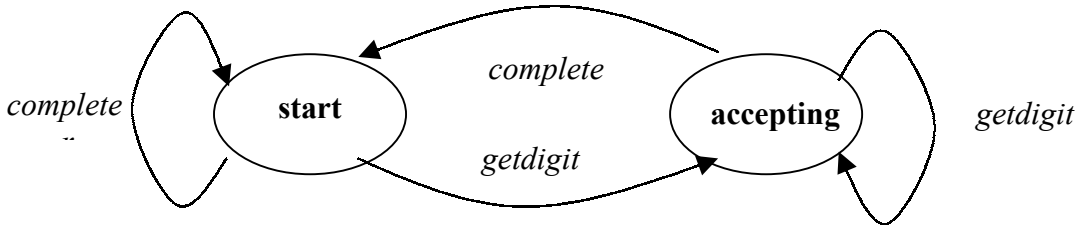
Example of Communicating Stream X-Machine

Let two Stream X-Machines:

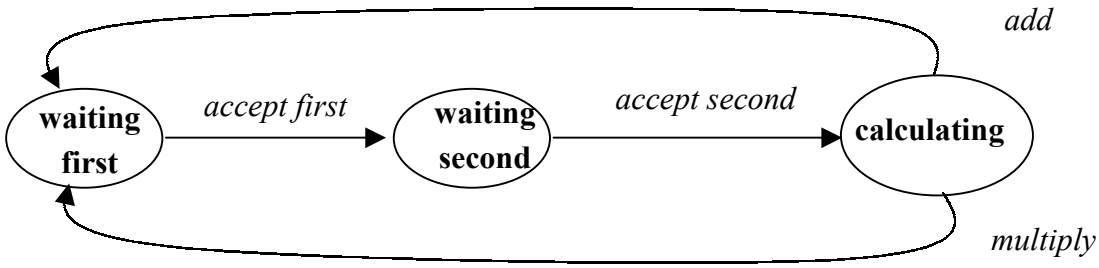
- one that generates a natural by accepting as input digits 0-9, one at a time until space is input, and
- one that accepts two naturals, puts them into memory and adds or multiples them according to input operator + or *.




```
#model generator.
#type digit = {0,1,2,3,4,5,6,7,8,9}.
#type space = {sp}.
#type weight = natural0.
#type number = natural0.
#inputset = digit union space.
#outputset = natural0.
#states = {start, accepting}.
#memory (number, weight).
#input (inputset).
#output (number).
#init_state {start}.
#init_memory (0,0).

#fun getdigit ((?d),(?n,?w)) =
      if ?d belongs digits then ((?new), (?new, ?weight))
where ?new <- ?n + ?d * 10^?w and ?weight <- ?w + 1.

#fun complete ((sp),(?n,?w)) = ((?n), (0, 0)).
```



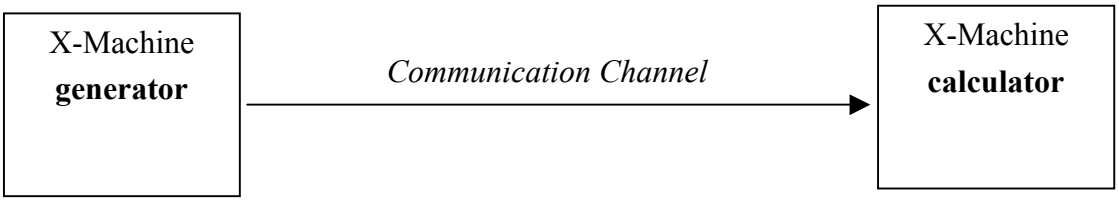

```
#model calculator.
#type number = natural0.
#type operator = {+, *}.
#inputset = operator union number.
#outputset = natural0.
#states = {waiting_first, waiting_second, calculating}.
#memory (number, number).
#input (inputset).
#output (number).
#init_state {waiting_first}.
#init_memory (0,0).

#fun waiting_first ((?n),(?m1,?m2)) = ((?n),(?n,?m2)).
#fun waiting_second ((?n),(?m1,?m2)) = ((?n),(?m1,?n)).

#fun add ((+),(?n1,?n2)) = ((?sum), (0, 0))
where ?sum <- ?n1 + ?n2.
#fun multiply ((*),(?n1,?n2)) = ((?sum), (0, 0))
where ?sum <- ?n1 * ?n2.
```

If left alone the two X-Machines would "function" independently, one generating and outputting natural numbers, while the other accepting two naturals and adding or multiplying them.

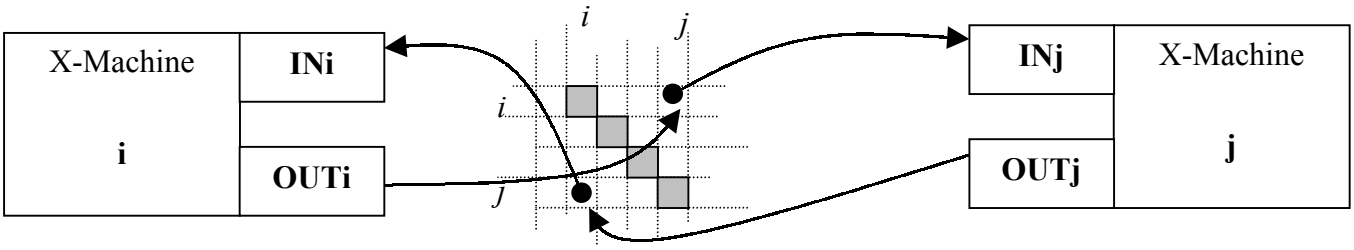
However, the two specifications may be combined in a producer-consumer fashion through a communication channel, as described in Communicating Stream X-Machine theory.



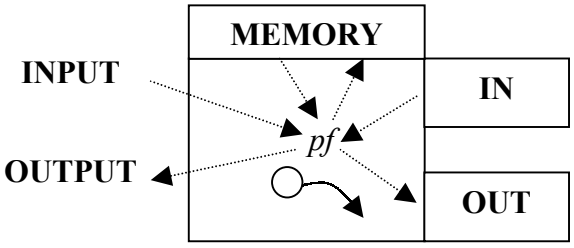
The communication in this case is from generator to calculator, but in principle it can also be bi-directional. Communication of X-Machines is established through a Communication Matrix, and Communication Functions.

- The CM is n×n where n is the number of X-Machines.
- The CM cells contain "messages" from one X-Machine to another.
- The "messages" can be of any type defined in all X-Machines memories.
- The CM cells may contain an undefined value λ which stands for "no message".
- All the (i,i) cells are empty.
- The (i, j) cell contains a "message" from X-Machine i to X-Machine j, i.e. X-Machine i reads only from i^{th} column and writes only in i^{th} row.
- Each X-Machine has an input port IN and an output port OUT.

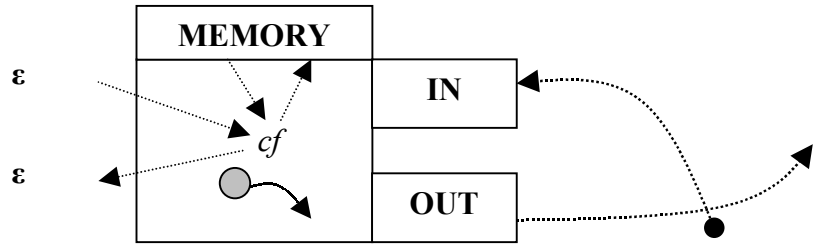
Communication Matrix



- The element types of IN and OUT are of the same type as the memory of each X-Machine.
- The Processing Functions of a X-Machine can affect the contents its IN and OUT ports.
- The Processing Functions emerge from Processing States.
- The Processing Functions do not affect the Communication Matrix.



- The Communicating Functions of a X-Machine read from the communication matrix and put what they read in the IN port, or write whatever exist in the OUT port in the communication matrix.
- The Communicating Functions emerge from Communication States.
- The Communicating Function takes ε as input, produce ε as output, and do not affect the memory.
- The Communicating Functions can write to the matrix only if the cell is λ.
- After the Communicating Functions read from the matrix the cell becomes λ.
- If a Communication Function is not applicable (e.g. cannot write if the cell is not λ), it "waits" until it becomes applicable.



Formally, the Processing functions of X-Machine are:

$pf(\sigma, m, in, out) = (\gamma, m', in', out')$

where: $\sigma \in \Sigma$, $m, m' \in M$, $in, in' \in IN$, $out, out' \in OUT$, $\gamma \in \Gamma$

and, the Communication Functions are:

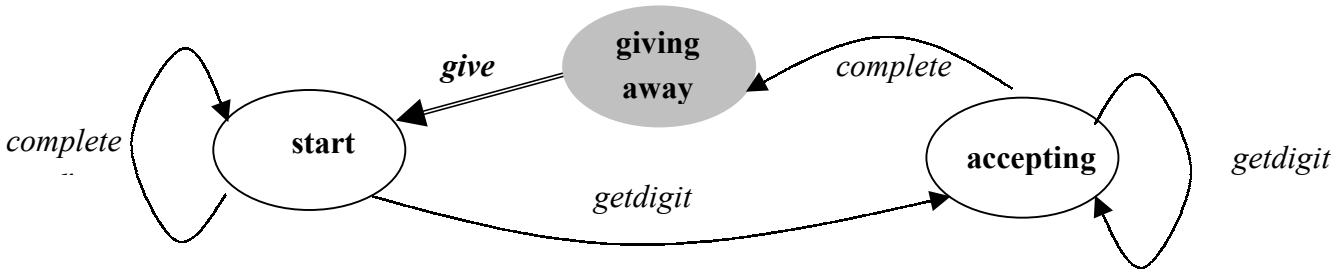
$cf(\varepsilon, m, in, out, c) = (\varepsilon, m, in', out', c')$

where: $m \in M$, $in, in' \in IN$, $out, out' \in OUT$, $c, c' \in CM$

For uniformity we can say that the type of all functions is:

$f : \Sigma \times M \times IN \times OUT \times CM \rightarrow \Gamma \times M \times IN \times OUT \times CM$

Going back to the example, the two X-Machines become:

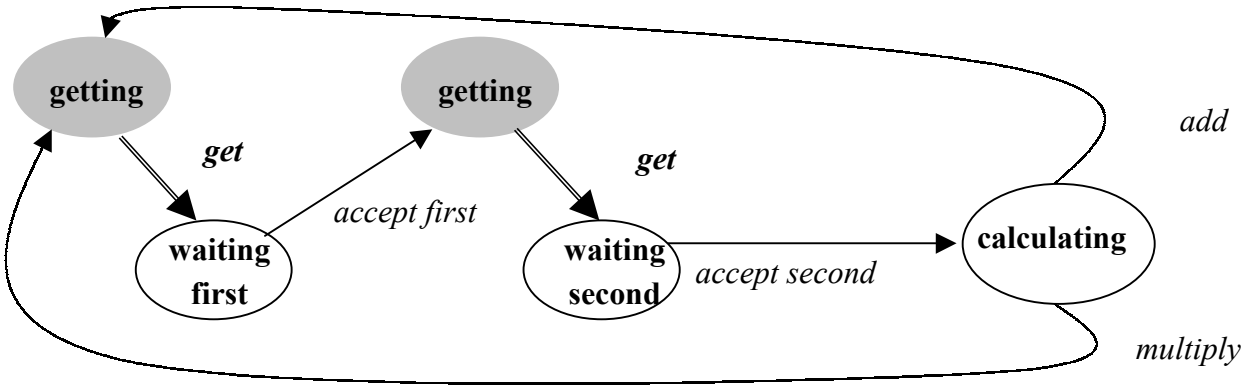


/* THE PROCESSING FUNCTIONS */

```
#fun getdigit ((?d),(?n,?w),?in, ?out, ?cm) =
      if ?d belongs digits then ((?new), (?new, ?weight), ?in, ?out, ?cm)
where ?new <- ?n + ?d * ?w and ?weight <- ?w + 1.
#fun complete ((sp),(?n,?w), ?in, ?out, ?cm) = ((?n), (0, 0), ?in, ?n, ?cm).
```

/* THE COMMUNICATING FUNCTION */

```
#fun give ((e),?m, ?in, ?out, ?cm) = ((e), ?m, ?in, lamda, ?newcm)
where newcm <- update((generator,calculator),?out).
```



/* THE PROCESSING FUNCTIONS */

```
#fun add ((+),(?n1,?n2),?in, ?out, ?cm) = ((?sum), (0, 0), ?in, ?out, ?cm)
where ?sum <- ?n1 + ?n2.

#fun multiply ((*),(?n1,?n2), ?in, ?out, ?cm) = ((?sum), (0, 0) , ?in, ?out, ?cm)
where ?sum <- ?n1 * ?n2.

#fun accept_first ((e),(?n1,?n2), ?in, ?out, ?cm) =
      ((e), (?in, ?n2) , ?in, ?out, ?cm).

#fun accept_second ((e),(?n1,?n2), ?in, ?out, ?cm) =
      ((e), (?n1, ?in) , ?in, ?out, ?cm).
```

/* THE COMMUNICATING FUNCTION */

```
#fun get ((e), ?m, ?in, ?out, ?cm) = ((e), ?m, ?newin, ?out, ?newcm)
where
      ?newin <- get((calculator,generator)) and
      ?newcm <- update((calculator,generator), lambda).
```

The Initial States of the X-Machines and the Communication matrix are:

Producer		**Matrix**	**Consumer**	
State:	start		**State**:	getting
Memory:	(0,0)	@	**Memory**:	(0,0)
Out:	λ	λ	**In**:	λ
		λ		
		@		

Animating the Communicating X-Machine will result in the following:

Producer		**Matrix**	**Consumer**	
State:	start		**State**:	getting
Input:	8	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	-
Next State:	accepting		**Next State**:	-
Memory:	(8,1)	λ	**Memory**:	(0,0)
		@		

Producer		**Matrix**	**Consumer**	
State:	accepting		**State**:	getting
Input:	4	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	-
Next State:	accepting		**Next State**:	-
Memory:	(48,2)	λ	**Memory**:	(0,0)
		@		

Producer		**Matrix**	**Consumer**	
State:	accepting		**State**:	getting
Input:	sp	@	**Input**:	ε
Applicable Function:	complete	λ	**Applicable Function**:	-
Next State:	giving away		**Next State**:	-
Memory:	(0,0)	λ	**Memory**:	(0,0)
Out:	48	@	**In**:	λ

Producer		**Matrix**	**Consumer**	
State:	giving away		**State**:	getting
Input:	e	@	**Input**:	ε
Applicable Function:	give	48	**Applicable Function**:	-
Next State:	start		**Next State**:	-
Memory:	(0,0)	λ	**Memory**:	(0,0)

Out:	λ	@	**In**:	λ

Producer		**Matrix**	**Consumer**	
State:	start		**State**:	getting
Input:	3	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	get
Next State:	accepting		**Next State**:	waiting first
Memory:	(3,1)	λ	**Memory**:	(0,0)
Out:	λ	@	**In**:	48

Producer		Matrix	Consumer	
State:	start		**State**:	waiting first
Input:	5	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	accept first
Next State:	accepting		**Next State**:	getting
Memory:	(53,2)	λ	**Memory**:	(48,0)
		@		

Producer		Matrix	Consumer	
State:	accepting		**State**:	getting
Input:	sp	@	**Input**:	ε
Applicable Function:	complete	λ	**Applicable Function**:	-
Next State:	giving away		**Next State**:	-
Memory:	(0,0)	λ	**Memory**:	(48,0)
Out:	53	@	**In**:	λ

Producer		Matrix	Consumer	
State:	giving away		**State**:	getting
Input:	e	@	**Input**:	ε
Applicable Function:	give	53	**Applicable Function**:	-
Next State:	start		**Next State**:	-
Memory:	(0,0)	λ	**Memory**:	(48,0)
Out:	λ	@	**In**:	λ

Producer		Matrix	Consumer	
State:	start		**State**:	waiting second
Input:	5	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	get
Next State:	accepting		**Next State**:	waiting second
Memory:	(5,1)	λ	**Memory**:	(48,0)
Out:	λ	@	**In**:	53

Producer		Matrix	Consumer	
State:	start		**State**:	waiting second
Input:	6	@	**Input**:	ε
Applicable Function:	getdigit	λ	**Applicable Function**:	accept second
Next State:	accepting		**Next State**:	calculating
Memory:	(65,2)	λ	**Memory**:	(48,53)
Out:	λ	@	**In**:	λ

Producer		**Matrix**	**Consumer**	
State:	start		**State**:	calculating
Input:	7	@	**Input**:	+
Applicable Function:	getdigit	λ	**Applicable Function**:	add
Next State:	accepting		**Next State**:	calculating
Memory:	(765,3)	λ	**Memory**:	(0,0)
		@		

and so on …

Drafted: P.Kefalas 20/6/00