Lecţia 15:

Spatio-temporal logics for the verification of interactive programs

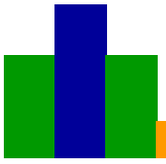
v1.0 (18.05.09)
Gheorghe Stefanescu — Universitatea Bucureşti

Metode de Dezvoltare Software, Sem.2
Februarie 2009— Iunie 2009

Contents

Contents

- *Generalities*
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- Conclusions



Models for interactive systems

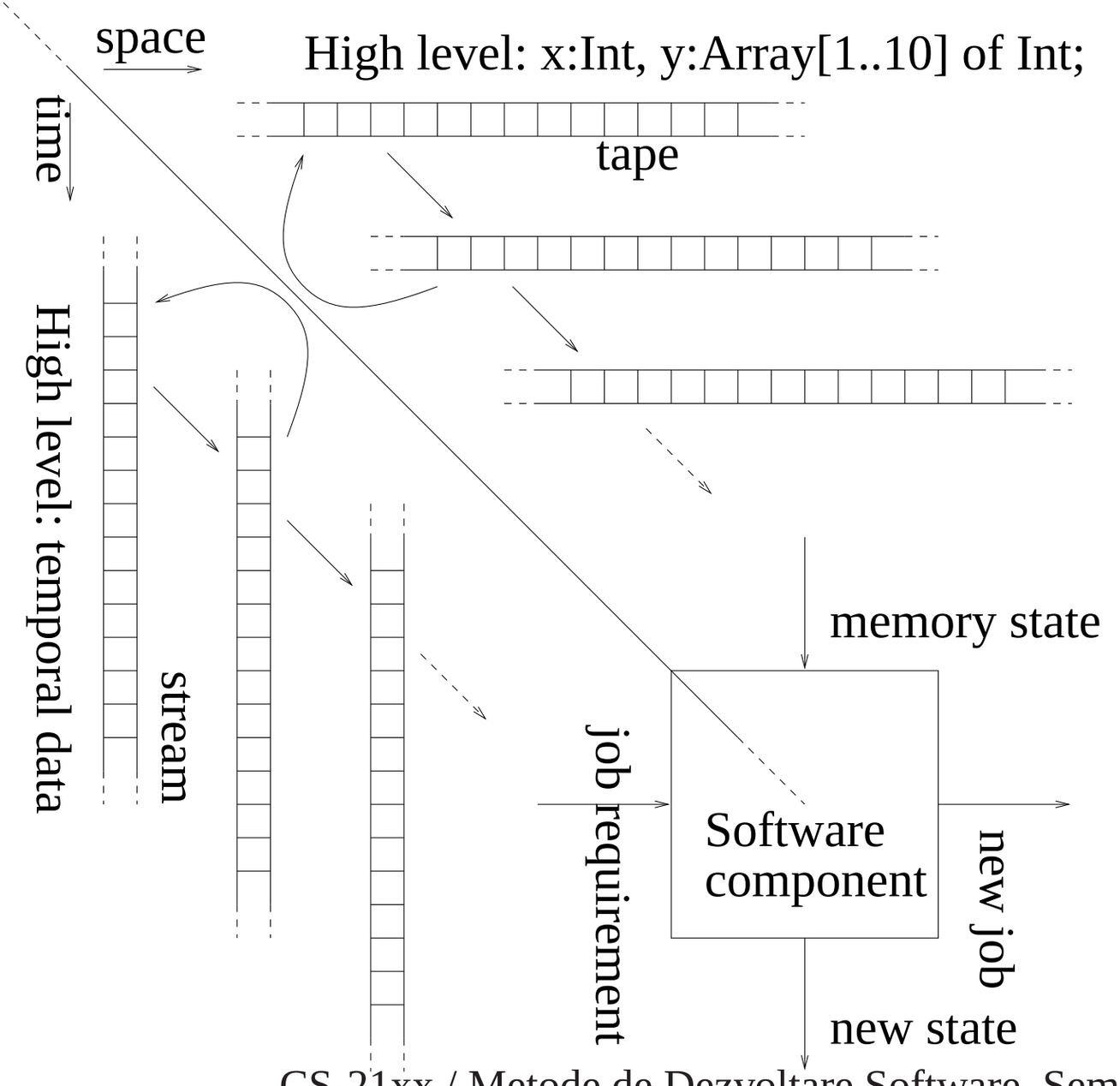
Rv-systems and Agapia Programming:

- counter machines, flowcharts, while programs
  - rv-systems include them and extend with interactive features
- π-calculus [Milner,..], actor systems [Agha,..]
  - rv-systems have data; srv-systems provide a (process) “name-free” interaction
- dataflow networks and Petri nets
  - rv-systems have dynamic, unbound interactions
- tile logic [Bruni, Gadducci, Montanari,..]
  - rv-systems provide an imperative programming paradigm

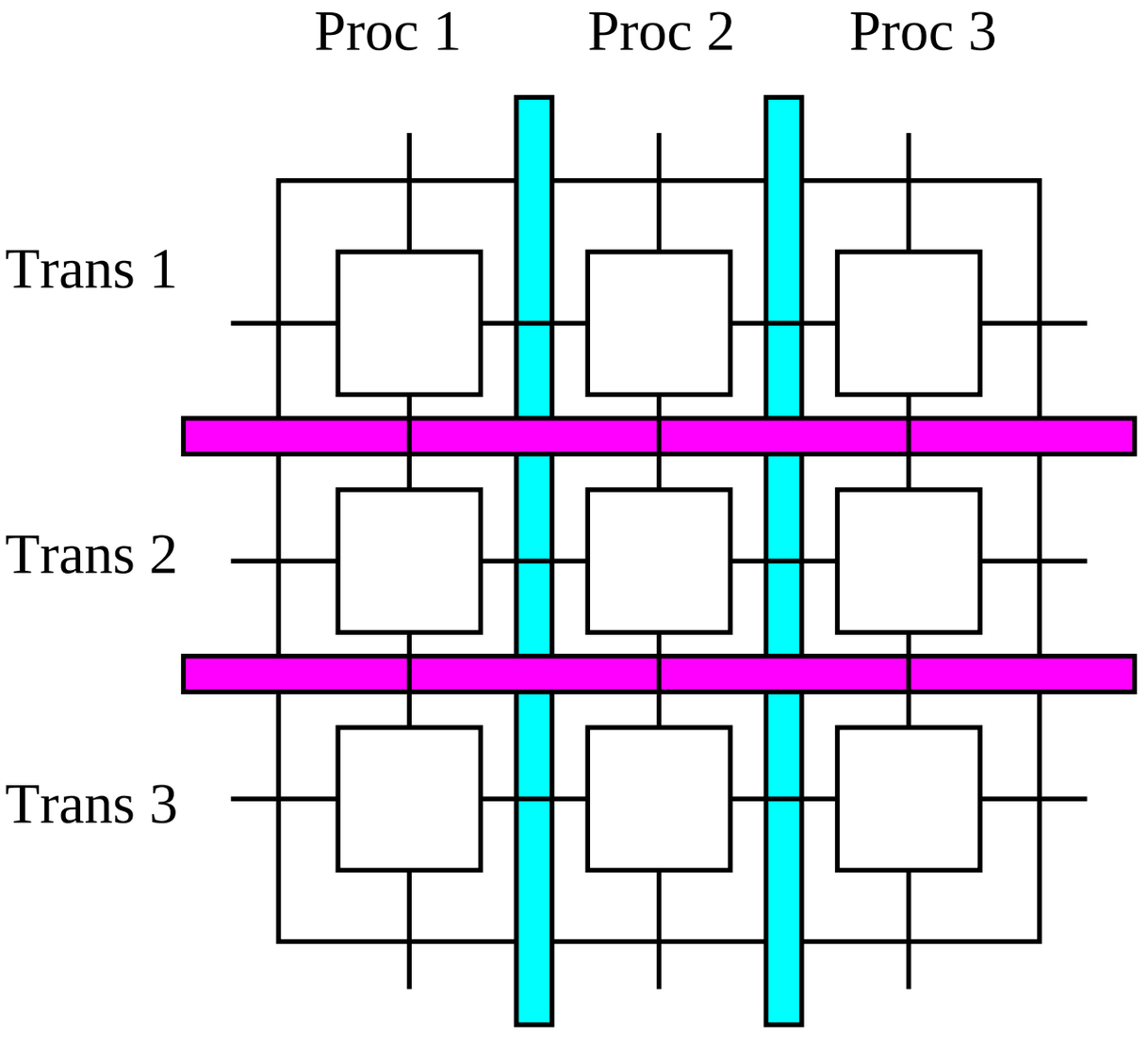
History:

- *space-time duality “thesis”* [*Network algebra*, Springer 2000]
- *finite interactive systems* Marktoberdorf Summer School 2001]
- *rv-systems* [NUS, Singapore, summer 2004]
- *structured rv-systems* [with Dragoi, fall 2006]

ST-Dual picture



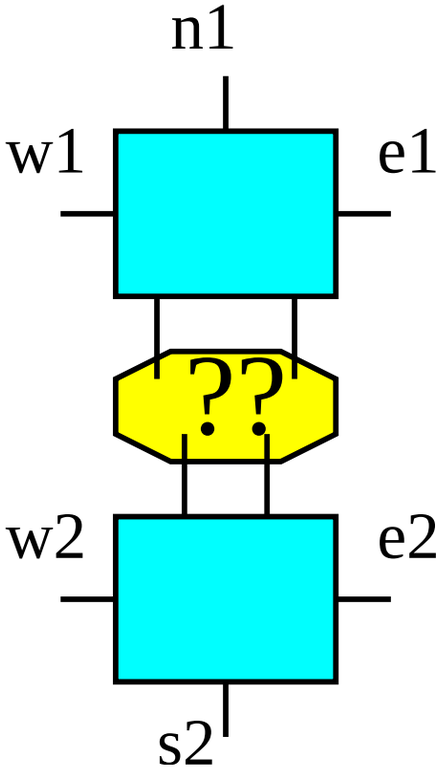
Processes and transactions



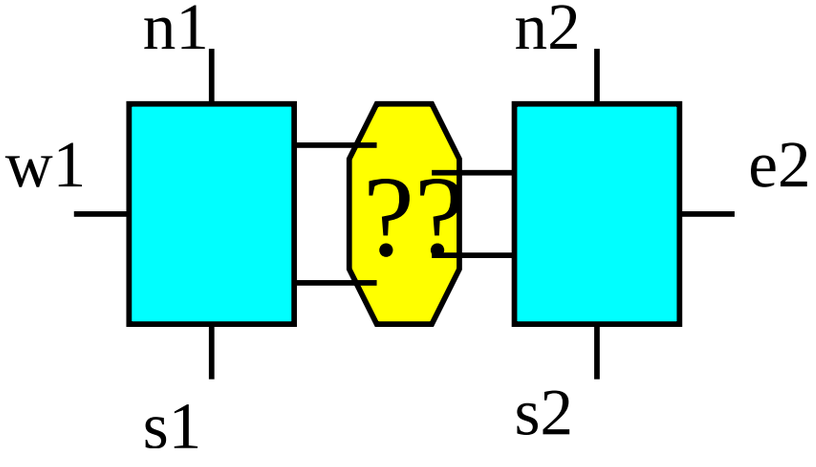
Types

Composition operations:

Vertical composition:



Horizontal composition:



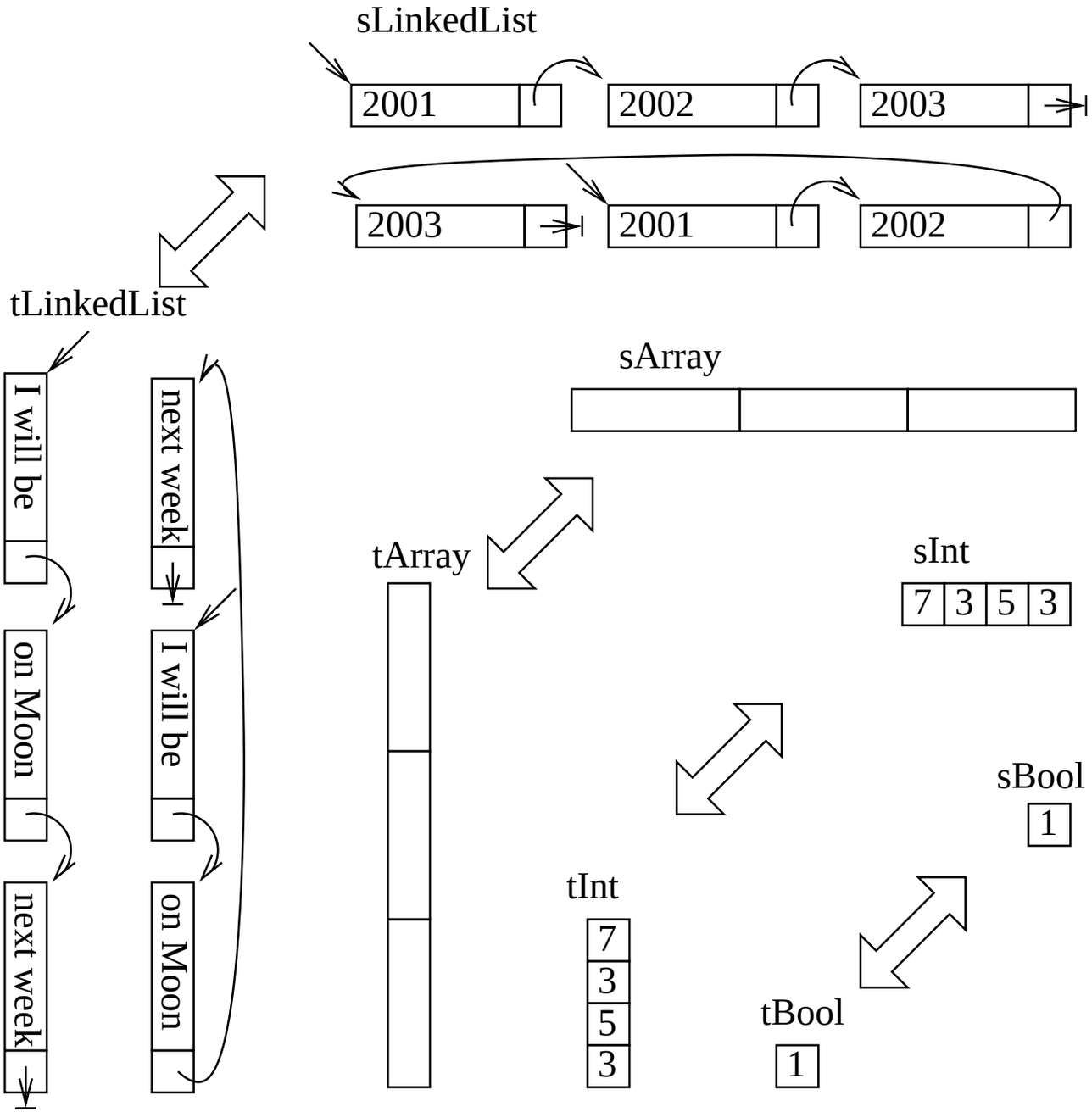
High level temporal structures

data with usual (*spatial*) representation:

sBool, sInt, sArray, sLinkedList, etc.

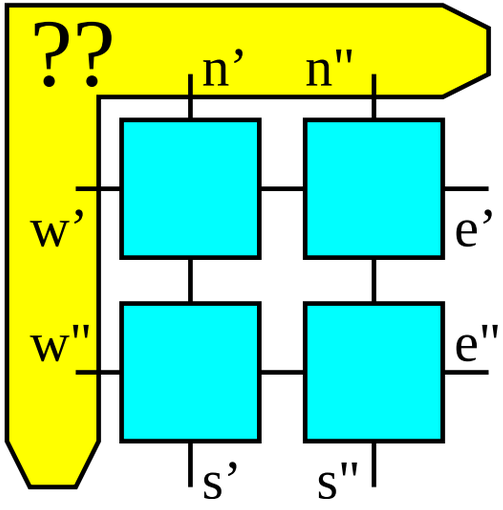
and their *time dual* (i.e., data with *temporal representation*):

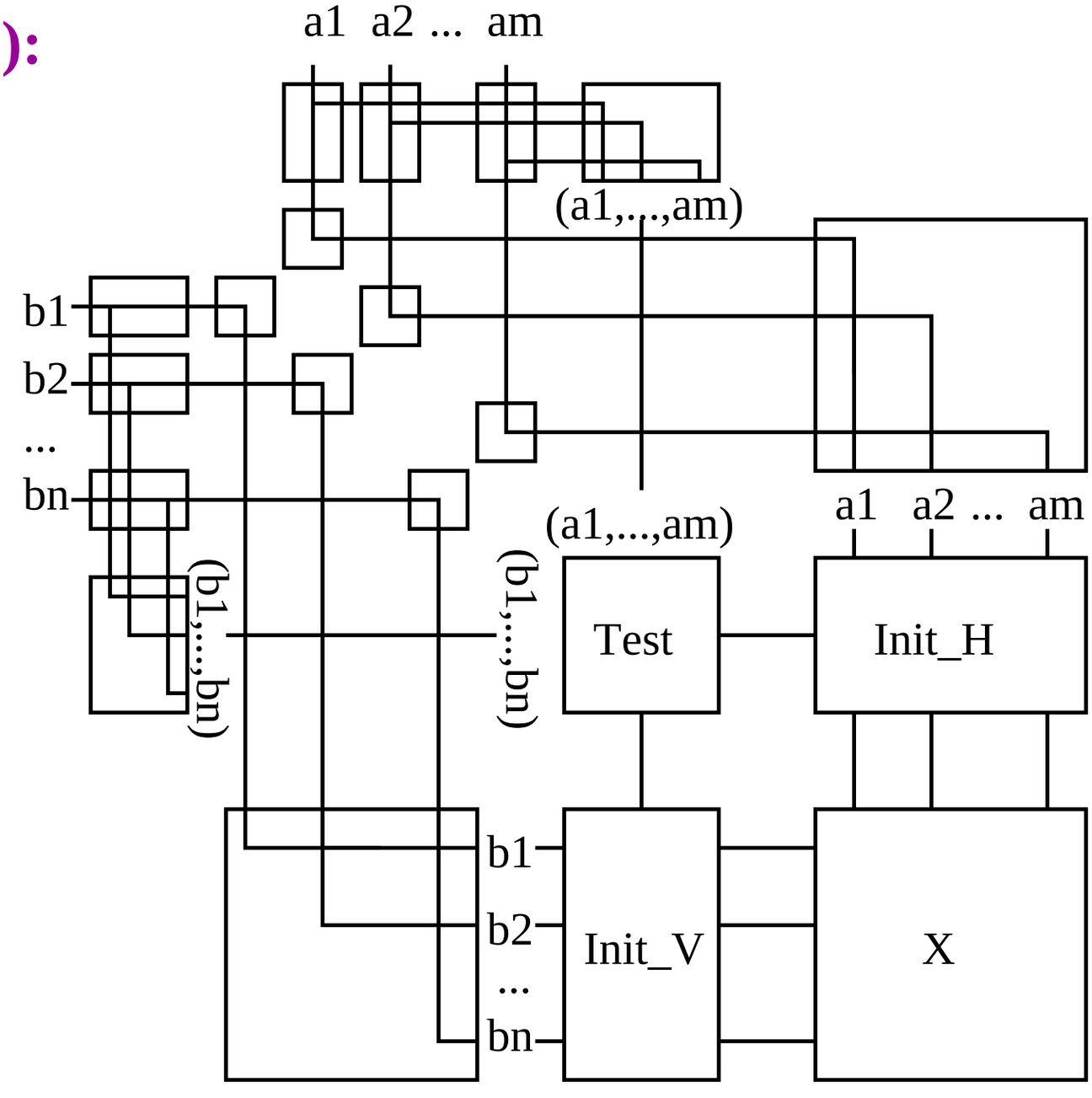
tBool, tInt, tArray, tLinkedList, etc.



Reshaping types

Collecting data (for “if”):





Contents

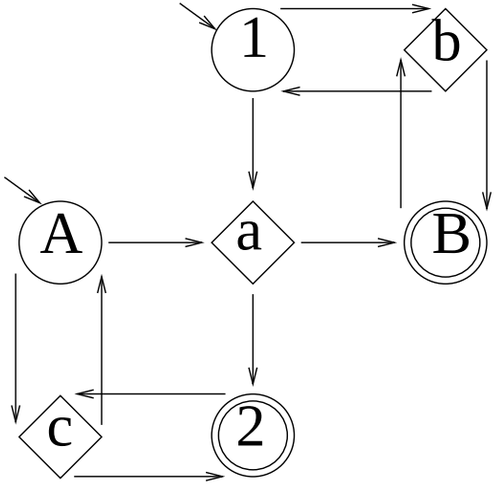
Contents

- Generalities
- *Finite interactive systems* ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- Conclusions

Finite interactive systems

Finite interactive systems:

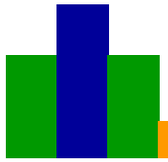
- *states*: 1,2 [1-initial; 2-final]
- *classes*: A, B [A-initial; B-final]
- *transitions*: a, b, c



Parsing procedure (to recognize grids):

A parssing for
```
abb
cab
cca
```
:

```
 1 1 1     1 1 1     1 1 1     1 1 1          1 1 1
Aa b b    AaBb b    AaBbBb    AaBbBb  ...    AaBbBbB
   2         2 1       2 1                    2 1 1
Ac a b    Ac a b    Ac a b    AcAa b         AcAaBbB
                              2                2 2 1
Ac c a    Ac c a    Ac c a    Ac c a         AcAcAaB
                                             2 2 2
```



FIS vs. 2-dimensional languages

Theorem:

The following are equivalent for a 2-dimensional language L (called recognizable two-dimensional language*; their class is denoted by REC):*

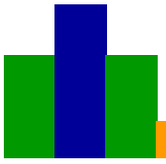
1. *L is by a a* finite interactive system*;*
2. *L is recognized by a* on-line tessellation automaton*;*
3. *L is defined by a* tile systems *(i.e., local lattice languages closed to letter-to-letter homomorphisms);*
4. *L is defined by an* existential monadic second order formula*;*
5. *etc.*

Notice: 2-dimensional languages are also known as "picture" languages.

Contents

Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- *Rv-programs* ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- Conclusions



RV-programs

RV-systems:

- An *rv-system* (*interactive system with registers and voices*) is a FIS enriched with:
  - *registers* associated to its *states* and *voices* associated to its *classes*;
  - appropriate *spatio-temporal transformations for actions*.

  We study rv-systems specified by *rv-programs* (see below)
- A *computation* is described by a scenario like in a FIS, but with concrete data around each action.

..RV-programs

An rv-program (for perfect numbers):

in: A,1; out: D,2

X::

(A,1)	x : sInt

	tx : tInt;
	tx = x;
	x = x/2;
	goto [B,3];

Y::

(B,1)	y : sInt
tx : tInt	y = tx; goto [C,2];

Z::

(C,1)	z : sInt
tx : tInt	z = tx; goto [D,2];

U::

(A,3)	x : sInt

	tx : tInt;
	tx = x;
	x = x − 1;
	if (x > 0) goto [B,3]
	else goto [B,2];

V::

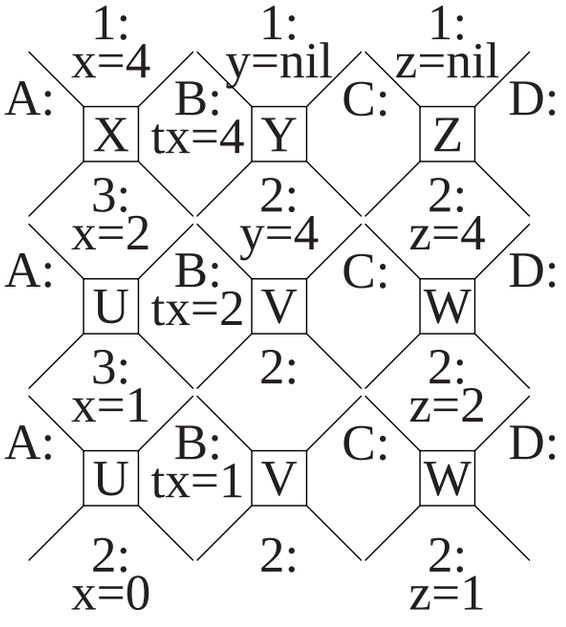
(B,2)	y : sInt
tx : tInt	if(y%tx != 0) tx = 0; goto [C,2];

W::

(C,2)	z : sInt
tx : tInt	z = z − tx; goto [D,2];

..RV-programs

Scenario:



Operational semantics:

- defined in terms of scenarious

Relational semantics:

- input-output relation generated by all possible scenarious

Contents

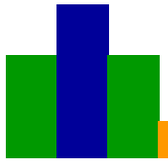
Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- *Structured rv-programs* ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- 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*
- a *name-free* interaction calculus



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}
```

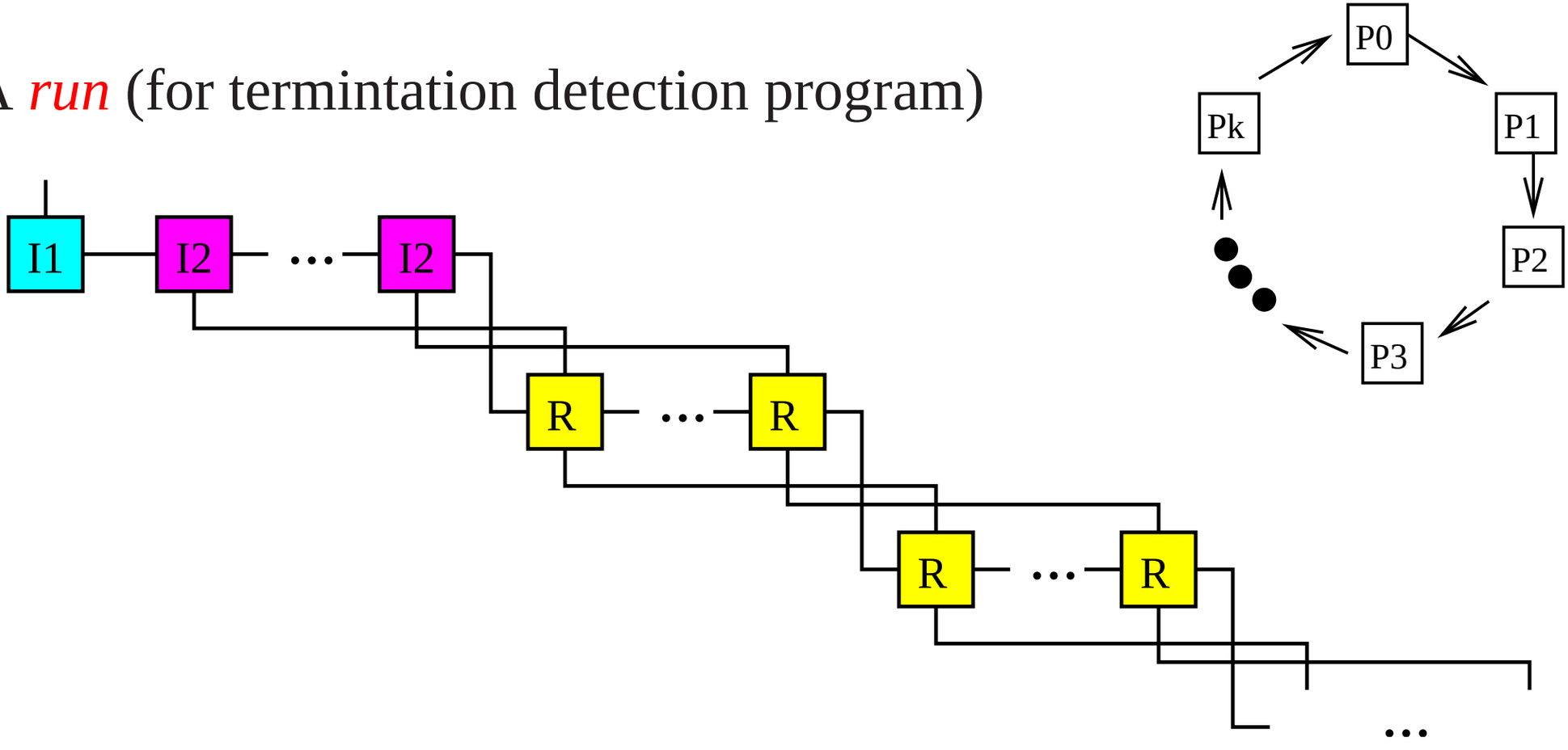
I2=
```
module{listen tm,tid,msg[ ],token(col,pos)}
{read nil}{
id=tid; c=white; active=true; msg[id]=null;
}{speak tm,tid,msg[ ],token(col,pos)}
{write id,c,active}
```

..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)

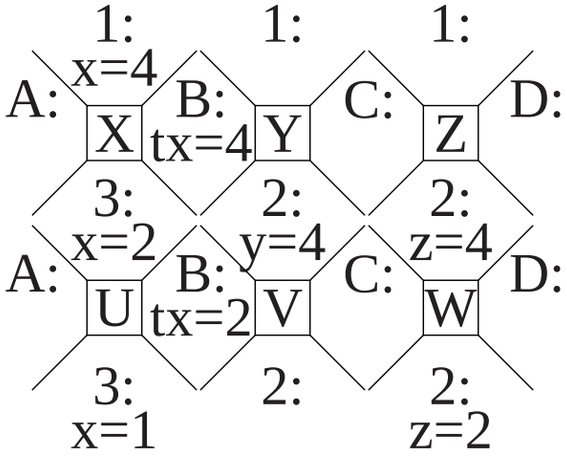


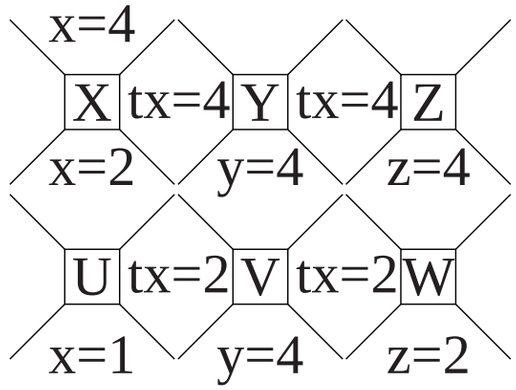

```
I1# for_s(tid=0;tid<tm;tid++){I2}#
$ while_st(!(token.col==white && token.pos==0)){
    for_s(tid=0;tid<tm;tid++){R}}
```

Scenarios

Scenarios:

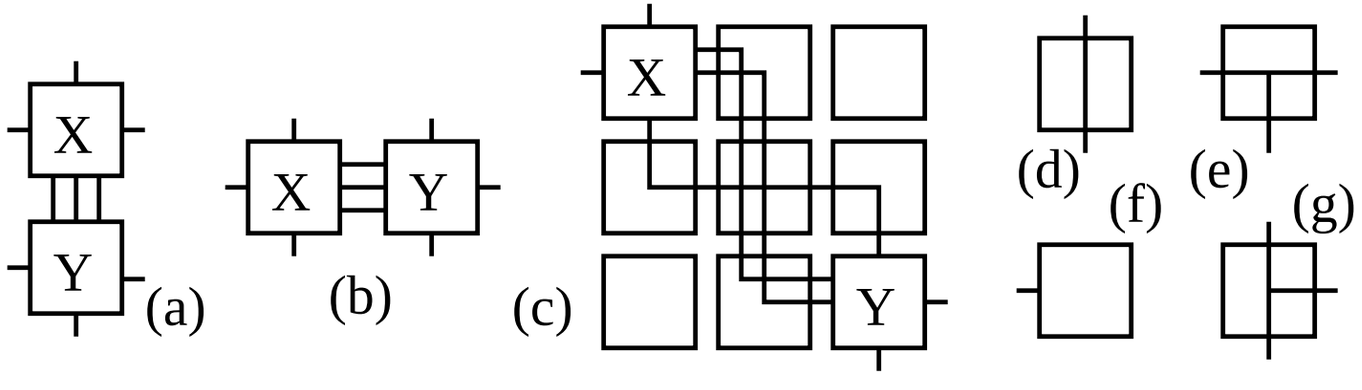
```
 1 1 1
AaBbBbB
 2 1 1
AcAaBbB
 2 2 1
AcAcAaB
 2 2 2
```





(1) FIS's scenario (2) rv-scenario (3) srv-scenario

Srv-scenario operations:

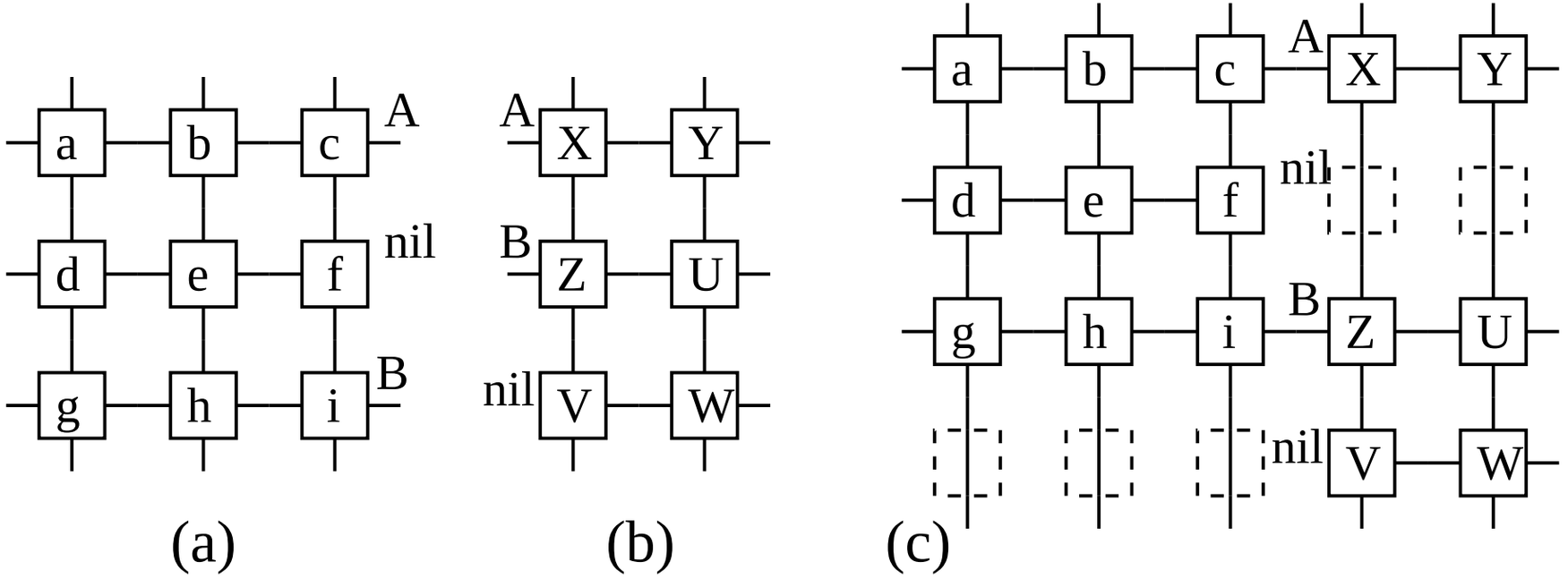


(4)

..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

Contents

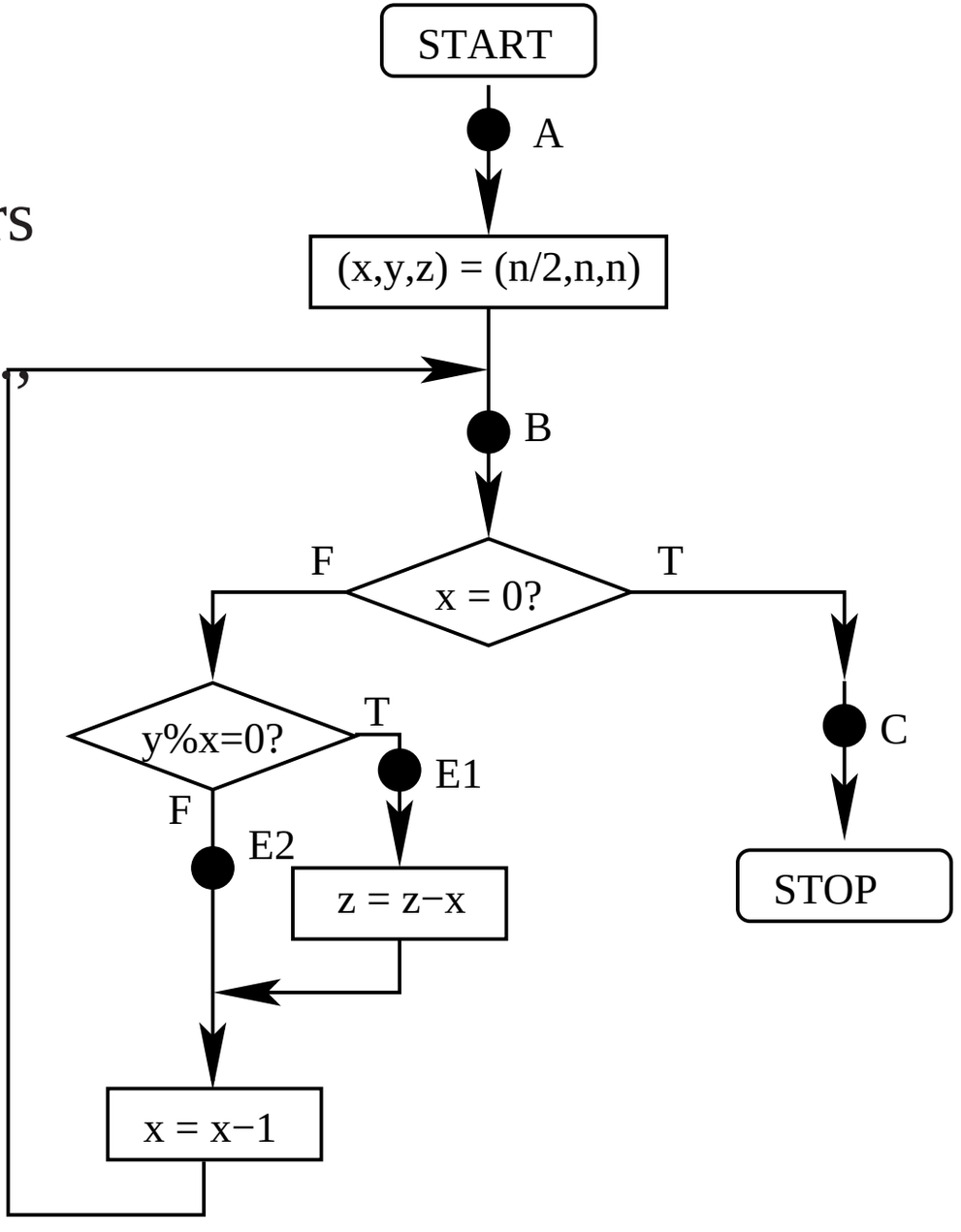
Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- *Floyd logics for rv-programs*
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- Conclusions

Floyd's method for flowchart programs

Floyd's method for flowcharts:

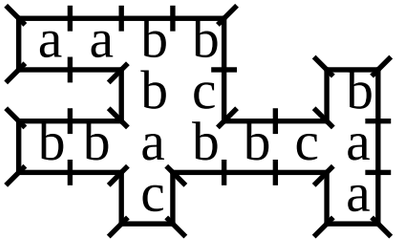
- a program for perfect numbers
- *cut-points* and *assertions*, e.g.,
  ϕ_B : "$0 \leq x \wedge y = n \geq 2$
  $\wedge z = n - \sum_{d|n, x<d<n} d$"
- *invariance conditions*, e.g.,
  $\phi_B \wedge C_{p(B,E1,B)} \Rightarrow \sigma_2(\phi_B)$
- *termination*: no infinite computation



Grids and scenarios

Grids:

```
aabbabb
abbcdbb
bbabbca
ccccaaa
```



(a) (b)

Contour-and-contents representation of grids:

The grid in (b) is represented as:

- Contour: $e^4s^2e^2n^1e^1s^3w^1n^1w^3s^1w^1n^1w^2n^1e^2n^1w^2n^1$
- Contents: $a^2b^3cb^3ab^2caca$.

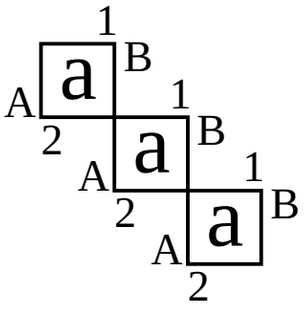
..Grids and scenarios

Scenarios:

(a)

```
 1 1 1
AaBbBbB
 2 1 1
AcAaBbB
 2 2 1
AcAcAaB
 2 2 2
```

(b)



Scenario = Grid + Data [around its letters]

Contour-and-contents representation of scenarios:

The scenario in (b) is represented as:

- Contour: $e_1 s_B e_1 s_B e_1 s_B w_2 n_A w_2 n_A w_2 n_A$
  (or, shortly, $(e_1 s_B)^3 (w_2 n_A)^3$)
- Contents: *aaa*.

..Verification of rv-programs

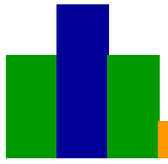
Assertions:

- *contours* with *assertions on state and class variables*;
- example:

  $e_1\{x = x_0\}s_D e_2\{z = x_0 - 1\}\ldots$

  means:

  — go towards east having on left the state 1 satisfying condition $x = x_0$;
  — then go towards south having on left the class D with no condition (i.e., $True$);
  — then go towards east having on left the state 2 with condition $z = x_0 - 1$; etc.



Verification of rv-programs

A framework for rv-program verification:

Three steps:

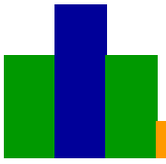
- find an appropriate set of *contours* and *assertions* (it should be a *finite* and *complete* set);
  [complete = all scenarious of the associated FIS may be decomposed into such contours]
- fill in the contours with all *possible scenarios*; and
- prove the *invariance condition*, i.e., these scenarios respect the border assertions.

Except for the guess of assertions, the proof is finite and fully automatic.

Contents

Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- *Hoare logics for srv-programs*
- Case study: Termination detection protocol
- Conclusions



Hoare logic

Differences (compared with the general Floyd method):

- take advantage of the structured programming constructs
- no control-interaction labels
- regular [rectangular] contours
- extend smoothly to large programs

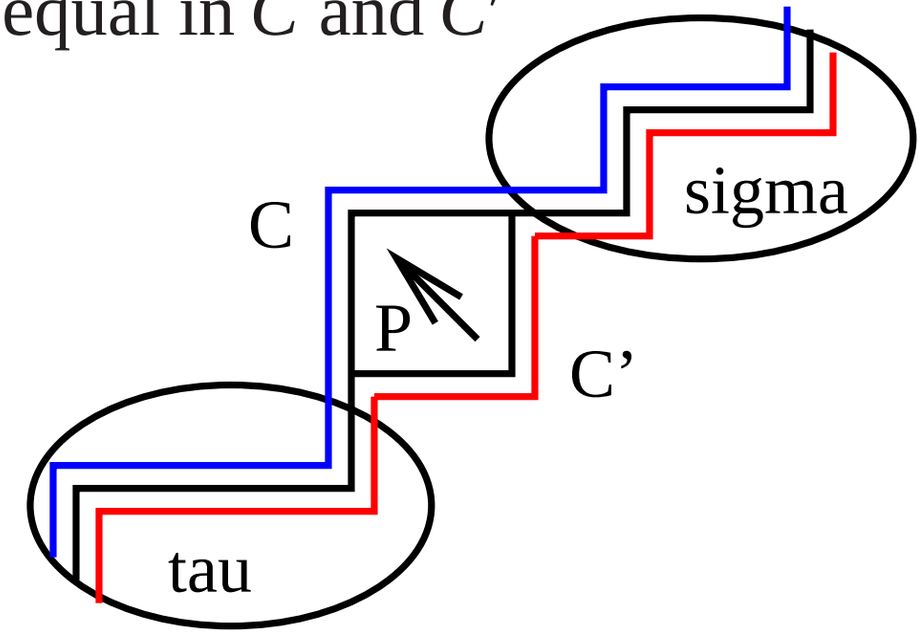
Verification rules

R0: Basic rule: For a module M, the validity of

$$\{\tau(NE)\sigma{:}C\}\ M\ \{\tau(EN)\sigma{:}C'\}$$

is reduced to:

- $\{C|_{NE}\}\ M\ \{C'|_{EN}\}$ for while programs and
- the variables in τ, σ are equal in C and C'



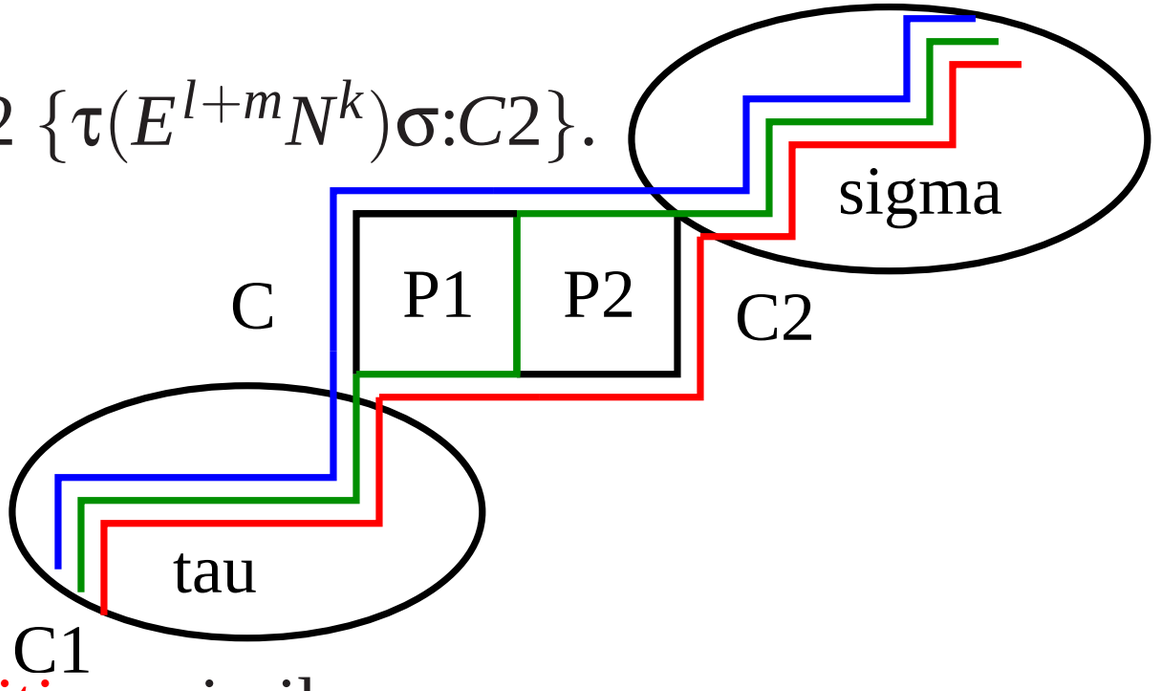
..Verification rules

R1: Rule for horizontal composition: If

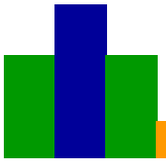
- $\{\tau(N^k E^l)E^m\sigma{:}C\}\ P1\ \{\tau(E^l N^k)E^m\sigma{:}C1\}$ and
- $\{\tau E^l(N^k E^m)\sigma{:}C1\}\ P2\ \{\tau E^l(E^m N^k)\sigma{:}C2\}$

then

- $\{\tau(N^k E^{l+m})\sigma{:}C\}\ P1\#P2\ \{\tau(E^{l+m} N^k)\sigma{:}C2\}$.



R2: Rule for vertical composition: similar



..Verification rules

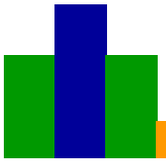
R3: Rule for diagonal composition: If

- $\{\tau(N^k E^l)\sigma{:}C\}\ P1\ \{\tau(E^l N^k)\sigma{:}C1\}$ and
- $\{\tau(N^k E^l)\sigma{:}C1\}\ P2\ \{\tau(E^l N^k)\sigma{:}C2\}$

then

- $\{\tau(N^k E^l)\sigma{:}C\}\ P1\$P2\ \{\tau(E^l N^k)\sigma{:}C2\}$.

Notice that C1 is used on two different paths.



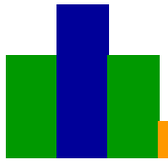
..Verification rules

R4: Rule for "if": For $Q = if(Cond)\{P1\}else\{P2\}$, if

- $\{\tau(N^k E^l)\sigma{:}C \wedge Cond\}\ P1\ \{\tau(E^l N^k)\sigma{:}C'\}$ and
- $\{\tau(N^k E^l)\sigma{:}C \wedge \neg Cond\}\ P2\ \{\tau(E^l N^k)\sigma{:}C'\}$

then

- $\{\tau(N^k E^l)\sigma{:}C\}\ Q\ \{\tau(E^l N^k)\sigma{:}C'\}$



..Verification rules

R5: Rule for autonomous temporal or spatial "while":

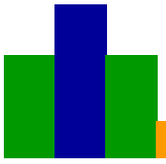
- for a temporal while with dummy temporal interfaces
  ... use the classical while rule
- similarly, a spatial while with dummy spatial interfaces

R6: Rule for spatio-temporal "while": If *Inv* is such that

- $\{\tau(N^k E^l)\sigma{:}Inv \wedge Cond\}\ P\ \{\tau(E^l N^k)\sigma{:}Inv\}$
  (Notice that *Inv* on left and right are on different paths.)

then

- $\{\tau(N^k E^l)\sigma{:}Inv\}\ while\_st(Cond)\{P\}$
  $\{\tau(E^l N^k)\sigma{:}Inv \wedge \neg Cond\}.$



..Verification rules

R7: Rule for a simple "for": If `i` is not changed by `R`, then for

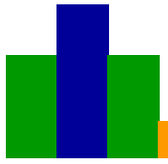
$$Q = \texttt{for\_s}(\texttt{i} = \texttt{0}; \texttt{i} < \texttt{a}; \texttt{i++})\{\texttt{R}\}$$

if

- $\{\tau(E^l)^j(N^kE^l)(E^l)^{a-j-1}\sigma{:}C_j\}\ R\ \{\tau(E^l)^j(E^lN^k)(E^l)^{a-j-1}\sigma{:}C_{j+1}\}$, for all $j < a$

then

- $\{\tau(N^k(E^l)^{a-1})\sigma{:}C_0\}\ Q\ \{\tau((E^l)^{a-1}N^k)\sigma{:}C_{a-1}\}$.



Soundness

R8: Rule for implication: Given a Hoare assertion

- $\{\tau(N^k E^l)\sigma{:}C\}\ P\ \{\tau(E^l N^k)\sigma{:}C'\}$

if $D \rightarrow C$, $C' \rightarrow D'$, then

- $\{\tau(N^k E^l)\sigma{:}D\}\ P\ \{\tau(E^l N^k)\sigma{:}D'\}$.

Proposition (Soundness): The inference rules R0-R8 are sound, i.e.,

if an assertion $\{\tau(N^k E^l)\sigma{:}C\}\ P\ \{\tau(E^l N^k)\sigma{:}C'\}$ *is proved*

then

all scenarios of P satisfying the input condition, satistfy the output condition, too.

Contents

Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- *Case study: Termination detection protocol*
- Conclusions

Termination detection protocol

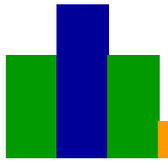
The program is:

```
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}}
```

Specification:

- input: n.
- output: $\forall k \in [0,n) : (id, c, active)[k] = (k, white, false)$
  $\wedge tn = n \wedge token = (white, 0) \wedge \forall k \in [0,n) : msg[k] = \emptyset$.

Notation: $(\texttt{id}, \texttt{c}, \texttt{active})[\texttt{k}]$ *denotes the variables* $(\texttt{id}, \texttt{c}, \texttt{active})$ *in process* `k`.



..Termination detection protocol

Invariant (for `for_s(tid=0;tid<tn;tid++){R}`):

P1: $token = (white, i) \rightarrow$

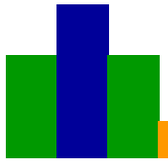
$$[(\forall r \in [0, i-1] : active[r] = false \wedge msg[r] = \emptyset)$$
$$\vee (\exists k > i-1 : c[k] = black)]$$

where the value $i-1$ is interpreted as $tn-1$ for $i = 0$.

(If the token is white and reached process i, then all processes with smaller `id` *terminate and have no pending messages in the sent list or a process with a larger* `id` *is black.)*

P2 $token.col = white \rightarrow (\forall k \in [0, n) : msg[k] \neq \emptyset \rightarrow c[k] = black)$

(If a process has a job inserted in the pending message list, then its color is black.)



Contents

A *detailed invariant* (for R): $Inv2 = \texttt{P1d} \wedge \texttt{P2d}$, where:

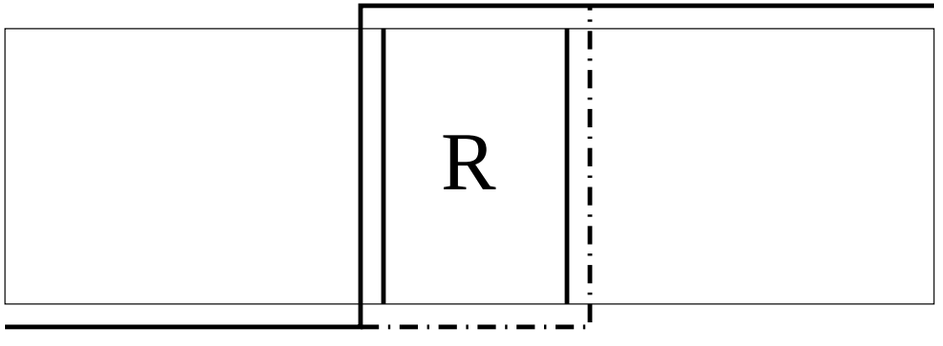
P1d: $token = (white, i) \rightarrow$

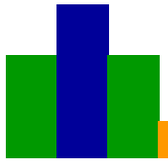
$$[(\forall r \in [0, i-1] : active[r] = false \wedge msg[r] \subseteq [max(tid, i), n))$$
$$\vee (\exists k > i-1 : c[k] = black)]$$

($i-1$ is interpreted as $tn-1$ for $i=0$)

P2d $token.col = white \rightarrow$

$$\forall k \in [0, tid) : msg[k] \subseteq [0, k) \cup [tid, n) \wedge$$
$$msg[k] \cap [0, k) \neq \emptyset \rightarrow c[k] = black$$





Contents

The final step: applying spatio-temporal while rule for $Q' =$ `for_(tid=0;tid<tn;tid++){R}` we get

$$\{|Inv|\}\texttt{while\_st(!(token=(white,0)))\{Q'\}}\{|Inv \wedge (token = (white, 0))|\}$$

hence

$$\forall i \in [0, tn-1] : active[i] = false \wedge msg[i] = \emptyset$$

or

all process have *terminated* and there are *no pending jobs/messages* in the communication lists.

Theorem: The program for the dual-pass ring termination detection protocol is correct.

Contents

Contents

- Generalities
- Finite interactive systems ← *[nfa]*
- Rv-programs ← *[flowchart programs]*
- Structured rv-programs ← *[while programs]*
- Floyd logics for rv-programs
- Hoare logics for srv-programs
- Case study: Termination detection protocol
- *Conclusions*

Concluzions, etc.

to be inserted...