

High-level interactive systems with registers and voices

**Alexandru Popa, Alexandru Sofronia,
and Gheorghe Stefanescu**

Faculty of Mathematics and Computer Science
University of Bucharest

IMAR, Bucharest, 1-st November, 2007



High-level srv-programs

Contents:

- *Generalities*
- A glimpse on AGAPIA programming
- Structured rv-programs
- High-level structured rv-programs
- Conclusions

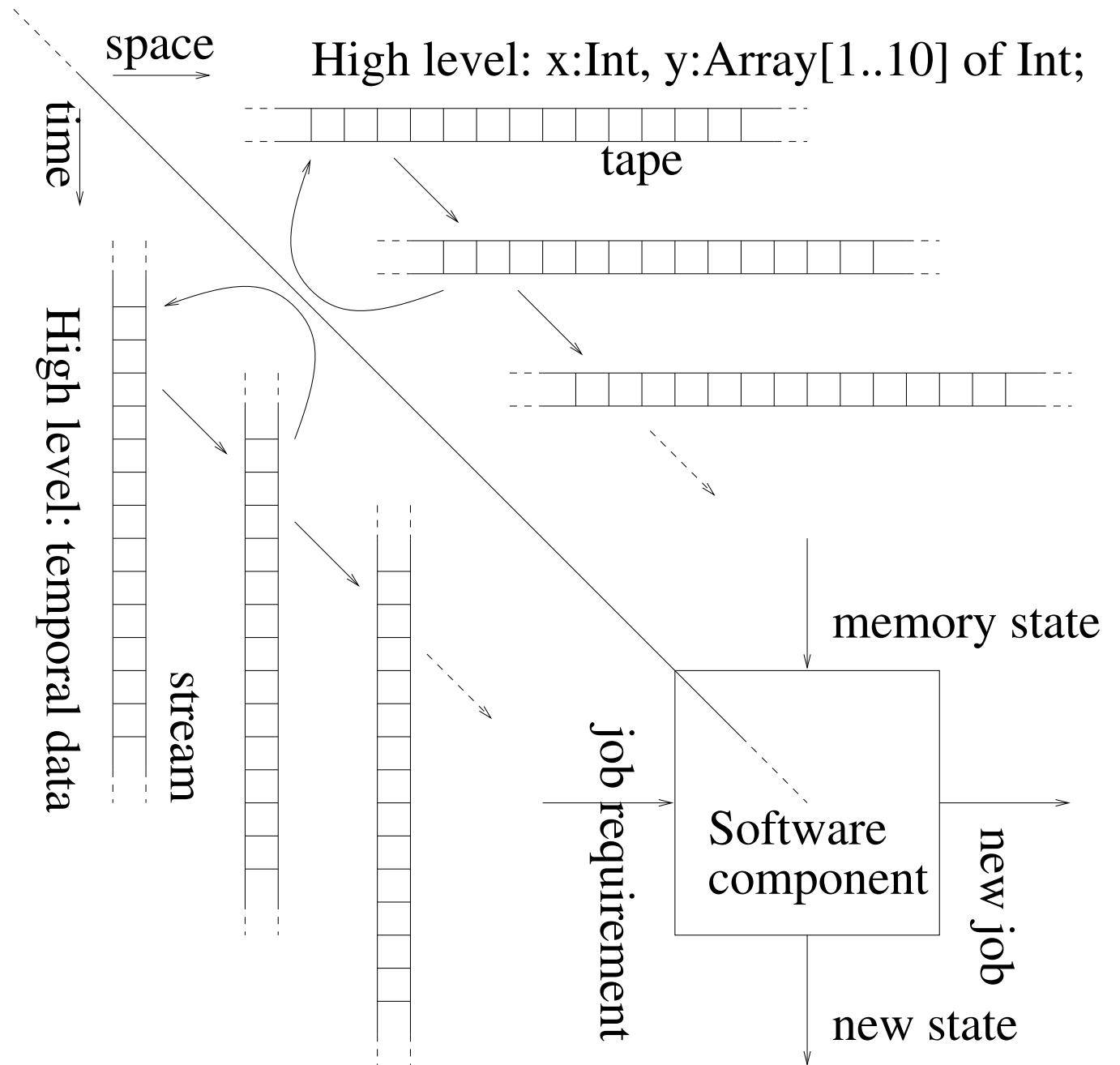
History

- *space-time duality “thesis”*
 - Stefanescu, *Network algebra*, Springer 2000
- *finite interactive systems*
 - Stefanescu, Marktoberdorf Summer School 2001
- *rv-systems* (interactive systems with registers and voices)
 - Stefanescu, NUS, Singapore, summer 2004
- *structured rv-systems*
 - Stefanescu, Dragoi, fall 2006



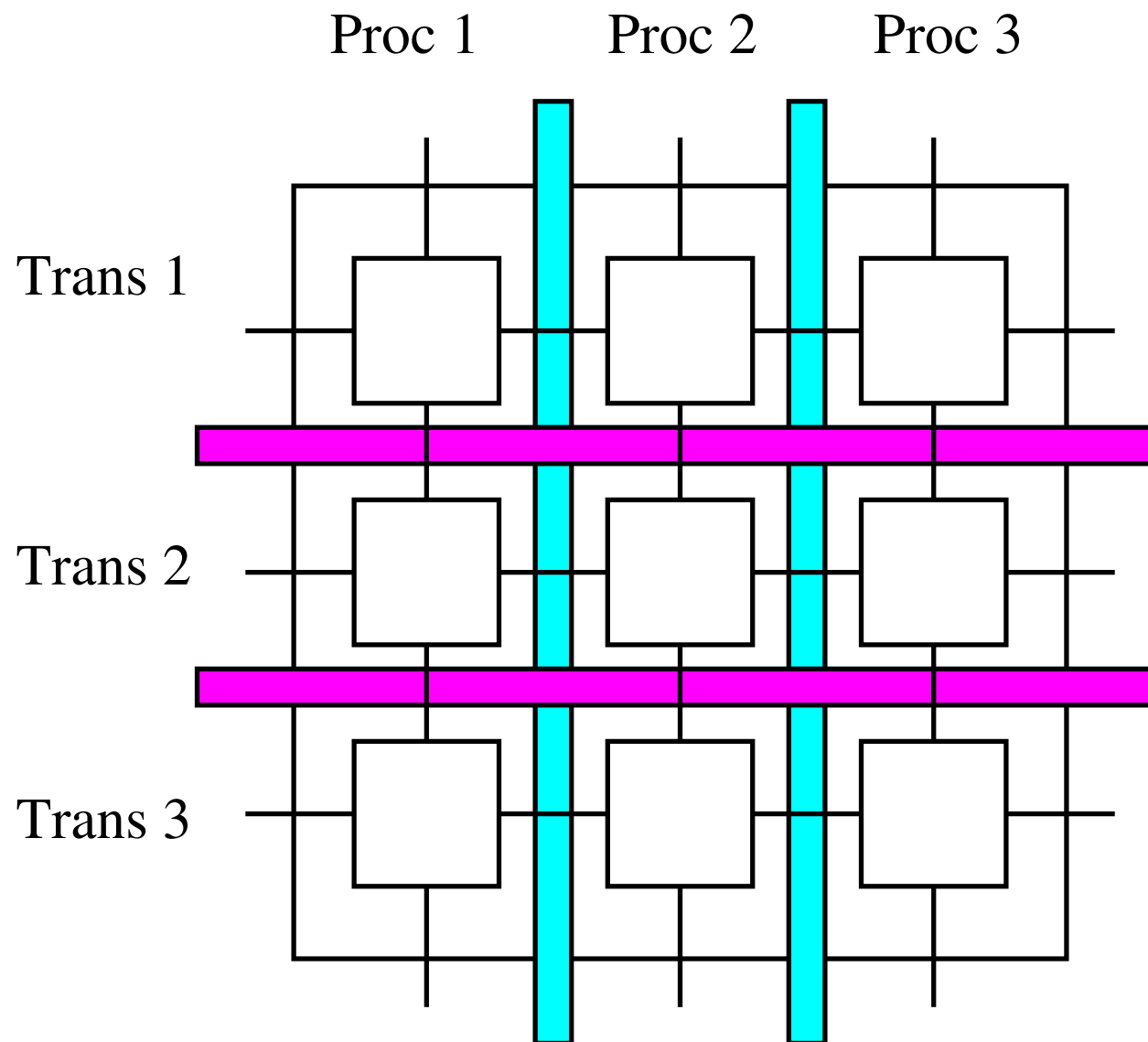
ST-Dual picture

ST-Dual picture



Processes and transactions

Processes and transactions



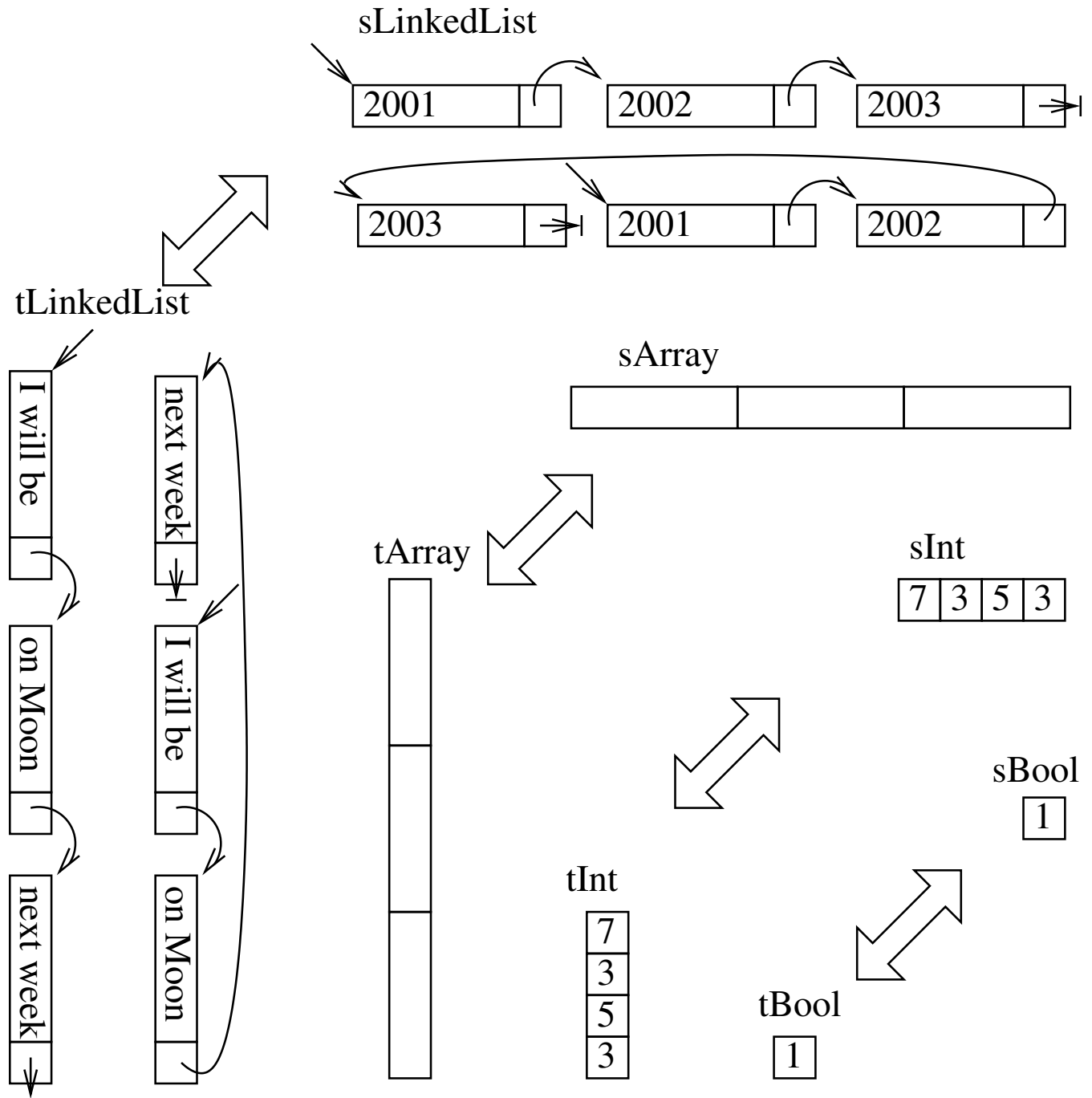
High level temporal structures

data with usual
(*spatial*)
representation:

sBool, sInt, sArray,
sLinkedList, etc.

and their *time dual*
(i.e., data with
temporal
representation):

tBool, tInt, tArray,
tLinkedList, etc.





High-level srv-programs

Contents:

- Generalities
- *A glimpse on AGAPIA programming*
- Structured rv-programs
- High-level structured rv-programs
- Conclusions



Srv-programs for perfect numbers

A specification for perfect numbers:

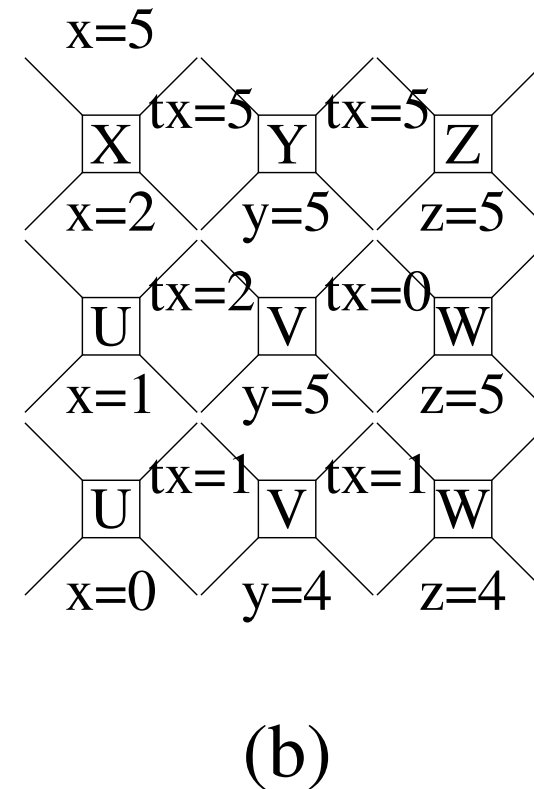
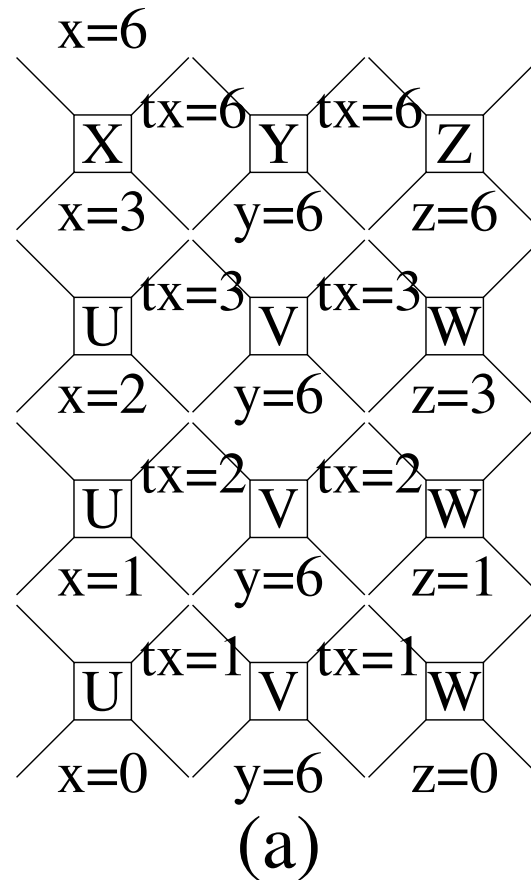
3 components C_x, C_y, C_z where:

- C_x : read n from north and write $n \frown \lfloor n/2 \rfloor \frown (\lfloor n/2 \rfloor - 1) \frown \dots \frown 2 \frown 1$ on east;
- C_y : read $n \frown \lfloor n/2 \rfloor \frown (\lfloor n/2 \rfloor - 1) \frown \dots \frown 2 \frown 1$ from west and write $n \frown \phi(\lfloor n/2 \rfloor) \frown \dots \frown \phi(2) \frown \phi(1)$ on east
[$\phi(k) = \text{“if } k \text{ divides } n \text{ then } k \text{ else } 0\text{”}$];
- C_z : read $n \frown \phi(\lfloor n/2 \rfloor) \frown \dots \frown \phi(2) \frown \phi(1)$ from west and subtract from the first the other numbers.

These components are composed *horizontally*. The global input-output specification: *if the input number in C_x is n , then the output number in C_z is 0 iff n is perfect.*

..Srv-programs for perfect numbers

Two scenarios for perfect numbers:



Types are denoted as $\langle west|north \rangle \rightarrow \langle east|south \rangle$

Our (s)rv-scenarios are similar with the tiles of Bruni-Gadducci-Montanari, et.al.



..Srv-programs for perfect numbers

The 1st AGAPIA program **Perfect1** (construction by rows):

(X # Y # Z) % while_t (x>0) {U # V # W}

Its type is **Perfect1** : $\langle nil | sn; nil; nil \rangle \rightarrow \langle nil | sn; sn; sn \rangle$.

Modules:

```
X:: module{listen nil;}{read x:sn;}
      {tx:tn; tx=x; x=x/2;}{speak tx;}{write x;}
Y:: module{listen tx:tn;}{read nil;}
      {y:sn; y=tx;}{speak tx;}{write y;}
Z:: module{listen tx:tn;}{read nil;}
      {z:sn; z=tx;}{speak nil;}{write z;}
U:: module{listen nil;}{read x:sn;}
      {tx:tn; tx=x; x=x-1;}{speak tx;}{write x;}
V:: module{listen tx:tn;}{read y:sn;}
      {if(y%tx != 0) tx=0;}{speak tx;}{write y;}
W:: module{listen tx:tn;}{read z:sn}
      {z=z-tx;}{speak nil;}{write z;}
```



..Srv-programs for perfect numbers

The 2nd AGAPIA program **Perfect2** (construction by columns):

```
(X % while_t (x>0) {U} % U1)
# (Y % while_t (tx>-1) {V} % V1)
# (Z % while_t (tx>-1) {W} % W1)
```

Its type is **Perfect2** : $\langle nil|sn;nil;nil \rangle \rightarrow \langle nil|nil;nil;sn \rangle$.

New modules:

```
U1:: module{listen nil;}{read x:sn;}
      {tx:tn; tx=-1;}{speak tx;}{write nil;}
V1:: module{listen tx:tn;}{read y:sn;}
      {null;}{speak tx;}{write nil;}
W1:: module{listen tx:tn;}{read z:sn}
      {null;}{speak nil;}{write z;}
```



High-level srv-programs

Contents:

- Generalities
- A glimpse on AGAPIA programming
- *Structured rv-programs*
- Conclusions

Basic characteristics of AGAPIA

- *space-time invariant*
- *high-level temporal data* structures
- *computation extends* both in *time* and *space*
- a *structural, compositional model*
- simple *operational semantics* (using *scenarios*)
- simple *relational semantics*

AGAPIA v0.1: Syntax

Syntax of AGAPIA v0.1:

Interfaces

$SST ::= nil \mid sn \mid sb$
 $\mid (SST \cup SST) \mid (SST, SST) \mid (SST)^*$
 $ST ::= (SST)$
 $\mid (ST \cup ST) \mid (ST; ST) \mid (ST;)^*$
 $STT ::= nil \mid tn \mid tb$
 $\mid (STT \cup STT) \mid (STT, STT) \mid (STT)^*$
 $TT ::= (STT)$
 $\mid (TT \cup TT) \mid (TT; TT) \mid (TT;)^*$

Expressions

$V ::= x : ST \mid x : TT$
 $\mid V(k) \mid V.k \mid V.[k] \mid V@k \mid V@[k]$
 $E ::= n \mid V \mid E + E \mid E * E \mid E - E \mid E / E$
 $B ::= b \mid V \mid B \&\& B \mid B || B \mid !B \mid E < E$

Programs

$W ::= null \mid new x : SST \mid new x : STT$
 $\mid x := E \mid if(B)\{W\}else\{W\}$
 $\mid W; W \mid while(B)\{W\}$
 $M ::= module\{listen x : STT\}\{read x : SST\}$
 $\{ W \}\{speak x : STT\}\{write x : SST\}$
 $P ::= null \mid M \mid if(B)\{P\}else\{P\}$
 $\mid P \% P \mid P \# P \mid P \$ P$
 $\mid while_{\perp}(B)\{P\} \mid while_{\neg s}(B)\{P\}$
 $\mid while_{\neg st}(B)\{P\}$



Example: Termination detection

Example: A program for distributed termination detection

```
P= I1# for_s(tid=0;tid<tm;tid++) {I2}#  
    $ while_st(!(token.col==white && token.pos==0)) {  
        for_s(tid=0;tid<tm;tid++) {R}}
```

where:

```
I1= module{listen nil}{read m}{  
    tm=m; token.col=black; token.pos=0;  
}{speak tm,tid,msg[ ],token(col,pos)}{write nil}
```

```
I2= module{listen tm,tid,msg[ ],token(col,pos)}  
    {read nil}{  
    id=tid; c=white; active=true; msg[id]=null;  
}{speak tm,tid,msg[ ],token(col,pos)}  
    {write id,c,active}
```

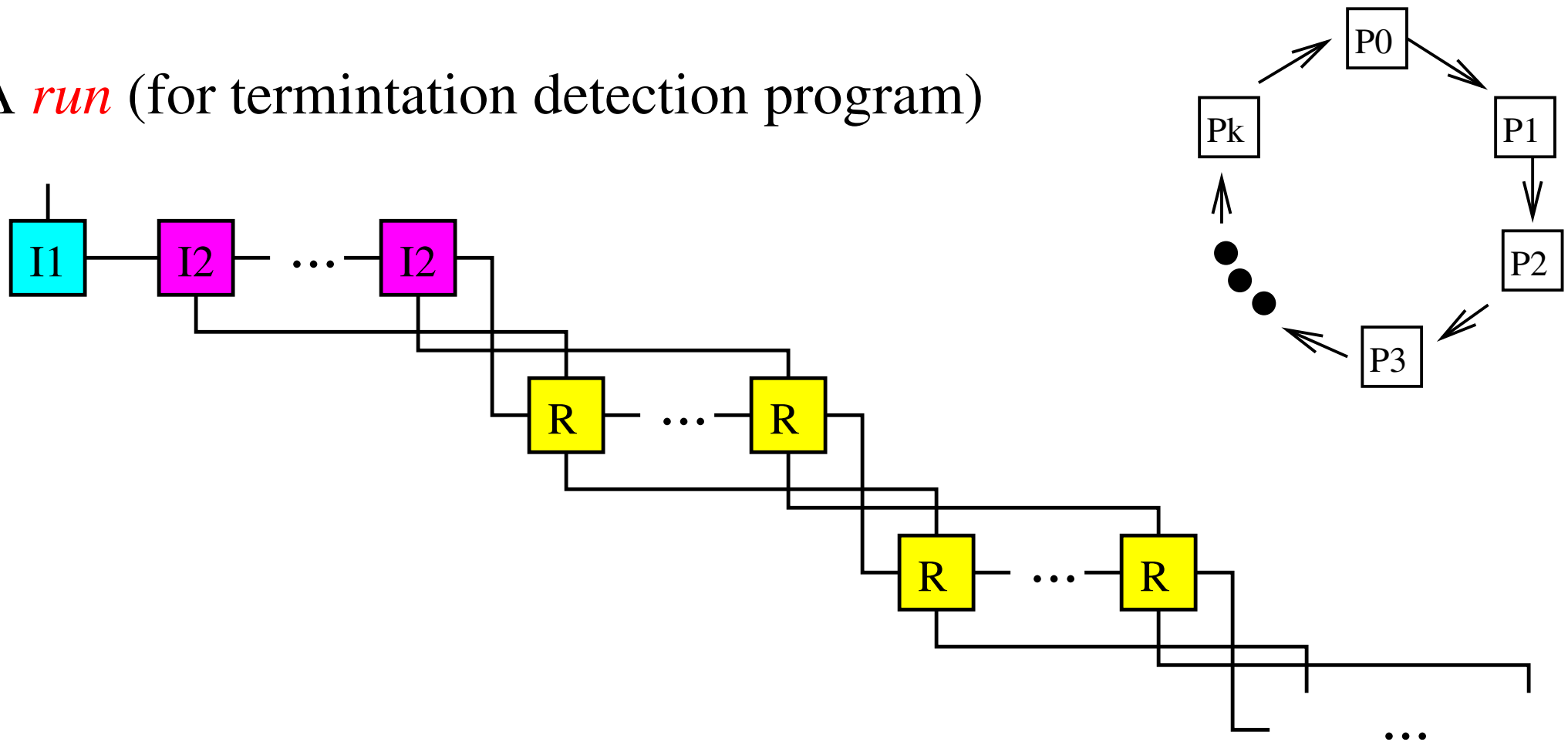


..Example: Termination detection

```
R=module{listen tm,tid,msg[ ],token(col,pos)}
{read id,c,active}{
  if(msg[id]!=emptyset){ //take my jobs
    msg[id]=emptyset;
    active=true;}
  if(active){ //execute code, send jobs, update color
    delay(random_time);
    r=random(tm-1);
    for(i=0;i<r;i++){ k=random(tm-1);
      if(k!=id){msg[k]=msg[k]∪{id}};
      if(k<id){c=black};}
    active=random(true,false);}
  if(!active && token.pos==id){ //termination
    if(id==0)token.col=white;
    if(id!=0 && c==black){token.col=black;c=white};
    token.pos=token.pos+1[mod tm];}
}{speak tm,tid,msg[ ],token(col,pos)}
{write id,c,active}
```


..Example: Termination detection

A *run* (for termination detection program)



```
I1# for_s(tid=0;tid<tm;tid++){I2}#  
$ while_st(!(token.col==white && token.pos==0)){  
  for_s(tid=0;tid<tm;tid++){R}}  
}
```

Syntax of AGAPIA v0.1:

Interface types

We use two special separators “,” and “;”

On spatial interfaces:

- “,” separates the types used in *a process*
- “;” separates the types used in *different processes*

On temporal interfaces:

- “,” separates the types used within *a transaction*
- “;” separates the types used in *different transactions*



Interface types

Simple spatial types are defined by:

$$SST ::= nil \mid sn \mid sb \mid (SST \cup SST) \mid (SST, SST) \mid (SST)^*$$

(“,” - associative with “nil” neutral element; “ \cup ” - associative)

Example:

$$((((sn)^*)^*, sb, (sn, sb, sn)^*)^*, (sb \cup sn))$$

represents the following data structure (for *a process*)

```
x:  struc1[], where
    struc1 = ( a:  Int[][],
               b:  Bool,
               c:  struc2[], where
                   struc2 = (p:Int, q:Bool, r:Int)
               ),
y:  Bool or Int
```

Simple temporal types — similar



Interface types

Spatial types are defined by::

$$ST ::= nil \mid (SST) \mid (ST \cup ST) \mid (ST; ST) \mid (ST;)^*$$

(“;” - associative with “nil” neutral element; “ \cup ” - associative)

Example:

$$((sn)^*)^*; nil; sb; ((sn)^*;)^*$$

represents *a collection of processes* (A, B, C, D) , where

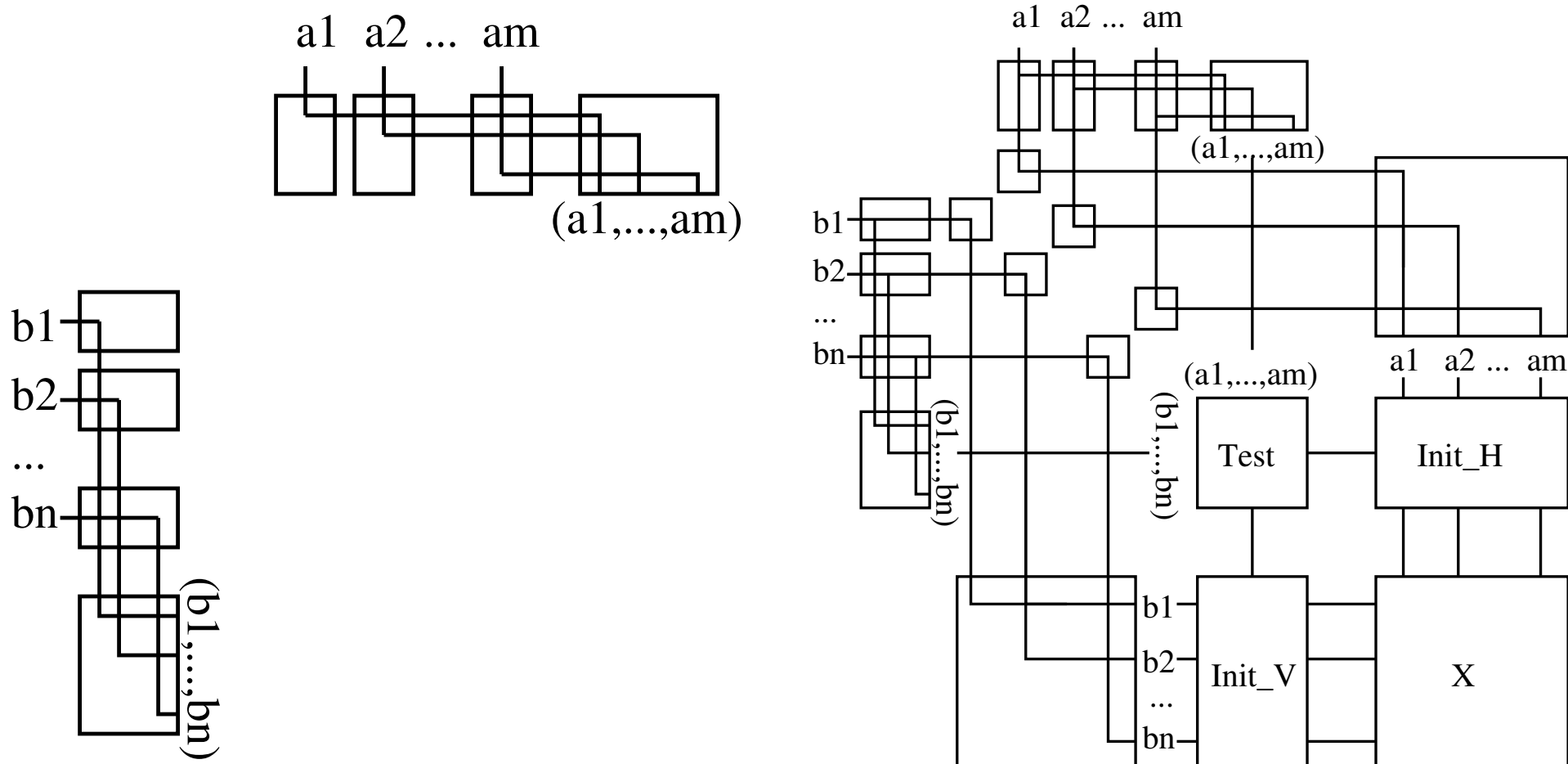
- A is *a process* using an array of arrays of integers
- B is *a process* with no starting spatial data
- C is *a process* using a boolean variable
- D is *an array of processes*, each process using an array of integers

Temporal types — similar

Interface types

Reshaping types

- interface types may be changed using special morphisms
- examples $(sn;)^* \mapsto (sn)^*$ and $(tn;)^* \mapsto (tn)^*$ (left)



Expressions

Variables

$$V ::= x : ST \mid x : TT \mid V(k) \mid V.k \mid V.[k] \mid V @ k \mid V @ [k]$$

Arithmetic expressions

$$E ::= n \mid V \mid E + E \mid E * E \mid E - E \mid E / E$$

Boolean expressions

$$B ::= b \mid V \mid B \& \& B \mid B || B \mid !B \mid E < E$$



..AGAPIA v0.1: Syntax

Programs

Simple while programs

$$\begin{aligned} W ::= & \text{null} \mid \text{new } x : SST \mid \text{new } x : STT \\ & \mid x := E \mid \text{if}(B)\{W\}\text{else}\{W\} \\ & \mid W;W \mid \text{while}(B)\{W\} \end{aligned}$$

Modules

$$\begin{aligned} M ::= & \text{module}\{\text{listen } x : STT\}\{\text{read } x : SST\} \\ & \{ W \}\{\text{speak } x : STT\}\{\text{write } x : SST\} \end{aligned}$$

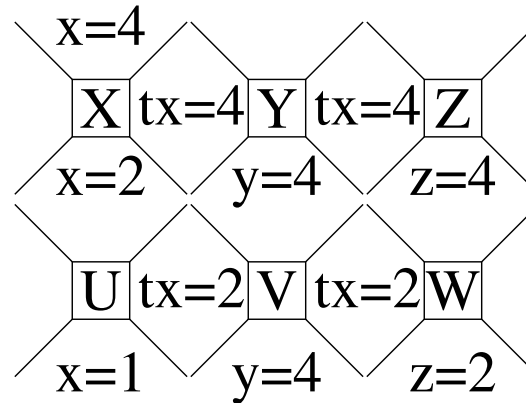
Agapia v0.1 programs

$$\begin{aligned} P ::= & \text{null} \mid M \mid \text{if}(B)\{P\}\text{else}\{P\} \\ & \mid P\%P \mid P\#P \mid P\$P \\ & \mid \text{while_t}(B)\{P\} \mid \text{while_s}(B)\{P\} \mid \text{while_st}(B)\{P\} \end{aligned}$$

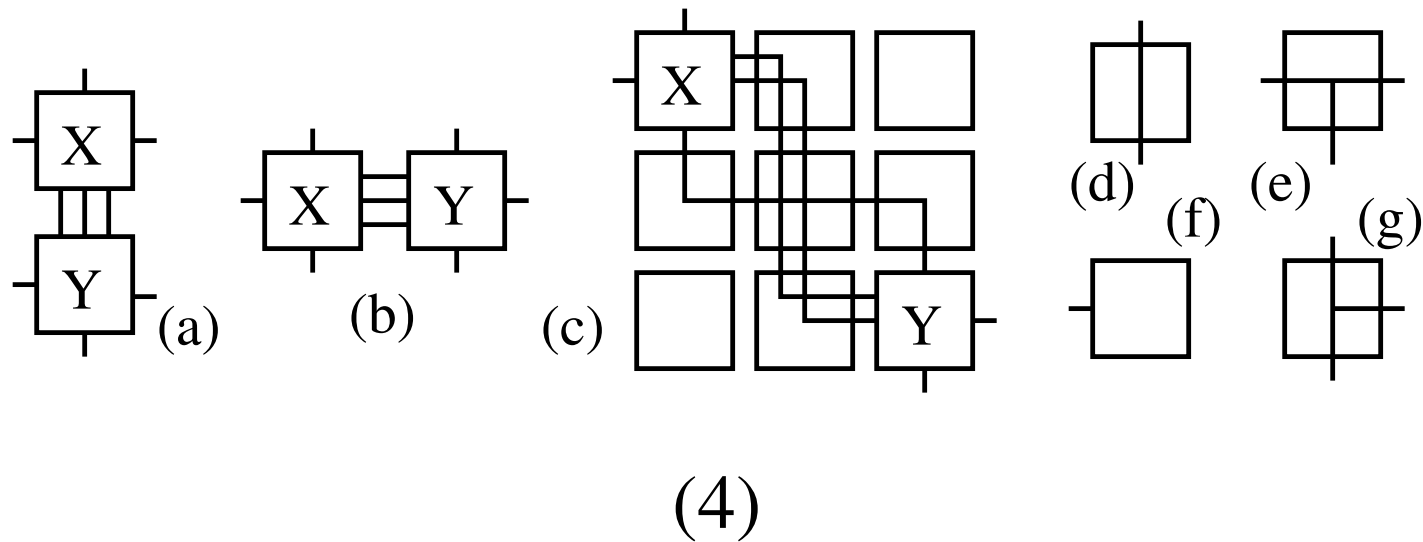


Scenarios

Srv-scenarios:



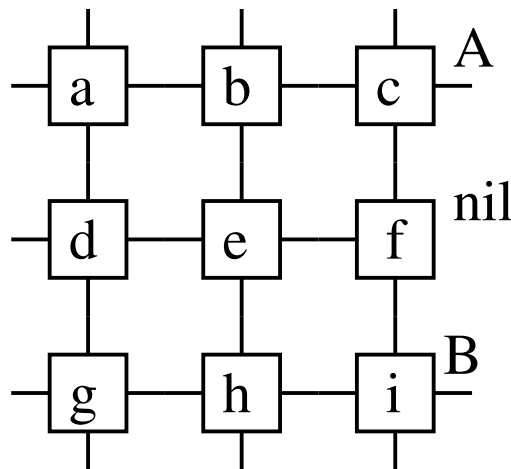
Srv-scenario operations:



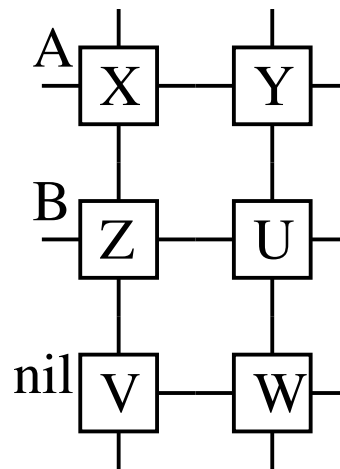
..Operations on srv-scenarios

..Srv-scenario operations:

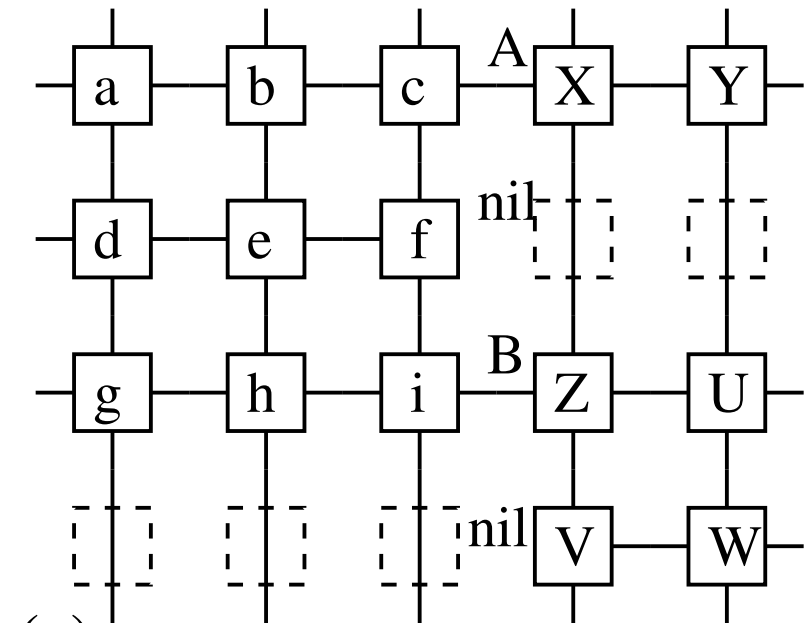
- Details for horizontal composition



(a)



(b)



(c)

- Similar procedures applies to the vertical and the diagonal srv-scenario compositions



High-level srv-programs

Contents:

- Generalities
- A glimpse on AGAPIA programming
- Structured rv-programs
- *High-level structured rv-programs*
- Conclusions



General modules

General modules:

- use general *ST*, *TT* types for its interfaces;
- the body is a usual while program, but now using variables accessing the components of these general type interfaces

Example:

```
900 module Sort {listen A1:tn; B1:tn}{read X1:sn; Y1:sn}
901 {
902   A2 = max(A1, B1);
903   B2 = min(A1, B1);
904   X2 = max(X1, Y1);
905   Y2 = min(X1, Y1);
906 }
907 {speak A2:tn; B2:tn}{write X2:sn; Y2:sn}
```



Encapsulating programs in modules

Encapsulating programs in modules with simple interfaces:

$ScatterS ::= \textbf{module } module_name \{listen\ nil\} \{read\ x : SST\}$
 $\{ body; \} \{speak\ nil\} \{write\ x : ST\}$

where *body* is a program using only $x := y$ assignments;

— it is used to “scatter” data from the spatial interface of a process to the spatial interface of a collection of processes

$ScatterT ::= \textbf{module } module_name \{listen\ x : STT\} \{read\ nil\}$
 $\{ body; \} \{speak\ x : TT\} \{write\ nil\}$

— similar for temporal interfaces

$GatherS ::= \textbf{module } module_name \{listen\ nil\} \{read\ x : ST\}$
 $\{ body; \} \{speak\ nil\} \{write\ x : SST\}$

— it is used to “gather” data from the spatial interface of a collection of processes to the spatial interface of a single process

$GatherT ::= \textbf{module } module_name \{listen\ x : TT\} \{read\ nil\}$
 $\{ body; \} \{speak\ x : STT\} \{write\ nil\}$

— similar for temporal interfaces



..Encapsulating programs in modules

Example:

```
75 module Scatter {listen nil}{read stemp, Pst[]}  
76 {  
77   south@1.stemp = north.stemp;  
78   south@1.Pst = north.Pst[0];  
79   for(i=1; i<length(Pst[]); i++)  
80     south@2@[i-1] = north.Pst[i];  
81 }  
82 {speak nil}{write (stemp,Pst); ((Pst;)[])}
```

Interfaces

$$SST ::= nil \mid sn \mid sb \\ \mid (SST \cup SST) \mid (SST, SST) \mid (SST)^*$$
$$ST ::= (SST) \mid (ST \cup ST) \mid (ST; ST) \mid (ST;)^*$$
$$STT ::= nil \mid tn \mid tb \\ \mid (STT \cup STT) \mid (STT, STT) \mid (STT)^*$$
$$TT ::= (STT) \mid (TT \cup TT) \mid (TT; TT) \mid (TT;)^*$$

Expressions

$$V ::= x : ST \mid x : TT \mid V(k) \\ \mid V.k \mid V.[k] \mid V@k \mid V@[k]$$
$$E ::= n \mid V \mid E + E \mid E * E \mid E - E \mid E / E$$
$$B ::= b \mid V \mid B \& \& B \mid B || B \mid !B \mid E < E$$

Programs

$$W ::= nil \mid new\ x : SST \mid new\ x : STT \\ \mid x := E \mid if(B)\{W\}else\{W\} \\ \mid W;W \mid while(B)\{W\}$$
$$M ::= \mathbf{module}\ module_name \{listen\ x : STT\} \{read\ x : SST\} \\ \{W\} \{speak\ x : STT\} \{write\ x : SST\} \\ \mid \mathbf{module}\ module_name \{listen\ x : STT\} \{read\ x : SST\} \\ \{ScatterT \# (ScatterS \% P \% GatherS) \# GatherT\} \\ \{speak\ x : STT\} \{write\ x : SST\}$$
$$P ::= nil \mid M \mid if(B)\{P\}else\{P\} \\ \mid P \% P \mid P \# P \mid P \$ P \\ \mid while_t(B)\{P\} \mid while_s(B)\{P\} \\ \mid while_st(B)\{P\}$$

..Programs

$ScatterS ::= \text{module } module_name \{listen\ nil\} \{read\ x : SST\}$
 $\{ body; \} \{speak\ nil\} \{write\ x : ST\}$

$ScatterT ::= \text{module } module_name \{listen\ x : STT\} \{read\ nil\}$
 $\{ body; \} \{speak\ x : TT\} \{write\ nil\}$

$GatherS ::= \text{module } module_name \{listen\ nil\} \{read\ x : ST\}$
 $\{ body; \} \{speak\ nil\} \{write\ x : SST\}$

$GatherT ::= \text{module } module_name \{listen\ x : TT\} \{read\ nil\}$
 $\{ body; \} \{speak\ x : STT\} \{write\ nil\}$

Note: The body part in these Scatter/Gather modules use only $x := y$ assignments, i.e., no real computation, only copy/delete data.



Example: Ring of dynamic processes

Case study: Communication in a cluster of dynamic processes

- a cluster of computers, each node having a set of running processes
- dynamic: allow new processes to join the set and old processes to leave it
- termination detection by extending a classical dual-pass ring termination detection protocol

..Example: Ring of dynamic processes

The Agapia v0.2 program

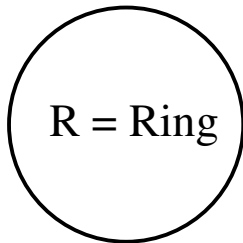
The full code of the termination protocol

```
1 // Notations:
2
3 temp = // temporary ring data passed between processes in the same ring
4 (ti, ts, msg_int[], msg_out, token(col,pos),
5  proc[], counter, rid, rc)
6 Temp = //temporary system data passed between rings
7 (trings, trid, proc[], msg_ext, bigtoken(col,ring))
8 stemp = (stl, stm, msg_int[], msg_out, stoken(col,pos),
9  sprc[], scounter, srid, src)
10 Pat = (pid,c,active) // the type of process state is (pid,c,active)
11 Sp = (stemp,Pst[]) // record states and temporal interface
12
13 // Main program and modules
14
15 main (listen nil)(read rings)
16 {
17   if for_a(trid = 0; trid < trings; trid++) (ID)#
18   while_at((bigtoken.col == white && bigtoken.ring == 0))
19   {
20     for_a(trid=0; trid<trings; trid++)(R)
21   }
22   (speak Temp)(write (Sp:))[]
23
24   module I1 (listen nil)(read rings)
25   {
26     trings = rings; bigtoken.col = black; bigtoken.ring = 0;
27     trid = 0;
28     for(i=0; i < rings; i++)
29       proc[i] = (0);
30     msg_ext = emptyset;
31   }
32   (speak Temp)(write nil)
33
34   module I2 (listen Temp)(read nil)
35   {
36     srid = trid; rc = white;
37     stm = stl = 0; scounter = 1;
38     stoken.col = black; stoken.pos = 0;
39     Pst[0].pid=0; Pst[0].c= white; Pst[0].active = true;
40     src = white;
41     smsg_out = emptyset;
42   }
43   (speak Temp)(write (stemp,Pst[0]))
44
45   module R (listen Temp)(read Sp)
46   {
47     Scatter
48   }
49   %
50   %
51   %
52   %
53   %
54   %
55   %
```

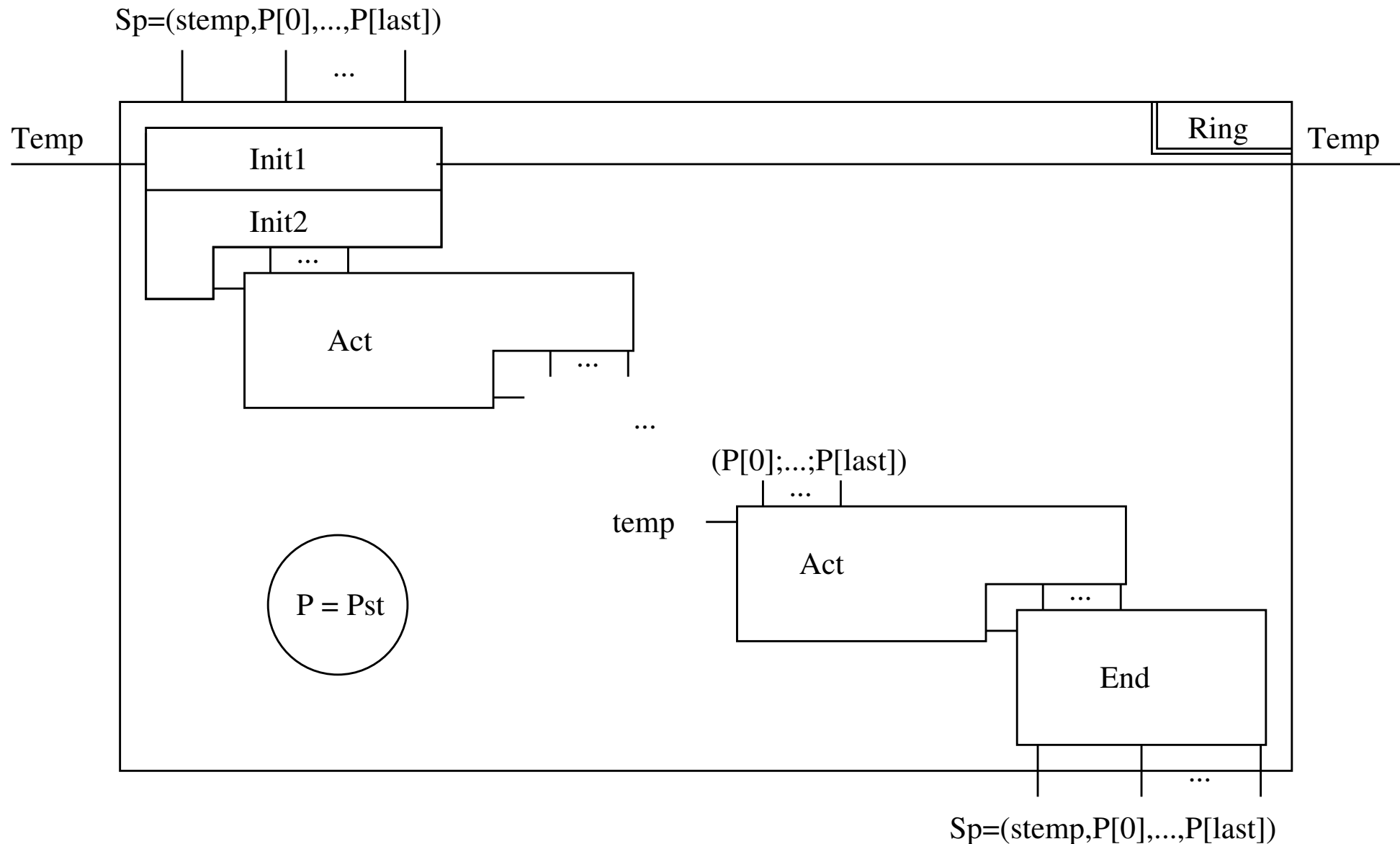
```
56   %
57   %
58   %
59   %
60   %
61   %
62   %
63   %
64   %
65   %
66   %
67   %
68   %
69   %
70   %
71   %
72   %
73   %
74   %
75   %
76   %
77   %
78   %
79   %
80   %
81   %
82   %
83   %
84   %
85   %
86   %
87   %
88   %
89   %
90   %
91   %
92   %
93   %
94   %
95   %
96   %
97   %
98   %
99   %
100  %
101  %
102  %
103  %
104  %
105  %
106  %
107  %
108  %
109  %
110  %
111  %
112  %
113  %
114  %
115  %
116  %
117  %
118  %
119  %
120  %
121  %
122  %
123  %
124  %
125  %
126  %
127  %
128  %
129  %
130  %
131  %
132  %
133  %
134  %
135  %
136  %
137  %
138  %
139  %
140  %
141  %
142  %
143  %
144  %
145  %
146  %
147  %
148  %
149  %
150  %
151  %
152  %
153  %
154  %
155  %
156  %
157  %
158  %
159  %
160  %
161  %
162  %
163  %
164  %
165  %
166  %
167  %
168  %
169  %
170  %
171  %
172  %
173  %
174  %
175  %
176  %
177  %
178  %
179  %
180  %
181  %
182  %
183  %
184  %
185  %
186  %
187  %
188  %
189  %
190  %
191  %
192  %
193  %
194  %
195  %
196  %
197  %
198  %
199  %
200  %
201  %
202  %
203  %
204  %
205  %
206  %
207  %
208  %
209  %
210  %
211  %
212  %
213  %
214  %
215  %
216  %
217  %
218  %
219  %
220  %
221  %
222  %
223  %
224  %
225  %
226  %
227  %
228  %
229  %
230  %
231  %
232  %
233  %
234  %
235  %
236  %
237  %
238  %
239  %
240  %
241  %
242  %
243  %
244  %
245  %
246  %
247  %
248  %
249  %
250  %
251  %
252  %
253  %
254  %
255  %
256  %
257  %
258  %
259  %
260  %
261  %
262  %
263  %
264  %
265  %
266  %
267  %
268  %
269  %
270  %
271  %
272  %
273  %
274  %
275  %
276  %
277  %
278  %
279  %
280  %
281  %
282  %
283  %
284  %
285  %
286  %
287  %
288  %
289  %
290  %
291  %
292  %
293  %
294  %
295  %
296  %
297  %
298  %
299  %
300  %
301  %
302  %
303  %
304  %
305  %
306  %
307  %
308  %
309  %
310  %
311  %
312  %
313  %
314  %
315  %
316  %
317  %
318  %
319  %
320  %
321  %
322  %
323  %
324  %
325  %
326  %
327  %
328  %
329  %
330  %
331  %
332  %
333  %
334  %
335  %
336  %
337  %
338  %
339  %
340  %
341  %
342  %
343  %
344  %
345  %
346  %
347  %
348  %
349  %
350  %
351  %
352  %
353  %
354  %
355  %
356  %
357  %
358  %
359  %
360  %
361  %
362  %
363  %
364  %
365  %
366  %
367  %
368  %
369  %
370  %
371  %
372  %
373  %
374  %
375  %
376  %
377  %
378  %
379  %
380  %
381  %
382  %
383  %
384  %
385  %
386  %
387  %
388  %
389  %
390  %
391  %
392  %
393  %
394  %
395  %
396  %
397  %
398  %
399  %
400  %
401  %
402  %
403  %
404  %
405  %
406  %
407  %
408  %
409  %
410  %
411  %
412  %
413  %
414  %
415  %
416  %
417  %
418  %
419  %
420  %
421  %
422  %
423  %
424  %
425  %
426  %
427  %
428  %
429  %
430  %
431  %
432  %
433  %
434  %
435  %
436  %
437  %
438  %
439  %
440  %
441  %
442  %
443  %
444  %
445  %
446  %
447  %
448  %
449  %
450  %
451  %
452  %
453  %
454  %
455  %
456  %
457  %
458  %
459  %
460  %
461  %
462  %
463  %
464  %
465  %
466  %
467  %
468  %
469  %
470  %
471  %
472  %
473  %
474  %
475  %
476  %
477  %
478  %
479  %
480  %
481  %
482  %
483  %
484  %
485  %
486  %
487  %
488  %
489  %
490  %
491  %
492  %
493  %
494  %
495  %
496  %
497  %
498  %
499  %
500  %
501  %
502  %
503  %
504  %
505  %
506  %
507  %
508  %
509  %
510  %
511  %
512  %
513  %
514  %
515  %
516  %
517  %
518  %
519  %
520  %
521  %
522  %
523  %
524  %
525  %
526  %
527  %
528  %
529  %
530  %
531  %
532  %
533  %
534  %
535  %
536  %
537  %
538  %
539  %
540  %
541  %
542  %
543  %
544  %
545  %
546  %
547  %
548  %
549  %
550  %
551  %
552  %
553  %
554  %
555  %
556  %
557  %
558  %
559  %
560  %
561  %
562  %
563  %
564  %
565  %
566  %
567  %
568  %
569  %
570  %
571  %
572  %
573  %
574  %
575  %
576  %
577  %
578  %
579  %
580  %
581  %
582  %
583  %
584  %
585  %
586  %
587  %
588  %
589  %
590  %
591  %
592  %
593  %
594  %
595  %
596  %
597  %
598  %
599  %
600  %
601  %
602  %
603  %
604  %
605  %
606  %
607  %
608  %
609  %
610  %
611  %
612  %
613  %
614  %
615  %
616  %
617  %
618  %
619  %
620  %
621  %
622  %
623  %
624  %
625  %
626  %
627  %
628  %
629  %
630  %
631  %
632  %
633  %
634  %
635  %
636  %
637  %
638  %
639  %
640  %
641  %
642  %
643  %
644  %
645  %
646  %
647  %
648  %
649  %
650  %
651  %
652  %
653  %
654  %
655  %
656  %
657  %
658  %
659  %
660  %
661  %
662  %
663  %
664  %
665  %
666  %
667  %
668  %
669  %
670  %
671  %
672  %
673  %
674  %
675  %
676  %
677  %
678  %
679  %
680  %
681  %
682  %
683  %
684  %
685  %
686  %
687  %
688  %
689  %
690  %
691  %
692  %
693  %
694  %
695  %
696  %
697  %
698  %
699  %
700  %
701  %
702  %
703  %
704  %
705  %
706  %
707  %
708  %
709  %
710  %
711  %
712  %
713  %
714  %
715  %
716  %
717  %
718  %
719  %
720  %
721  %
722  %
723  %
724  %
725  %
726  %
727  %
728  %
729  %
730  %
731  %
732  %
733  %
734  %
735  %
736  %
737  %
738  %
739  %
740  %
741  %
742  %
743  %
744  %
745  %
746  %
747  %
748  %
749  %
750  %
751  %
752  %
753  %
754  %
755  %
756  %
757  %
758  %
759  %
760  %
761  %
762  %
763  %
764  %
765  %
766  %
767  %
768  %
769  %
770  %
771  %
772  %
773  %
774  %
775  %
776  %
777  %
778  %
779  %
780  %
781  %
782  %
783  %
784  %
785  %
786  %
787  %
788  %
789  %
790  %
791  %
792  %
793  %
794  %
795  %
796  %
797  %
798  %
799  %
800  %
801  %
802  %
803  %
804  %
805  %
806  %
807  %
808  %
809  %
810  %
811  %
812  %
813  %
814  %
815  %
816  %
817  %
818  %
819  %
820  %
821  %
822  %
823  %
824  %
825  %
826  %
827  %
828  %
829  %
830  %
831  %
832  %
833  %
834  %
835  %
836  %
837  %
838  %
839  %
840  %
841  %
842  %
843  %
844  %
845  %
846  %
847  %
848  %
849  %
850  %
851  %
852  %
853  %
854  %
855  %
856  %
857  %
858  %
859  %
860  %
861  %
862  %
863  %
864  %
865  %
866  %
867  %
868  %
869  %
870  %
871  %
872  %
873  %
874  %
875  %
876  %
877  %
878  %
879  %
880  %
881  %
882  %
883  %
884  %
885  %
886  %
887  %
888  %
889  %
890  %
891  %
892  %
893  %
894  %
895  %
896  %
897  %
898  %
899  %
900  %
901  %
902  %
903  %
904  %
905  %
906  %
907  %
908  %
909  %
910  %
911  %
912  %
913  %
914  %
915  %
916  %
917  %
918  %
919  %
920  %
921  %
922  %
923  %
924  %
925  %
926  %
927  %
928  %
929  %
930  %
931  %
932  %
933  %
934  %
935  %
936  %
937  %
938  %
939  %
940  %
941  %
942  %
943  %
944  %
945  %
946  %
947  %
948  %
949  %
950  %
951  %
952  %
953  %
954  %
955  %
956  %
957  %
958  %
959  %
960  %
961  %
962  %
963  %
964  %
965  %
966  %
967  %
968  %
969  %
970  %
971  %
972  %
973  %
974  %
975  %
976  %
977  %
978  %
979  %
980  %
981  %
982  %
983  %
984  %
985  %
986  %
987  %
988  %
989  %
990  %
991  %
992  %
993  %
994  %
995  %
996  %
997  %
998  %
999  %
1000 %
```

```
113 msg_ext = msg_ext Union msg_out;
114
115 // keep big-token, if necessary
116 if(bigtoken.ring == trid)
117 {
118   if(stoken.col == white && stoken.pos == 0)
119   {
120     if (trid == 0)
121       bigtoken.col = white;
122     if (trid != 0 && src == black)
123     {
124       bigtoken.col = black;
125       src = white;
126     }
127     bigtoken.ring = bigtoken.ring + 1 [mod trings];
128   }
129   }
130   ti = stl;
131   ts = stm;
132 }
133 (speak Temp, ti, ts)(write stemp, Pst)
134
135 module RInit12 (listen nil)(read stemp, Pst)
136 {
137   temp = stemp; // vector assignment
138   // choose a random number of communication rounds in the ring
139   stemp = 0;
140   maxsteps = random () + 1;
141 }
142 (speak temp, stemp, maxsteps)(write Pst)
143
144 module RInit13 (listen temp)(read Pst)
145 {
146   stemp = temp; // vector assignment
147 }
148 (speak nil)(write stemp, Pst)
149
150 module D1 (listen temp)(read nil)
151 {
152   tld = length(proc[trid]); // the number of current processes
153   tnew = random(); // choose how many new processes to add
154   ts = tld + tnew; // update ts
155 }
156 (speak temp, tld)(write nil)
157
158 module I11 (listen Temp)(read Pst) { nil; } (speak Temp)(write Pst)
159 module I12 (listen Temp)(read Pst) { nil; } (speak Temp)(write Pst)
160 module I13 (listen nil)(read Pst) { nil; } (speak nil)(write Pst)
161
162 module NewProc (listen temp)(read Pst)
163 {
164   pid = counter; // pick the next free id from the list
165   counter++; // increase the counter
166   proc[trid] = proc[trid] Union {pid};
167   c = white;
168   active = true;
169 }
170 }
```

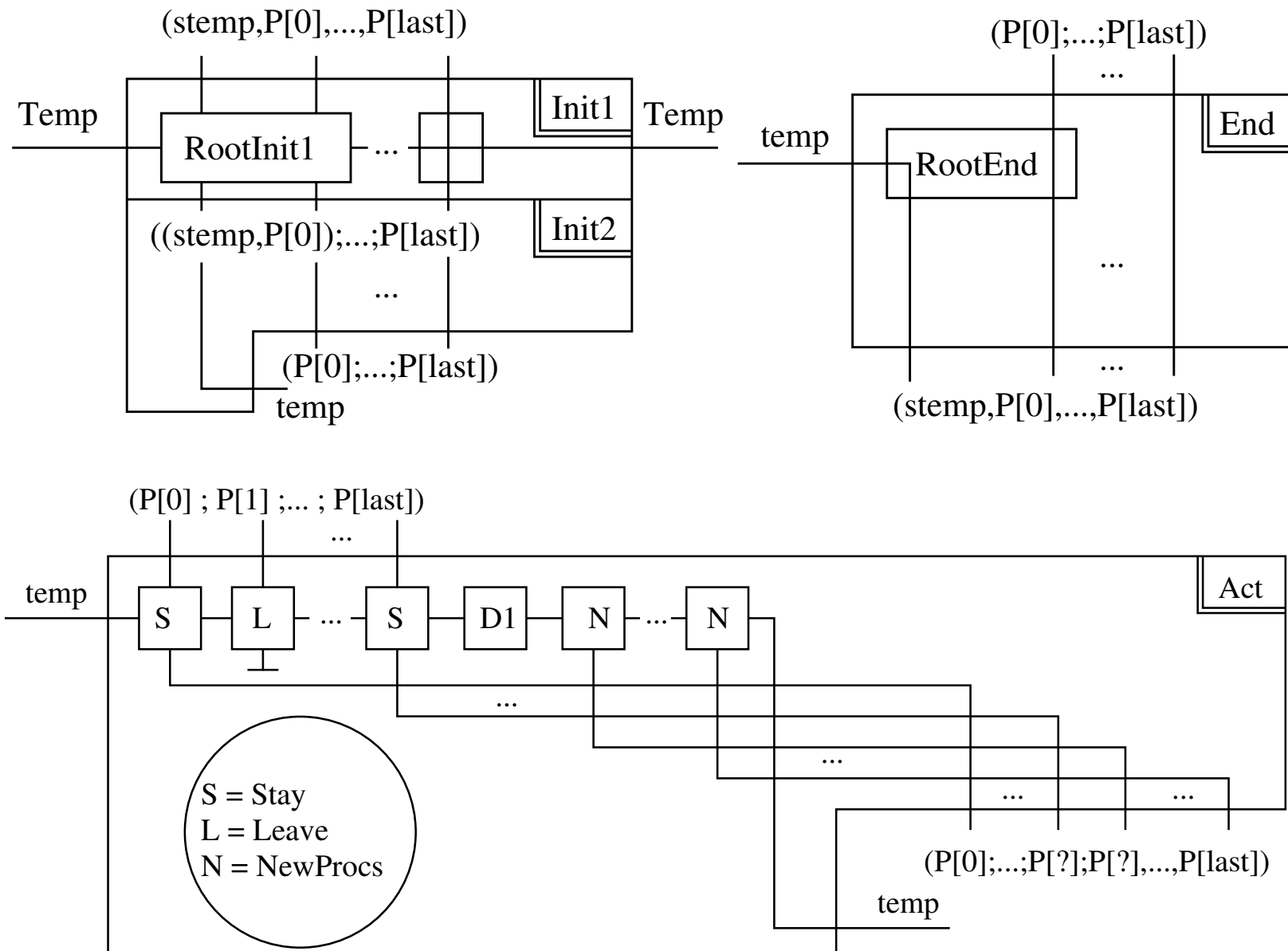
```
170 (speak temp)(write Pst)
171
172 module Stay (listen temp)(read Pst)
173 {
174   if(msg_int[pid] != emptyset) // take my job
175   {
176     msg_int[pid] = emptyset;
177     active = true;
178   }
179   if(active)
180   {
181     delay(random,time); // execute
182     r = random(ts - 1); // choose how many messages to send
183     for(i=0; i<r; i++)
184     {
185       q = random(trings - 1); // choose the ring
186       w = random in proc[q]; // choose the process
187       if(q != rid) // for a process in another ring
188         msg_out = msg_out Union {(rid,pid,q,w)};
189       else if (w != pid) // for a process in the same ring
190         msg_int[q] = msg_int[q] Union {(q,w)};
191       if(q < trid) // color the ring in black
192         rc = black;
193       if(q == rid && w < pid) // color the process in black
194         c = black;
195     }
196     active = random(true,false);
197   }
198   if(!active && token.pos == pid) // termination
199   {
200     if(pid == 0) token.col = white;
201     if(pid != 0 && c == black)
202     {
203       token.col = black; c = white;
204     }
205     token.pos = next(proc[rid],token.pos); // pass the token
206   }
207 }
208 (speak temp)(write Pst)
209
210 module Leave (listen temp)(read Pst)
211 {
212   token.pos = next(proc[rid],token.pos); // pass the token
213   proc[rid] = proc[rid] - pid; // update process list
214 }
215 (speak temp)(write nil)
```



..Example: Ring of dynamic processes



..Example: Ring of dynamic processes





High-level srv-programs

Contents:

- Generalities
- A glimpse on AGAPIA programming
- Structured rv-programs
- High-level structured rv-programs
- *Conclusions*