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:

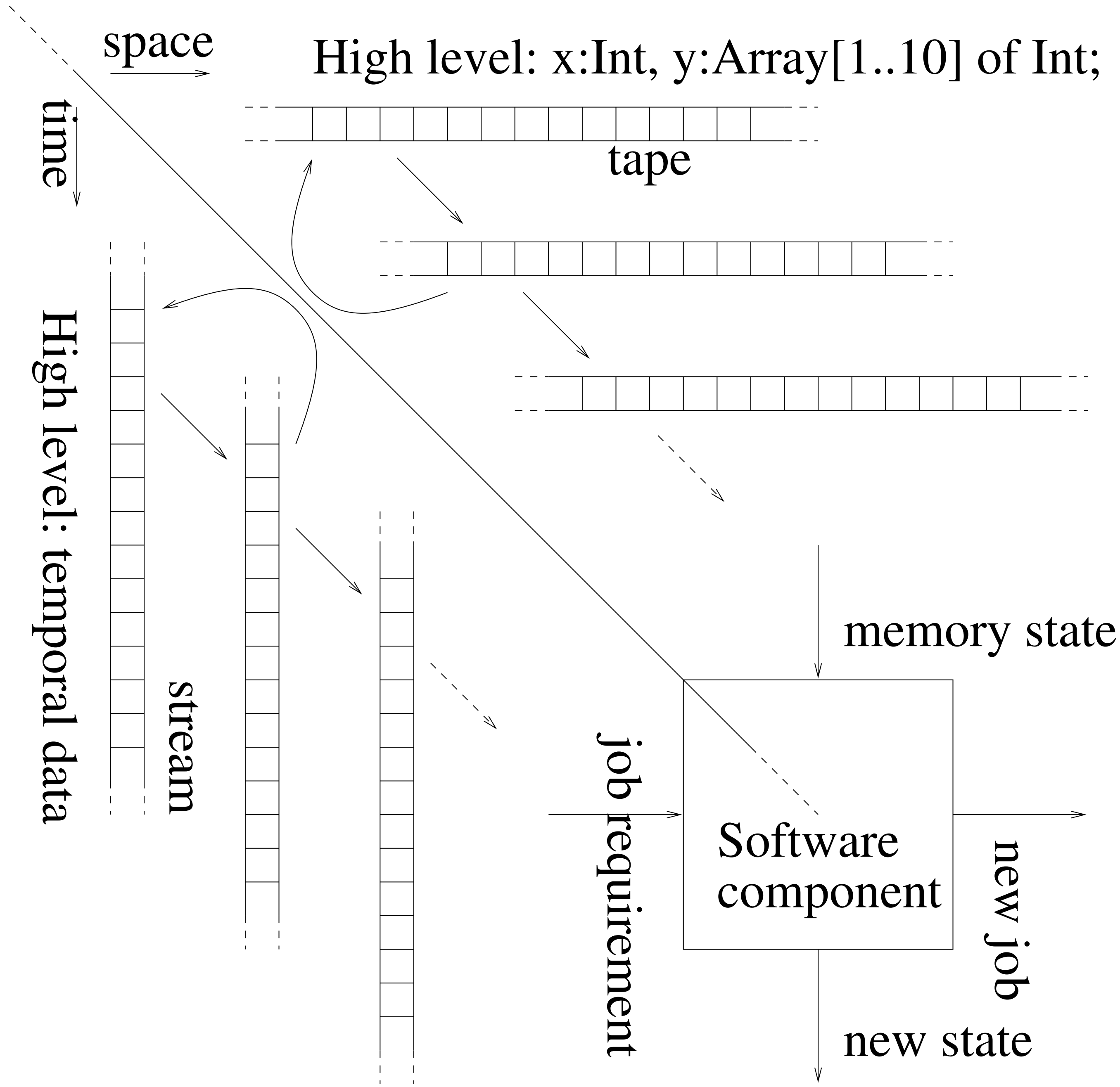
History

History

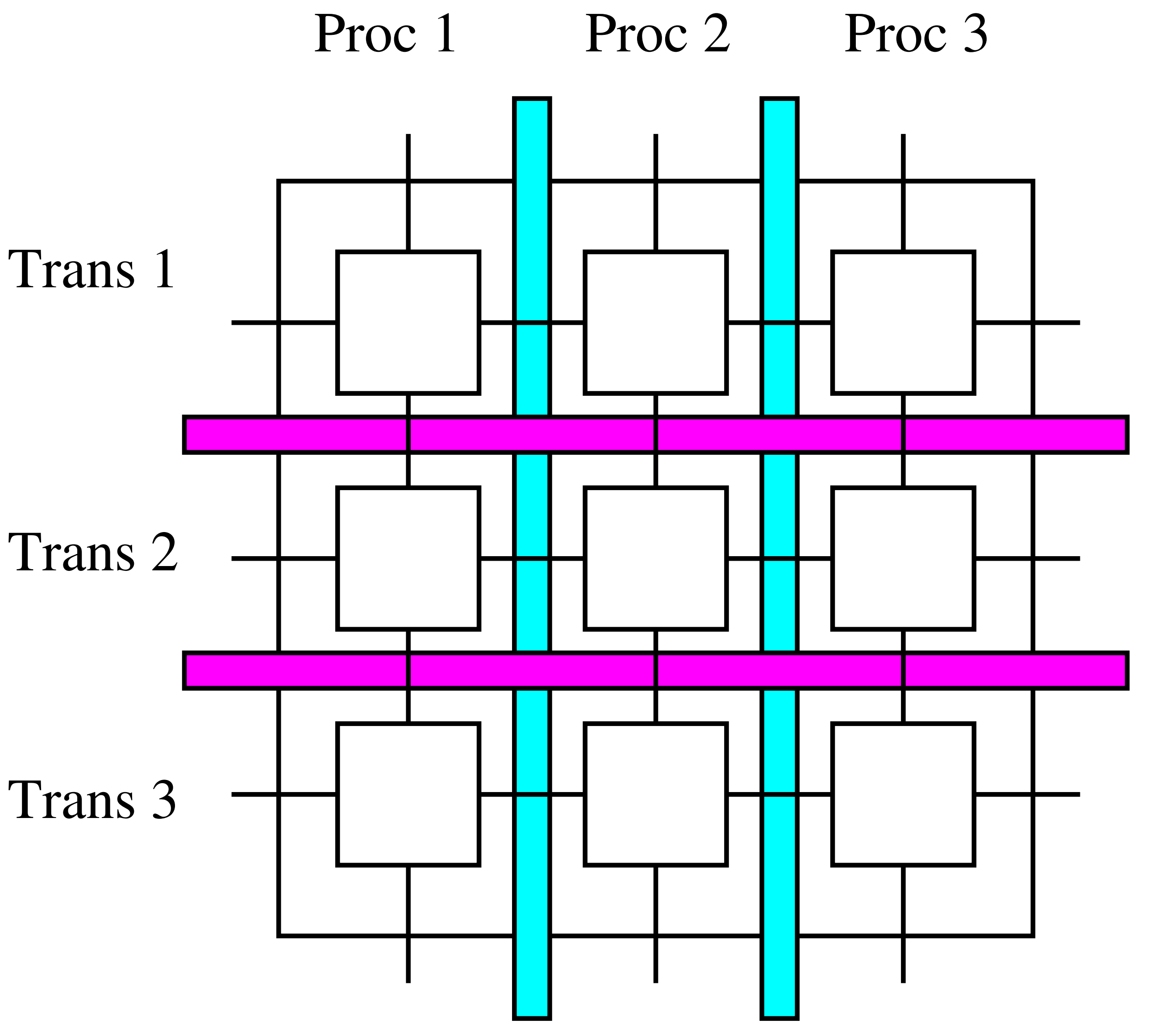
- *space-time duality "thesis"*
  - Stefanescu, *Network algebra*, Springer 2000
- *finite interactive systems*
  - Stefanescu, Marktoberdorf Summer School 2001
- *rv-systems* (interactive systems with registers and voices)
  - Stefanescu, NUS, Singapore, summer 2004
- *structured rv-systems*
  - Stefanescu,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

ST-Dual picture

ST-Dual picture



Processes and transactions



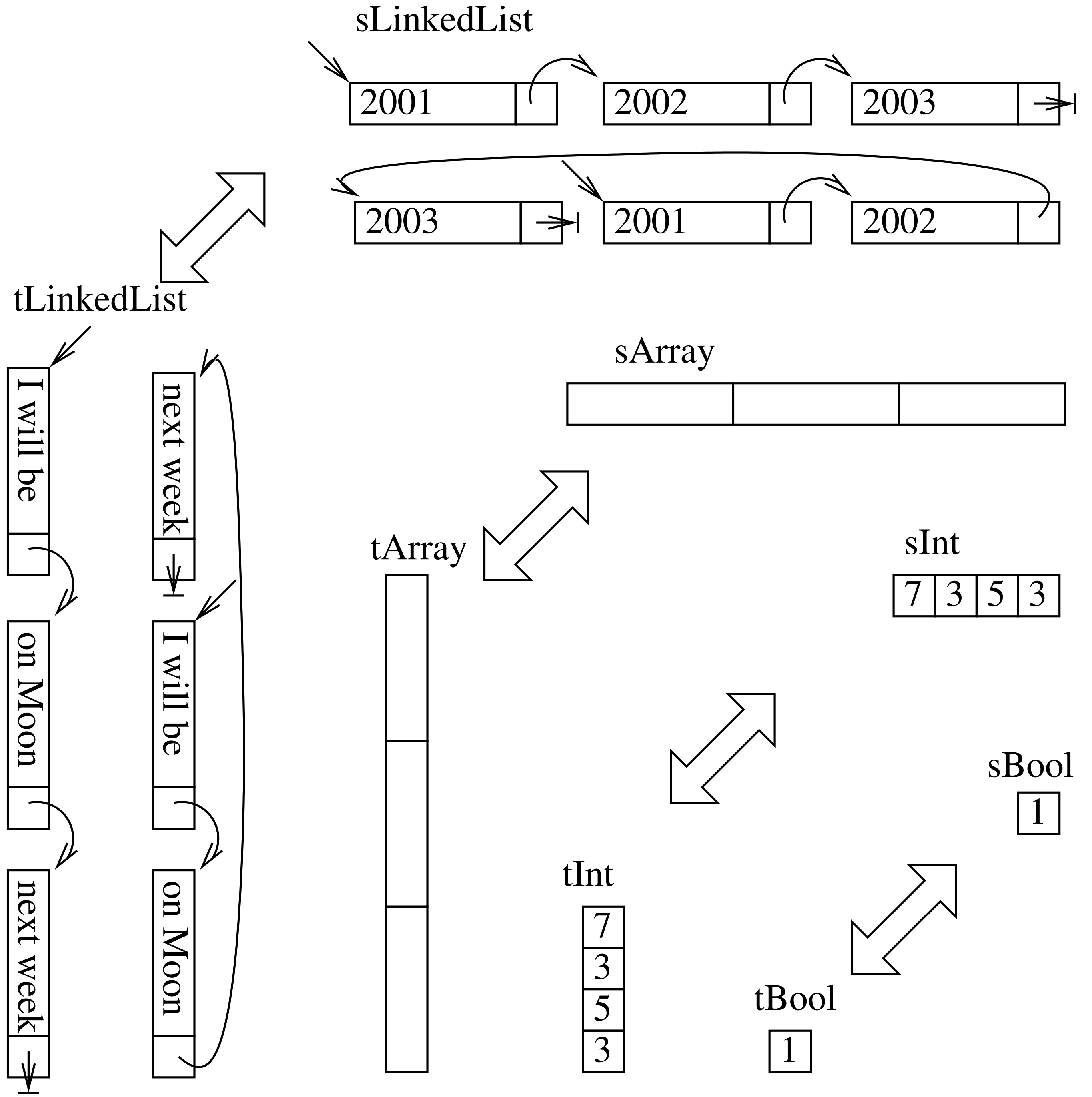
High level temporal structures

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:

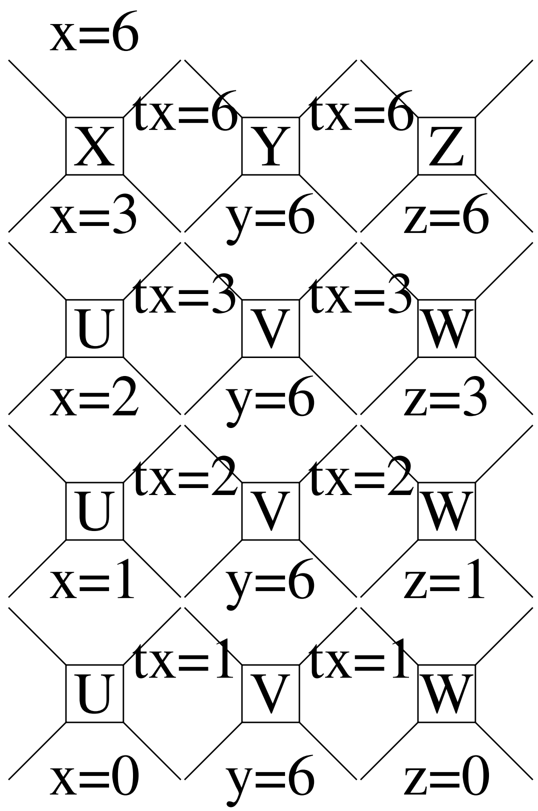
3 components C_x, C_y, C_z where:

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

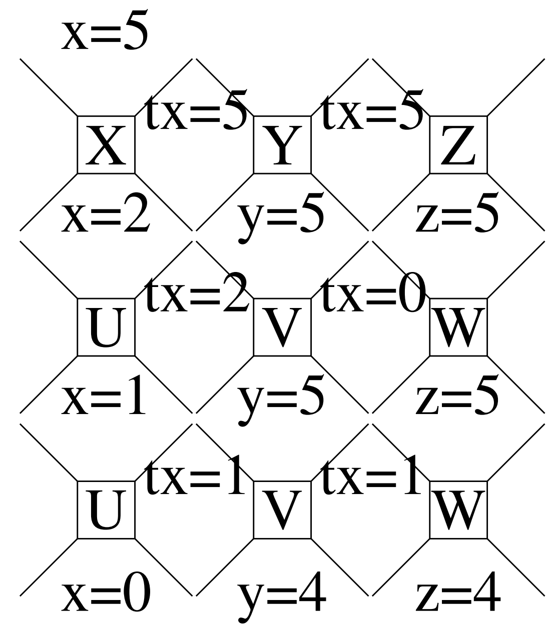
These components are composed *horizontally*. The global input-output specification: *if the input number in C_x is n, then the output number in C_z is 0 iff n is perfect*.

..Srv-programs for perfect numbers

Two scenarios for perfect numbers:



(a)



(b)

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

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

AGAPIA

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

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

Expressions

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

Programs

$W ::= null \mid new\ x : SST \mid new\ x : STT$
$\quad \mid x := E \mid if(B)\{W\}else\{W\}$
$\quad \mid W; W \mid while(B)\{W\}$
$M ::= module\{listen\ x : STT\}\{read\ x : SST\}$
$\quad \{\ W\ \}\{speak\ x : STT\}\{write\ x : SST\}$
$P ::= null \mid M \mid if(B)\{P\}else\{P\}$
$\quad \mid P\%P \mid P\#P \mid P\$P$
$\quad \mid while\_t(B)\{P\} \mid while\_s(B)\{P\}$
$\quad \mid while\_st(B)\{P\}$

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:

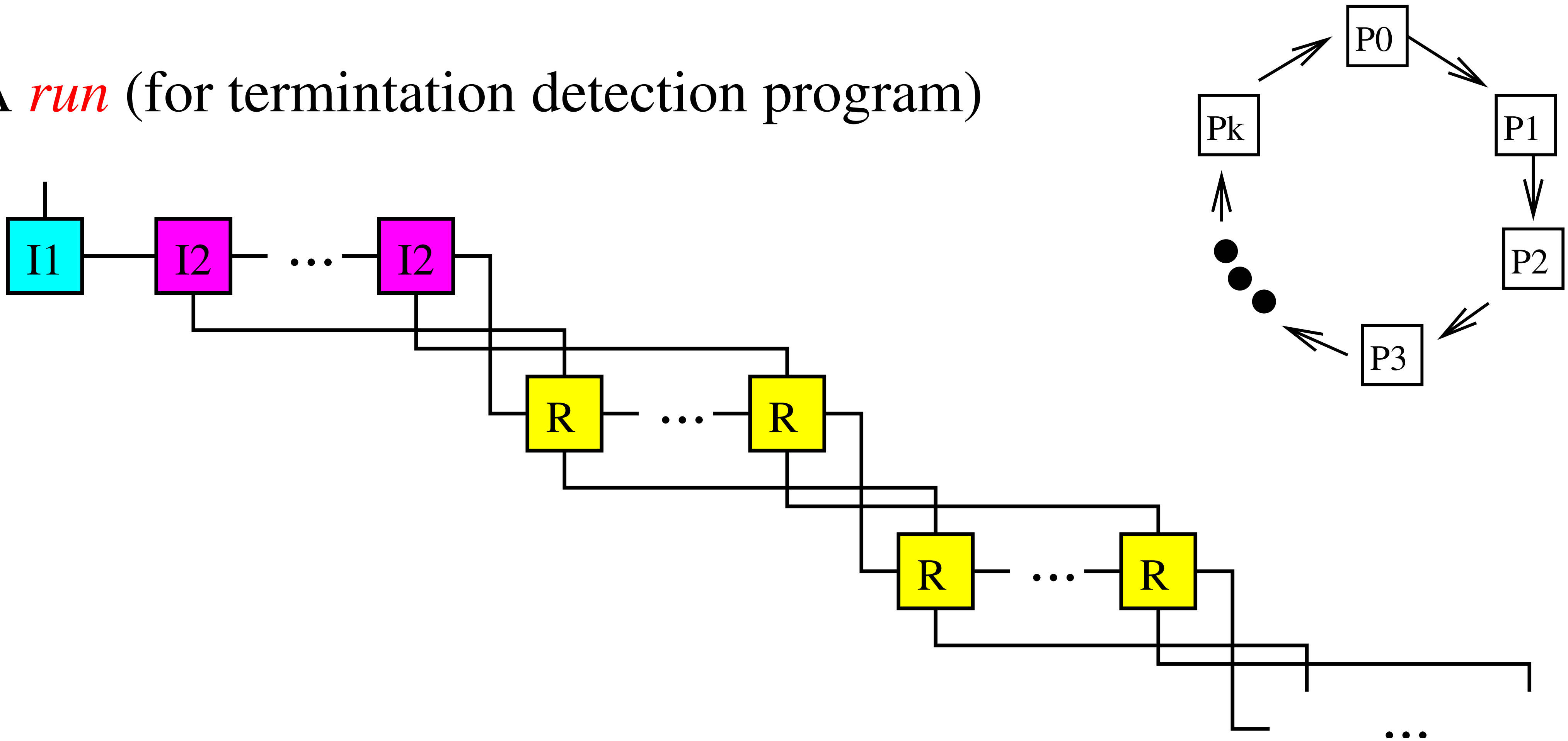
```
I1= module{listen nil}{read m}{
    tm=m; token.col=black; token.pos=0;
    }{speak tm,tid,msg[ ],token(col,pos)}{write nil}

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 termintation 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}}
```

AGAPIA v0.1: Syntax

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; “∪” - associative)

Example:

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

represents the folowing 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:

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

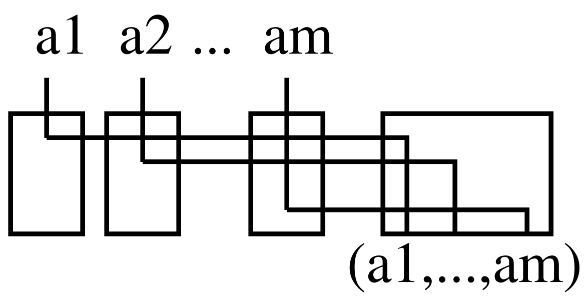
represents *a collection of processes* (A,B,C,D), where

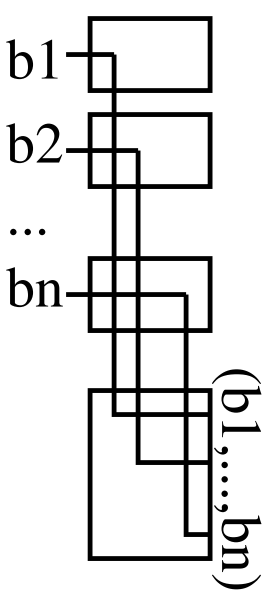
- A is *a process* using an array of arrays of integers
- B is *a process* with no starting spatial data
- C is *a process* using a boolean variable
- D is *an array of processes*, each process using an array of integers

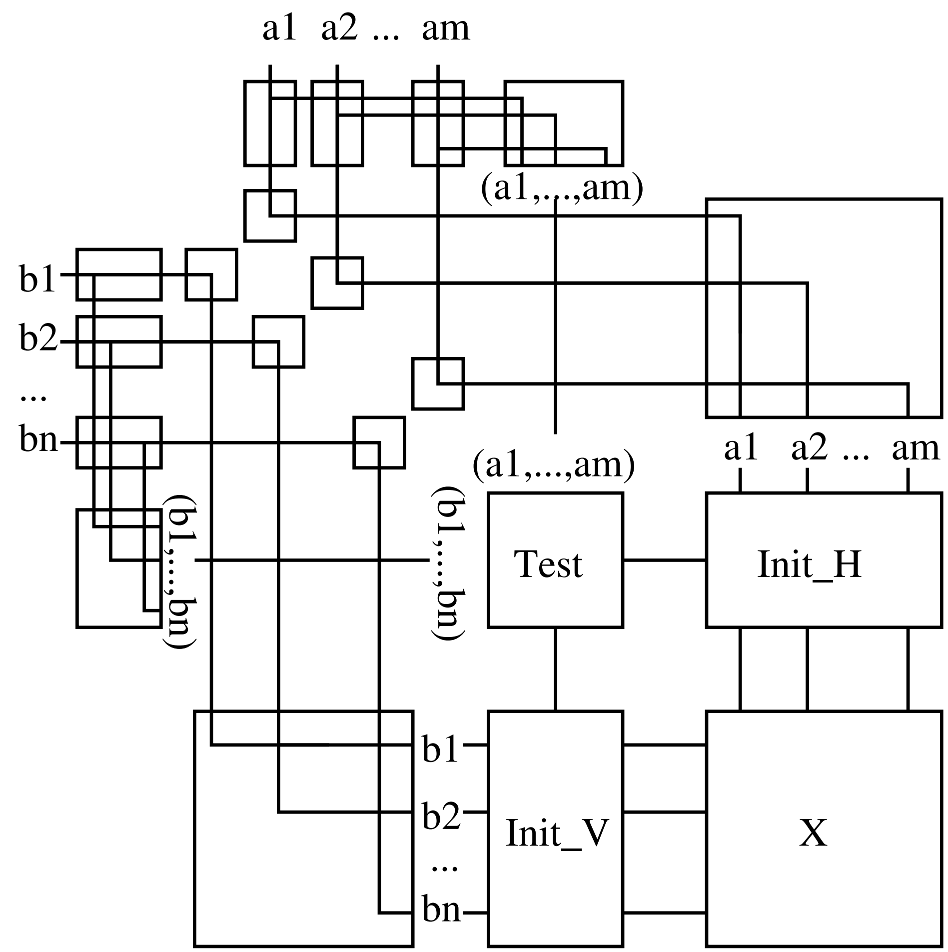
Temporal types — similar

Reshaping types

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







AGAPIA v0.1: Syntax

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

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

Modules

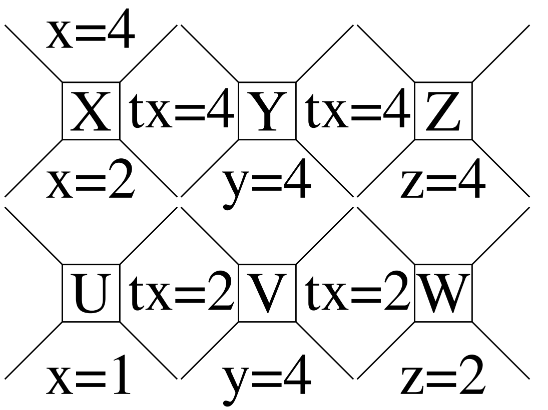
$$\begin{aligned} M ::= \; & module\{listen\ x : STT\}\{read\ x : SST\} \\ & \qquad \{\ W\ \}\{speak\ x : STT\}\{write\ x : SST\} \end{aligned}$$

Agapia v0.1 programs

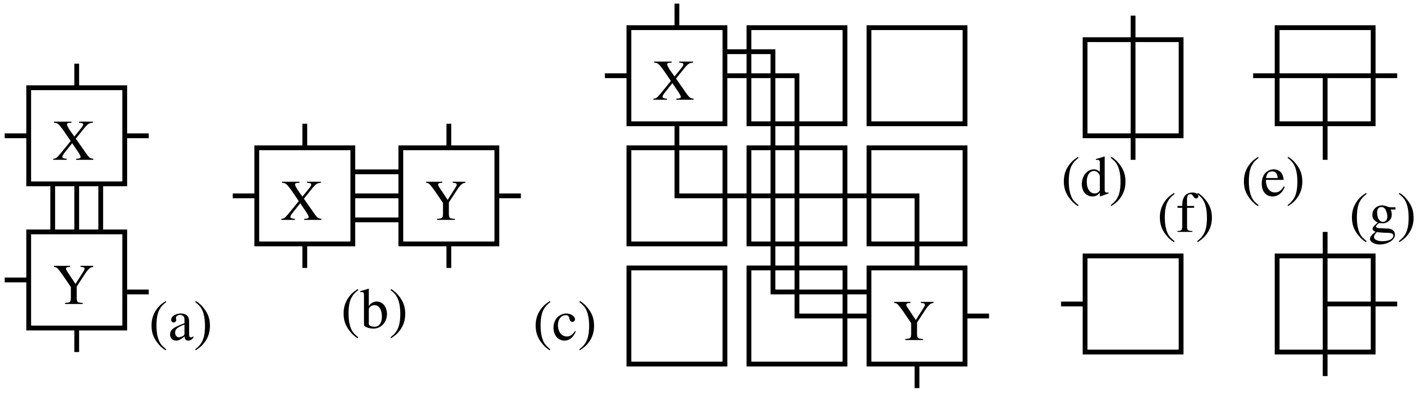
$$\begin{aligned} P ::= \; & null \mid M \mid if(B)\{P\}else\{P\} \\ & \mid P\%P \mid P\#P \mid P\$P \\ & \mid while\_t(B)\{P\} \mid while\_s(B)\{P\} \mid while\_st(B)\{P\} \end{aligned}$$

Scenarios

Srv-scenarios:



Srv-scenario operations:

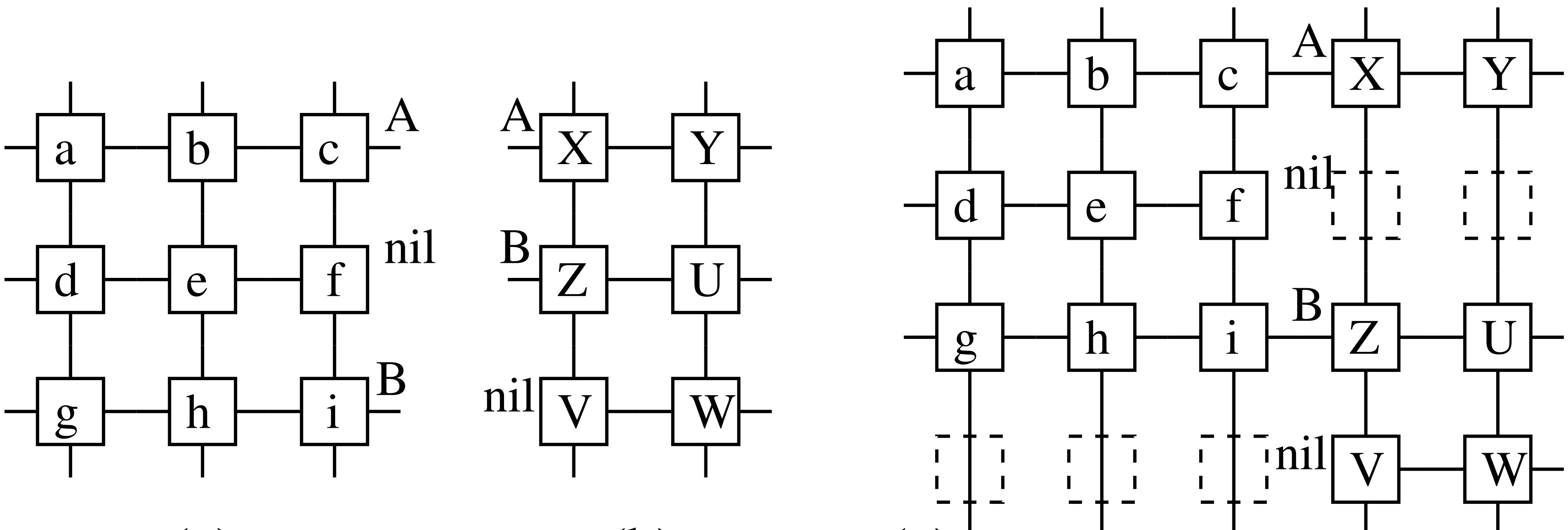


(4)

..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:

```
module Sort {listen A1:tn; B1:tn}{read X1:sn; Y1:sn}
{
 A2 = max(A1,B1);
 B2 = min(A1,B1);
 X2 = max(X1,Y1);
 Y2 = min(X1,Y1);
}
{speak A2:tn; B2:tn}{write X2:sn; Y2:sn}
```

Encapsulating programs in modules

Encapsulating programs in modules with simple interfaces:

$ScatterS ::=$ **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

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

— similar for temporal interfaces

$GatherS ::=$ **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

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

— similar for temporal interfaces

..Encapsulating programs in modules

Example:

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

Agapia v0.2

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

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

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

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

..Agapia v0.2

Programs

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

$$\begin{aligned}
M ::= \;& \textbf{module}\ module_name\{listen\ x : STT\}\{read\ x : SST\} \\
& \quad \{\ W\ \}\{speak\ x : STT\}\{write\ x : SST\} \\
& \mid \textbf{module}\ module_name\{listen\ x : STT\}\{read\ x : SST\} \\
& \quad \{\ ScatterT\ \#\ (ScatterS\ \%\ \ P\ \ \%\ GatherS)\ \#\ GatherT\ \} \\
& \quad \{speak\ x : STT\}\{write\ x : SST\}
\end{aligned}$$

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

..Agapia v0.2

..Programs

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

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

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

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

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

```
// Notations:

temp = // temporary ring data passed between processes in the same ring
       (ti, tm, msg_int[], msg_out, token(col,pos),
          procs[], counter, rid, rc)
Temp = //temporary system data passed between rings
        (trings, trid, procs[], msg_ext, bigtoken(col,ring))
stemp = (sti, stm, smsg_int[], smsg_out, stoken(col,pos),
          sprocs[], scounter, srid, src)
Pst = (pid,c,active) // the type of process state is (pid,c,active)
Sp = (stemp,Pst[]) // record states and temporal interface

// Main program and modules

main (listen nil)(read rings)
{
  I1# for_s(trid = 0; trid < trings; trid++) (I2)#
#
  while_st(!(bigtoken.col == white && bigtoken.ring == 0))
  {
    for_s(trid=0; trid<trings; trid++)(R)
  }
}
(speak Temp)(write (Sp;)[])

module I1 (listen nil)(read rings)
{
  trings = rings; bigtoken.col = black; bigtoken.ring = 0;
  trid = 0;
  for(i=0; i < rings; i++)
    procs[i] = (0);
  msg_ext = emptyset;
}
(speak Temp)(write nil)

module I2 (listen Temp)(read nil)
{
  srid = trid; rc = white;
  stm = sti = 0; scounter = 1;
  stoken.col = black; stoken.pos = 0;
  Pst[0].pid=0; Pst[0].c= white; Pst[0].active = true;
  src = white;
  smsg_out = emptyset;
}
(speak Temp)(write (stemp,Pst[0]))

module R (listen Temp)(read Sp)
{
  Scatter
%
  RootInit1# for_s(ti=1; ti<tm; ti++) (Id1)#
%
  {
    (RootInit2# for_s(ti=1; ti<tm; ti++) (Id2)#)$
    for_st(step = 0; !(token.col == white && token.pos != 0)
```

1

```
                     && step < maxsteps; step++ )
    {
      for_s(ti=0; ti<tm; ti++)
      {
        if(pid == 0 || random((leave, stay)) == stay)
          Stay  // the process stay in the ring
        else
          Leave  // the process will leave the ring
      }# D1# for_s(ti=told; ti<tm; ti++)(NewProc)#
    }#
    RootEnd# for_s(ti=1; ti<tm; ti++) (Id3)#
  }
%
  Gather
}
(speak Temp)(write Sp)

// Scatter and Gather modules

module Scatter (listen nil)(read stemp, Pst[])
{
  south@1.stemp = north.stemp;
  south@1.Pst = north.Pst[0];
  for(i=1; i<length(Pst[]); i++)
    south@2@(i-1) = north.Pst[i];
}
(speak nil)(write (stemp,Pst); ((Pst;)[]))

module Gather (listen nil)(read (stemp,Pst); ((Pst;)[]))
{
  south.stemp = north@1.stemp;
  south.Pst[0] = north@1.Pst;
  for(i=0; i<length(north@2); i++)
    south.Pst[i+1] = north@2.Pst@[i];
}
(speak nil)(write stemp, Pst[])

module RootInit1 (listen Temp)(read stemp, Pst)
{
  //update process list for this ring
  Temp.procs[trid] = stemp.sprocs[trid];

  //update process list from other rings
  stemp.sprocs=Temp.procs; //vector assignment

  // get the messages received for this rings and discard them from msg_ext
  for_each (exp_ring,exp_pid,dest_ring,dest_pid) from msg_ext
  {
    if(dest_ring == trid) // if the message is for a process in this ring
    {
      if(dest_pid In sprocs[trid])
        smsg_int[dest_pid] = smsg_int[dest_pid] Union ((exp_ring,exp_pid));
      msg_ext = msg_ext Minus ((exp_ring,exp_pid));
    }
  }

  // put the messages sent to other rings
```

2

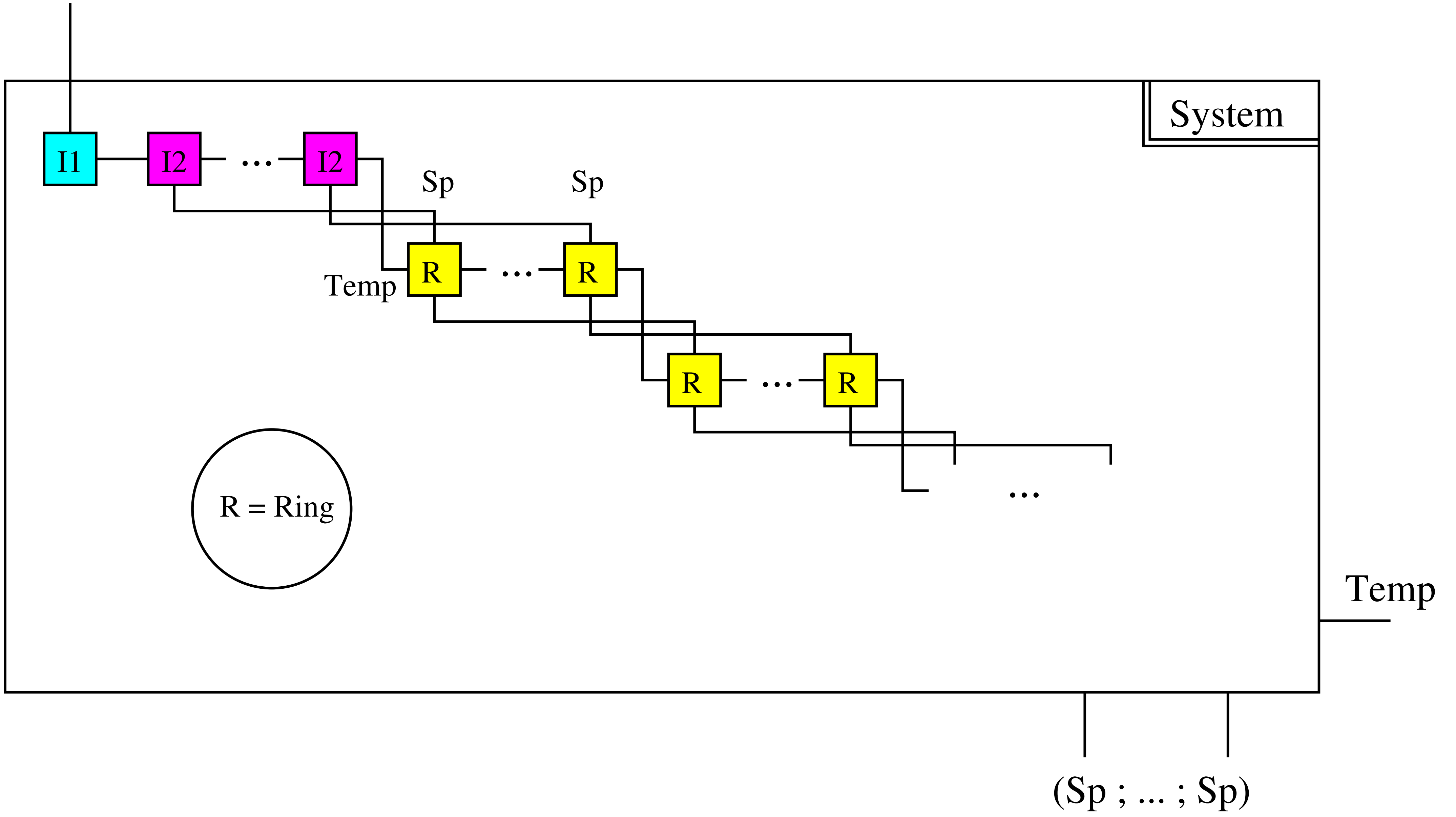
```
  msg_ext = msg_ext Union smsg_out;

  // keep big-token, if necessary
  if(bigtoken.ring == trid)
  {
    if(stoken.col == white && stoken.pos == 0)
    {
      if (trid == 0)
        bigtoken.col = white;
      if (trid != 0 && src == black)
      {
        bigtoken.col = black;
        src = white;
      }
      bigtoken.ring = bigtoken.ring + 1 [mod trings];
    }
  }
  ti = sti;
  tm = stm;
}
(speak Temp, ti, tm)(write stemp, Pst)

module RootInit2 (listen nil)(read stemp, Pst)
{
  temp = stemp; // vector assignment
  // choose a random number of communication rounds in the ring
  step = 0;
  maxsteps = random () + 1;
}
(speak temp, step, maxsteps)(write Pst)

module RootEnd (listen temp)(read Pst)
{
  stemp = temp; // vector assignment
}
(speak nil)(write stemp, Pst)

module D1 (listen temp)(read nil)
{
  told = length(procs[trid]);  // the number of current processes
  tnew = random();  // choose how many new processes to add
  tm = told + tnew;  // update tm
}
(speak temp, told)(write nil)

module Id1 (listen Temp)(read Pst)( null; )(speak Temp)(write Pst)
module Id2 (listen temp)(read Pst)( null; )(speak temp)(write Pst)
module Id3 (listen  nil)(read Pst)( null; )(speak  nil)(write Pst)

module NewProc (listen temp)(read Pst)
{
  pid = counter;  // pick the next free id from the list
  counter ++;  // increase the counter
  procs[trid] = procs[trid] Union (pid);
  c = white;
  active = true;
}
```

3

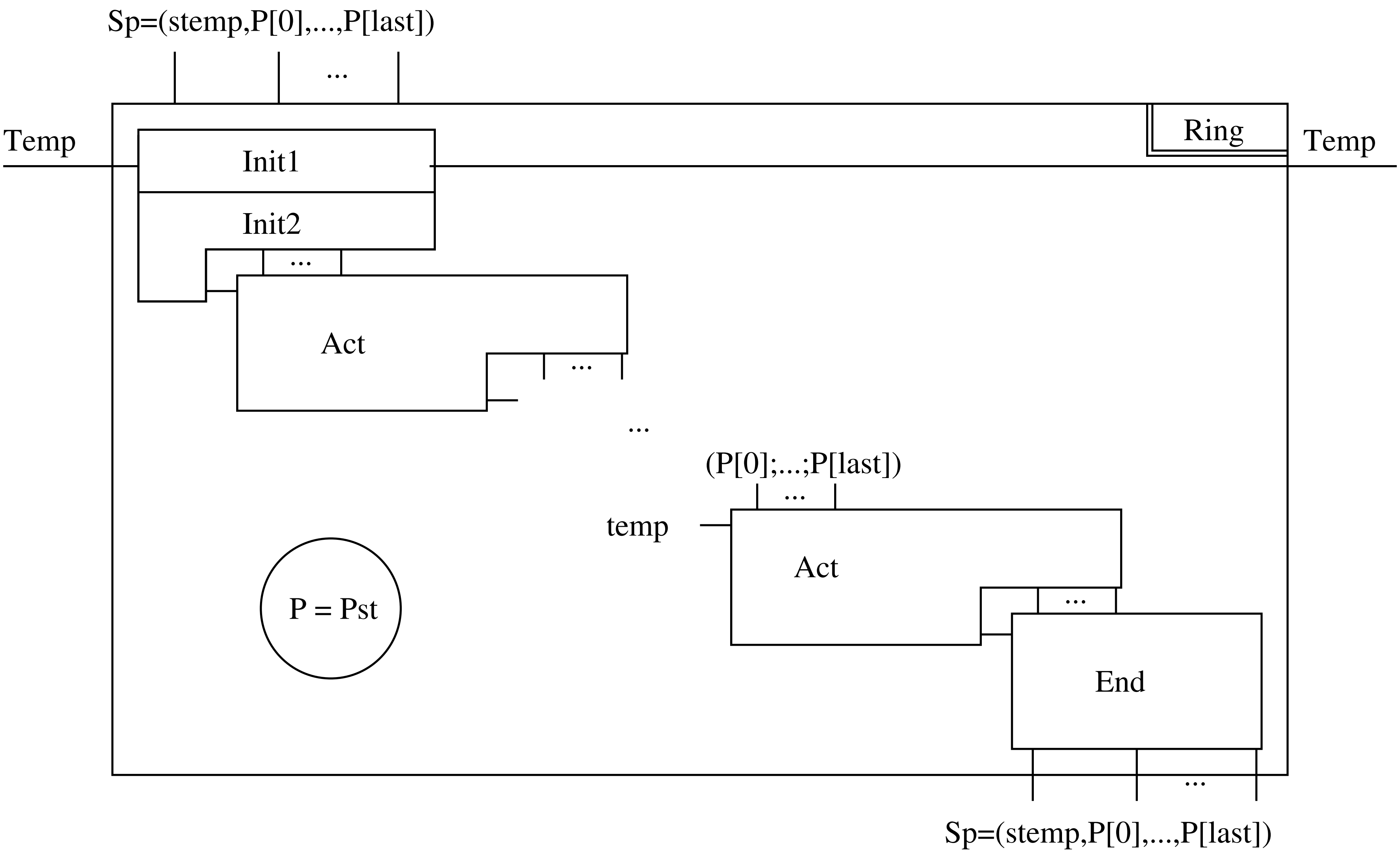
```
(speak temp)(write Pst)

module Stay (listen temp)(read Pst)
{
  if(msg_int[pid] != emptyset)  // take my jobs
  {
    msg_int[pid] = emptyset;
    active = true;
  }
  if(active)
  {
    delay(random_time);  // execute
    r = random(tm - 1);  // choose how many messages to send
    for(i=0; i<r; i++)
    {
      q = random(trings - 1); // choose the ring
      w = random in procs[q]; // choose the process
      if(q != rid) // for a process in another ring
        msg_out = msg_out Union ((rid,pid,q,w));
      elseif (w != pid) // for a process in the same ring
        msg_int[w] = msg_int[w] Union ((q,w));
      if(q < rid) // color the ring in black
        rc = black;
      if(q == rid && w < pid) // color the process in black
        c = black;
    }
    active = random(true,false);
  }
  if(!active && token.pos == pid) // termination
  {
    if(pid == 0) token.col = white;
    if(pid != 0 && c == black)
    {
      token.col = black; c = white;
    }
    token.pos = next(procs[rid],token.pos); // pass the token
  }
}
(speak temp)(write Pst)

module Leave (listen temp)(read Pst)
{
  token.pos = next(procs[rid],token.pos);  // pass the token
  procs[rid] = procs[rid] - pid;  // update process list
}
(speak temp)(write nil)
```

4

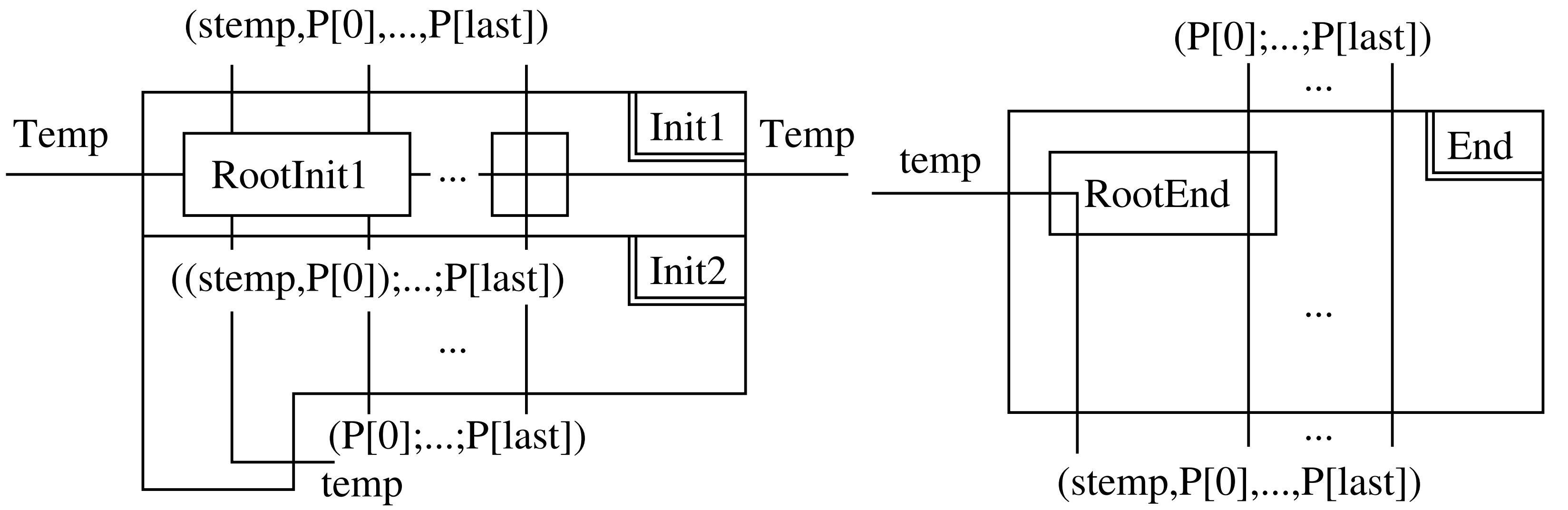
..Example: Ring of dynamic processes

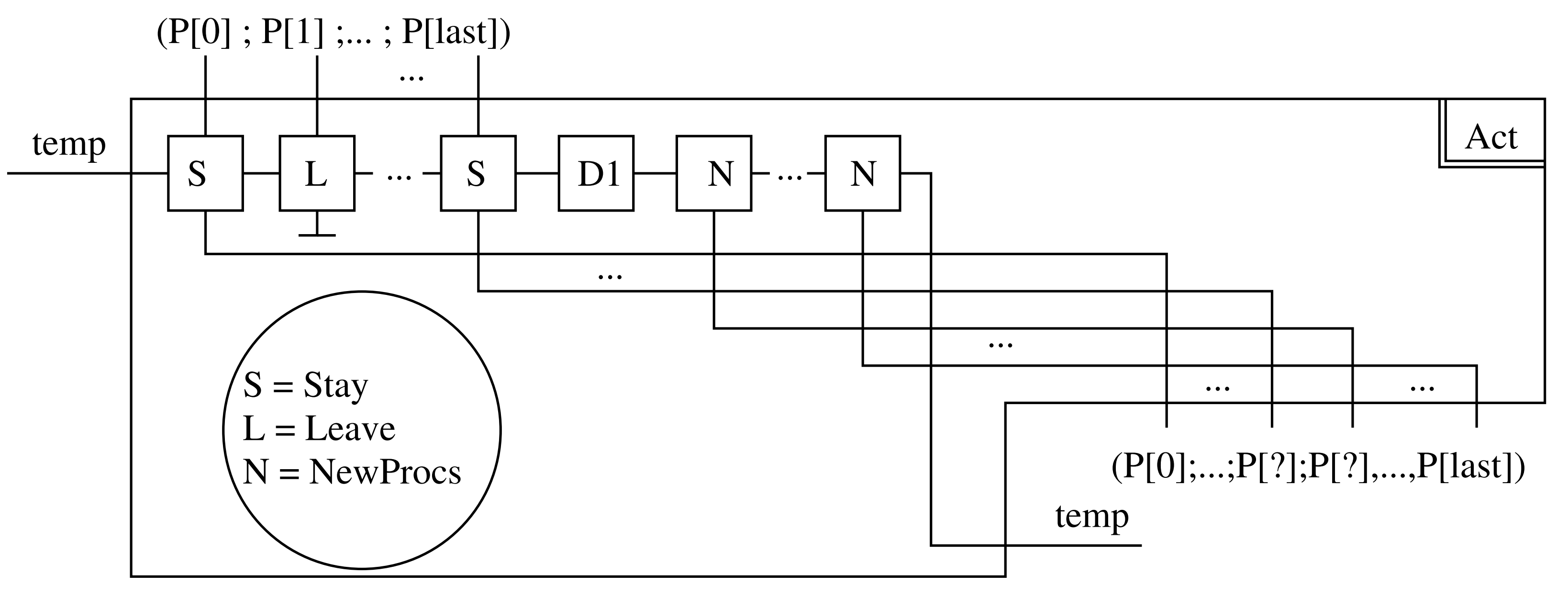


..Example: Ring of dynamic processes



..Example: Ring of dynamic processes





High-level srv-programs

Contents: