a third output coordinate could perhaps control the intensity of the beam.

Communication: **Communications**

- The solution is based on this Figure:
- Each column of this array except the last is
  - modelled on the solution to X4.
- it copies each value input on
  - its horizontal input channel to
    - its neighbour on its horizontal output channel.
- The processes on the right margin merely
  - discard the values they input.
- It would be possible to economise by
  - absorbing the functions of these marginal processors
    - into their neighbours.



Communication: **Communications**

X6. The input on channel c is the successive digits of a natural number C,

- starting from the least significant digit, and expressed with number base b.

- The value of the input number: $C = \Sigma_{i\geq 0} c[i] \times b^i$, where $c[i] < b$ for all i.
- Given a fixed multiplier M, the output on channel d is
  - the successive digits of the product $M \times C$: $d = \Sigma_{i\geq 0} M \times c[i] \times b^i$
- The j^{th} element of d must be the j^{th} digit:
  - $d[j] = ((\Sigma_{i\geq 0} M \times c[i] \times b^i) \text{ div } b^J) \text{ mod } b = (M \times c[j] + z_j) \text{ mod } b$
    - $z_j = (\Sigma_{i<j} M \times c[i] \times b^i) \text{ div } b^j$ and div is integer division.
- z_j is the carry term satisfied the inductive definition: $z_0 = 0$ and $z_{j+1} = ((M \times c[j] + z_j) \text{ div } b)$
- A process $MULT1(z)$, which keeps the carry z as a parameter

  $MULT1(z) = c\ ?\ x \rightarrow d\ !\ (M \times x + z) \text{ mod } b \rightarrow MULTI1((M \times x + z) \text{ div } b)$

- The initial value of z is zero, so the required solution is $MULT = MULT1(0)$

Communication: **Communications**

X7. The problem is the same as X6, except M is a multi-digit number

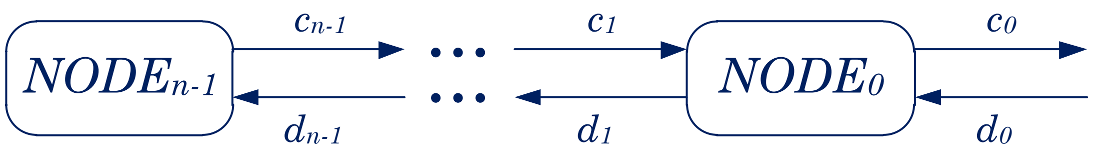
- $M = \Sigma_{i<n} M_i \times b_i$

- A single processor can multiply only single-digit numbers.
- Output is produced at a rate which allows only one multiplication per digit.
  - At least n processors are required.
- We will get each $NODE_i$ to look after one digit M_i of the multiplier.
- The basis of a solution is the traditional manual algorithm for multi-digit multiplication,
  - except that the partial sums are added immediately to the next row of the table

… 153091	C	the incoming number
253	M	the multiplier
… 306182	$M_2 \times C$	computed by $NODE_2$
… 765455	$M_1 \times C$	computed by $NODE_1$
… 827275	$25 \times C$	
… 459273	$M_0 \times C$	computed by $NODE_0$
… 732023	$M \times C$	

Communication: **Communications**

- The nodes are connected as shown in the figure:



- The original input
  - comes in on c_0 and
  - is propagated leftward on the c channels.
- The partial answers are
  - propagated rightward on the d channels, and
  - the desired answer is output on d_0.
- Fortunately each node can give one digit of its result before
  - communicating with its left neighbour.

Communication: **Communications**

- Furthermore, the leftmost node can be defined to behave like the answer to X6

$$NODE_{n-1}(z) = c_{n-1}\,?\,x \rightarrow d_{n-1}\,!\,(M_{n-1} \times x + z) \bmod b \rightarrow NODE_{n-1}((M_{n-1} \times x + z) \operatorname{div} b)$$

- Each of the remaining nodes
  - passes the input digit to its left neighbour, and
  - adds the result from its left neighbour to its own carry.

- For $k < n - 1$

$$NODE_k(z) = c_k\,?\,x \rightarrow d_k\,!\,(M_k \times x + z) \bmod b \rightarrow c_{k+1}\,!\,x \rightarrow d_{k+1}\,?\,y \rightarrow NODE_k(y + (M_k \times x + z) \operatorname{div} b)$$

- The whole network is $\Vert_{i<n}\ NODE_i(0)$

Communication: **Communications**

- The network algorithm of X7 is based on
  - a *cycle* in the directed graph of communication channels.
- The problem has been much simplified by
  - the assumption that the multiplier is *known* in advance and *fixed* for all time.
- In a practical application, such parameters would have to be
  - input along the same channel as the subsequent data, and
  - reinput whenever it is required to change them.
- The implementation of this requires great care, but little ingenuity.
- A simple implementation method is to introduce
  - a special symbol *reload* to indicate that
    - the next number or numbers are to be treated as a change of parameter;
  - a special symbol *endreload* to indicate that
    - the number of parameters is variable.

SX4: $right \le \Sigma_{j=0}^{n-1} V_j \times left_j$

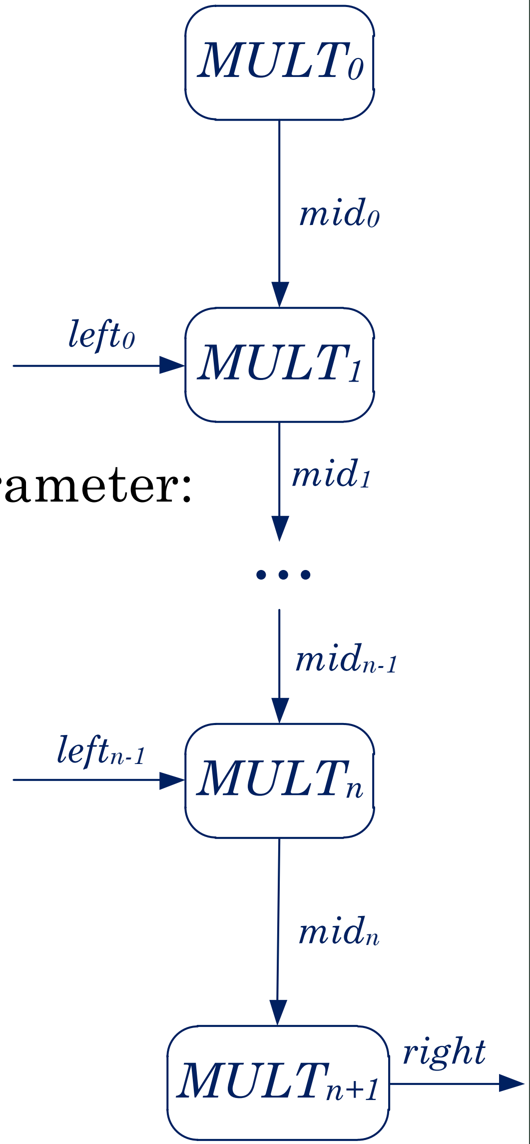
Communication: Communications

X8. Same as X4, except that the parameters V_j are

- reloaded by the number immediately following a *reload* symbol.

- The definition of $MULT_{j+1}$ is changed to include the multiplier as parameter:

$$
\begin{aligned}
&MULT_{j+1}(v) = left_j\ ?\ x \to \\
&\qquad \textbf{if}\ x = reload\ \textbf{then}\ (left_j\ ?\ y \to MULT_{j+1}(y)) \\
&\qquad \textbf{else}\ (mid_j\ ?\ y \to mid_{j+1}!\ (v \times x + y) \to MULT_{j+1}(v))
\end{aligned}
$$



$$
\begin{aligned}
MULT_0 &= (\mu\ X \bullet mid_0\ !\ 0 \to X) \\
MULT_{j+1} &= (\mu\ X \bullet left_j\ ?\ x \to mid_j\ ?y \to mid_{j+1}\ !\ (V_j \times x + y) \to X) \\
MULT_{n+1} &= (\mu\ X \bullet mid_n\ ?\ x \to right\ !\ x \to X) \\
NETWORK &= \|_{j<n+2}\ MULT_j
\end{aligned}
$$