

Introduction

Modelling parallel systems

Transition systems

Modeling hard- and software systems

Parallelism and communication



Linear Time Properties

Regular Properties

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

representation of data-dependent parallel systems with

\mathbb{P}_{oo} communication over **shared variables**

\mathbb{P}_{oo} **synchronous** message passing

\mathbb{P}_{oo} **asynchronous** message passing

representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing communication

• **asynchronous** message passing over **channels**

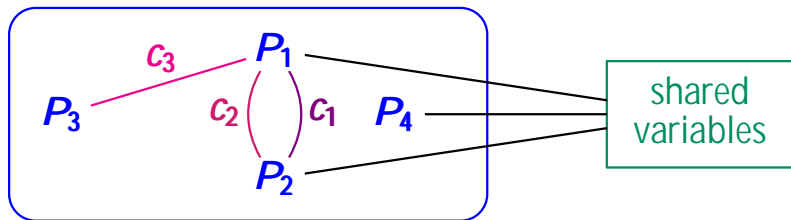
representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing

• **asynchronous** message passing

communication
over **channels**

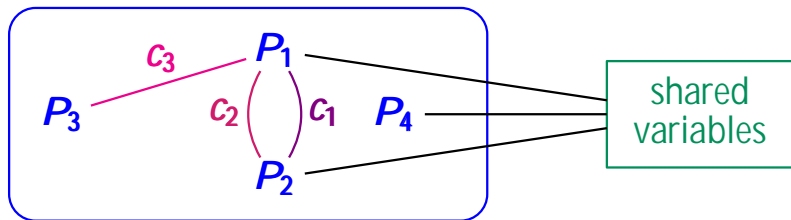


representation of data-dependent parallel systems with

communication over **shared variables**

communication over **shared variables**

communication over **channels**



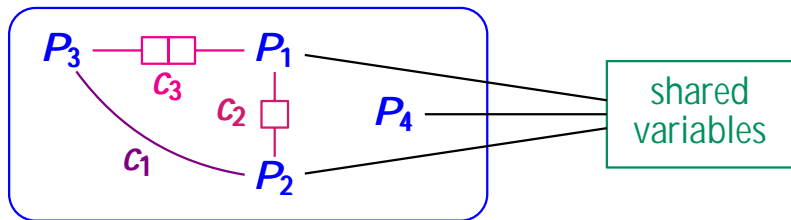
channel types: **synchronous** or **FIFO**

representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing communication

• **asynchronous** message passing over **channels**



channel types: **synchronous** or **FIFO**

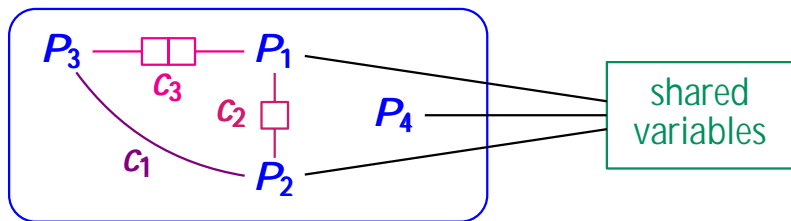
capacity = number of buffer cells

representation of data-dependent parallel systems with

communication over **shared variables**

communication over **synchronous** message passing capacity 0

communication over **asynchronous** message passing capacity 1



channel types: **synchronous** or **FIFO**

capacity 0 capacity = number of buffer cells

representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing communication

• **asynchronous** message passing over **channels**

formalization through **program graphs** for P_1, \dots, P_n

representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing communication

• **asynchronous** message passing over **channels**

formalization through **program graphs** for P_1, \dots, P_n

• with conditional transitions $i \xrightarrow{g} i$ (as before)

representation of data-dependent parallel systems with

• communication over **shared variables**

• **synchronous** message passing communication

• **asynchronous** message passing over **channels**

formalization through **program graphs** for P_1, \dots, P_n

• with conditional transitions $i \xrightarrow{g} i$ (as before)

• and **communication actions**

$i \xrightarrow{c!v}$	i sending value v via channel c
$i \xrightarrow{c?x}$	i receiving a value for variable x via channel c

typed variable: variable x with data domain $Dom(x)$

typed variable: variable x with data domain $Dom(x)$

evaluation for a set Var of typed variables:

type-consistent function $\sigma : Var \rightarrow Values$

i.e., $\sigma(x) \in Dom(x)$

typed variable: variable x with data domain $Dom(x)$

evaluation for a set Var of typed variables:

type-consistent function $\sigma : Var \rightarrow Values$

i.e., $\sigma(x) \in Dom(x)$

typed channel: channel c with

capacity $cap(c) \in \mathbb{N}$ and domain $Dom(c)$

typed variable: variable x with data domain $Dom(x)$

evaluation for a set Var of typed variables:

type-consistent function $\sigma : Var \rightarrow Values$

i.e., $\sigma(x) \in Dom(x)$

typed channel: channel c with

capacity $cap(c) \in \mathbb{N}$ and domain $Dom(c)$

evaluation for a set $Chan$ of typed channels:

type-consistent function $\sigma : Chan \rightarrow Values$

typed variable: variable x with data domain $Dom(x)$

evaluation for a set Var of typed variables:

type-consistent function $\sigma : Var \rightarrow Values$

i.e., $\sigma(x) \in Dom(x)$

typed channel: channel c with

capacity $cap(c) \in \mathbb{N}$ and domain $Dom(c)$

evaluation for a set $Chan$ of typed channels:

type-consistent function $\sigma : Chan \rightarrow Values$

s.t. $\sigma(c)$ is a word over $Dom(c)$ of length $cap(c)$

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs
over a pair (*Var*, *Chan*)

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs
over a pair $(Var, Chan)$

Var set of typed variables

$Chan$ set of typed channels with
capacities $cap(c)$ and domains $Dom(c)$

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs over a pair $(Var, Chan)$

Var set of typed variables

$Chan$ set of typed channels with capacities $cap(c)$ and domains $Dom(c)$

program graphs $P_i = (Loc_i, Act_i, E_{ect_i}, i, Loc_{0,i}, g_0)$ with conditional transitions

$g:$
 guarded command

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs over a pair $(Var, Chan)$

Var set of typed variables

$Chan$ set of typed channels with capacities $cap(c)$ and domains $Dom(c)$

program graphs $P_i = (Loc_i, Act_i, E_{ect_i}, Loc_{0,i}, g_0)$ with conditional transitions

g_i where $g_i = Cond(Var), Act_i$

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs over a pair $(Var, Chan)$

Var	set of typed variables
$Chan$	set of typed channels with capacities $cap(c)$ and domains $Dom(c)$

program graphs $P_i = (Loc_i, Act_i, E ect_i, i, Loc_{0,i}, g_0)$ with conditional transitions

$g:$
 i guarded command

$c!v$
 i sending value v via channel c

$P_1 / P_2 / \dots / P_n$ where P_i are program graphs over a pair $(Var, Chan)$

Var	set of typed variables
$Chan$	set of typed channels with capacities $cap(c)$ and domains $Dom(c)$

program graphs $P_i = (Loc_i, Act_i, E_{ect_i}, Loc_{0,i}, g_0)$ with conditional transitions

$g:$
 i guarded command

$c!v$
 i sending value v via channel c

$c?x$
 i receiving a value for variable x via channel c

asynchronous message passing via channels of capacity **1**

	enabled if ...	effect
sending $c!v$		
receiving $c?x$		

asynchronous message passing via channels of capacity 1

	enabled if ...	effect
sending $c!v$	channel c not full	$add(c, v)$
receiving $c?x$		



asynchronous message passing via channels of capacity 1

	enabled if ...	effect
sending $c!v$	channel c not full	$add(c, v)$
receiving $c?x$	channel c not empty $v = front(c)$	$x := v$ $remove(c)$



Effect of communication actions in CS

pc2.2-26

asynchronous message passing via channels of capacity 1

	enabled if ...	effect
sending $c!v$	channel c not full	$add(c, v)$
receiving $c?x$	channel c not empty $v = front(c)$	$x := v$ $remove(c)$



$$x = v$$

asynchronous message passing via channels of capacity 1

	enabled if ...	effect
sending $c!v$	channel c not full	$add(c, v)$
receiving $c?x$	channel c not empty $v = front(c)$	$x := v$ $remove(c)$

synchronous message passing via channels of capacity 0

$\not\models c!v$ and $c?x$ are executed at the same time

$\not\models$ effect $x := v$

channel system over (*Var*, *Chan*)

$$C = P_1 / \dots / P_n$$

transition system T_C

channel system over (*Var*, *Chan*)

$$\mathcal{C} = P_1 / \dots / P_n$$

transition system $T_{\mathcal{C}}$

states of $T_{\mathcal{C}}$ have the form

$1, \dots, n, '$

locations of P_1, \dots, P_n

channel evaluation

variable valuation

states $1, \dots, n$, where

i location of program graph P_i ,

$Eval(Var)$ variable evaluation

$Eval(Chan)$ channel evaluation

states $1, \dots, n$, where

i location of program graph P_i ,

$Eval(Var)$ variable evaluation

$Eval(Chan)$ channel evaluation

variable evaluation:

$: Var \rightarrow \mathcal{D}_o$ with $(x) \in Dom(x)$

channel evaluation:

$: Chan \rightarrow \mathcal{D}_c$ with $(c) \in Dom(c)$
and $| (c) | = cap(c)$

states $1, \dots, n$, where

i location of program graph P_i ,

$Eval(Var)$ variable evaluation

$Eval(Chan)$ channel evaluation

variable evaluation:

$: Var \rightarrow \mathcal{D}_o$ with $(x) \in Dom(x)$

channel evaluation:

$: Chan \rightarrow \mathcal{D}_o$ with $(c) \in Dom(c)$
and $| (c) | = cap(c)$

only channels c with $cap(c) = 1$ are relevant

Transition relation of channel systems

pc2.2-28

states $1, \dots, n$, where i Loc_i , $Eval(Var)$,
 $Eval(Chan)$

Transition relation of channel systems

pc2.2-28

states $1, \dots, n$, where i Loc_i , $Eval(Var)$,
 $Eval(Chan)$

transition relation \rightarrow is given by SOS-rules:

\rightarrow interleaving rules for Act_i

\rightarrow rules for message passing along channels

Transition relation of channel systems

pc2.2-28

states $1, \dots, n$, where i Loc_i , $Eval(Var)$, $Eval(Chan)$

transition relation \rightarrow is given by SOS-rules:

\rightarrow interleaving rules for Act_i

\rightarrow rules for message passing along channels

interleaving rule for actions Act_i :

$$\frac{i \quad g: \quad i \quad i \quad \models g}{1, \dots, i, \dots, n, \rightarrow 1, \dots, j, \dots, n, E \text{ ect}_i(,),}$$

Transition relation of channel systems

pc2.2-28

states $1, \dots, n$, where i Loc_i , $Eval(Var)$, $Eval(Chan)$

transition relation \rightarrow is given by SOS-rules:

\rightarrow interleaving rules for Act_i

\rightarrow rules for message passing along channels

interleaving rule for actions Act_i :

$$\frac{i \xrightarrow{g} i \quad i \models g}{1, \dots, i, \dots, n \rightarrow 1, \dots, j, \dots, n, E \text{ect}_i(,),}$$

does not affect the channel evaluation

SOS-rules for asynchronous message passing

pc2.2-29

for channel c with $cap(c)$ 1

SOS-rules for asynchronous message passing

pc2.2-29

for channel c with $cap(c) = 1$

receiving a message:

$$\frac{\begin{array}{c} c?x \\ i \quad i \quad j \end{array} \quad (c) = v_1 v_2 \dots v_k \quad k \quad 1}{1, \dots, i, \dots, n, \quad , \quad \text{no} \quad 1, \dots, j, \dots, n, \quad ,}$$

SOS-rules for asynchronous message passing

pc2.2-29

for channel c with $cap(c) = 1$

receiving a message:

$$\frac{i \quad c?x \quad i \quad j \quad (c) = v_1 v_2 \dots v_k \quad k \quad 1}{1, \dots, i, \dots, n, \quad , \quad \%o \quad 1, \dots, j, \dots, n, \quad ,}$$

where $\%o = [x := v_1]$

$$[x := v_1](y) = \begin{matrix} (y) & \text{if } y = x \\ v_1 & \text{if } y = x \end{matrix}$$

SOS-rules for asynchronous message passing

pc2.2-29

for channel c with $cap(c) = 1$

receiving a message:

$$\frac{\begin{array}{c} c?x \\ i \quad i \quad i \end{array} \quad (c) = v_1 v_2 \dots v_k \quad k = 1}{1, \dots, i, \dots, n, \quad , \quad \text{no} \quad 1, \dots, i, \dots, n, \quad ,}$$

where $\text{no} = [x := v_1]$ and $\text{no} = [c := v_2 \dots v_k]$

$$[x := v_1](y) = \begin{array}{ll} (y) & \text{if } y = x \\ v_1 & \text{if } y = x \end{array}$$

$$[c := v_2 \dots v_k](d) = \begin{array}{ll} (d) & \text{if } d = c \\ v_2 \dots v_k & \text{if } d = c \end{array}$$

SOS-rules for asynchronous message passing

pc2.2-29

for channel c with $cap(c) = 1$

receiving a message:

$$\frac{i \quad c?x \quad i \quad j \quad (c) = v_1 v_2 \dots v_k \quad k = 1}{1, \dots, i, \dots, n, \quad , \quad \text{no} \quad 1, \dots, j, \dots, n, \quad ,}$$

where $\text{no} = [x := v_1]$ and $\text{no} = [c := v_2 \dots v_k]$

sending a message:

$$\frac{i \quad c!v \quad i \quad j \quad (c) = v_1 \dots v_k \quad k < cap(c)}{1, \dots, i, \dots, n, \quad , \quad 1, \dots, j, \dots, n, \quad , \quad [c := v_1 \dots v_k v]}$$

SOS-rules for synchronous message passing

pc2.2-30

for synchronous channel c :

$$\frac{\begin{array}{c} i \quad c?x \quad i \quad i \quad j \quad c!v \quad j \quad j \quad i = j \\ \hline 1, \dots, i, \dots, j, \dots, n, \quad , \quad 1, \dots, i, \dots, j, \dots, n, \quad , \end{array}}{\quad}$$

for synchronous channel c :

$$\begin{array}{c}
 \begin{array}{ccccc}
 & c?x & & c!v & \\
 i & & i & j & i = j \\
 & & i & j & j
 \end{array} \\
 \hline
 1, \dots, i, \dots, j, \dots, n, \quad , \quad 1, \dots, i, \dots, j, \dots, n, \quad ,
 \end{array}$$

where $\quad = [x := v]$

for synchronous channel c :

$$\begin{array}{c}
 \begin{array}{ccccc}
 & c?x & & c!v & \\
 i & & i & j & j & i = j \\
 \hline
 1, \dots, i, \dots, j, \dots, n, \quad , & & 1, \dots, i, \dots, j, \dots, n, \quad ,
 \end{array}
 \end{array}$$

where $\quad = [x := v]$ and $\quad =$

How many states...

pc2.2-31

has a transition system for a channel system with ... ?

~~2~~ processes with **2** locations each

~~2~~ Boolean variables

~~2~~ channels of capacity **10** and Boolean values

How many states...

pc2.2-31

has a transition system for a channel system with ... ?

~~2~~ processes with 2 locations each

~~2~~ Boolean variables

~~2~~ channels of capacity 10 and Boolean values

answer:

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot (2^{11} \cancel{2}) \cdot (2^{11} \cancel{2})$$

$$\text{note: } 2^{11} \cancel{2} = 1 + 2 + 2^2 + \dots + 2^{10}$$

How many states...

pc2.2-31

has a transition system for a channel system with ... ?

~~2~~ processes with 2 locations each

~~2~~ Boolean variables

~~2~~ channels of capacity 10 and Boolean values

answer:

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot (2^{11} \cancel{10}) \cdot (2^{11} \cancel{10}) > 2^{24} > 25 \text{ mio}$$

$$\text{note: } 2^{11} \cancel{10} = 1 + 2 + 2^2 + \dots + 2^{10}$$

How many states...

pc2.2-31

has a transition system for a channel system with ... ?

~~2~~ processes with 2 locations each

~~2~~ Boolean variables

~~2~~ channels of capacity 10 and Boolean values

answer:

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot (2^{11} \cancel{2}) \cdot (2^{11} \cancel{2}) > 2^{24} > 25 \text{ mio}$$

$$\text{note: } 2^{11} \cancel{2} = 1 + 2 + 2^2 + \dots + 2^{10}$$

... with an unbounded channel ?

How many states...

pc2.2-31

has a transition system for a channel system with ... ?

~~2~~ processes with 2 locations each

~~2~~ Boolean variables

~~2~~ channels of capacity 10 and Boolean values

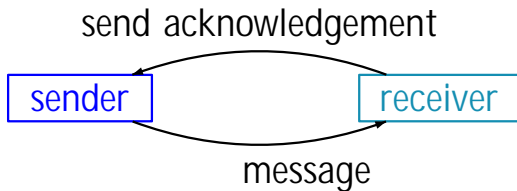
answer:

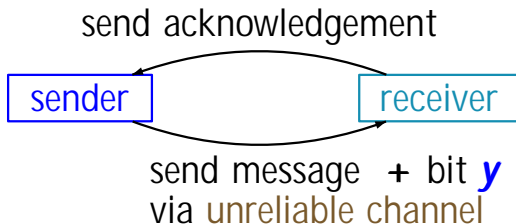
$$2 \cdot 2 \cdot 2 \cdot 2 \cdot (2^{11} \cancel{2}) \cdot (2^{11} \cancel{2}) > 2^{24} > 25 \text{ mio}$$

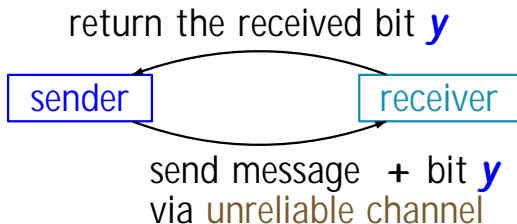
$$\text{note: } 2^{11} \cancel{2} = 1 + 2 + 2^2 + \dots + 2^{10}$$

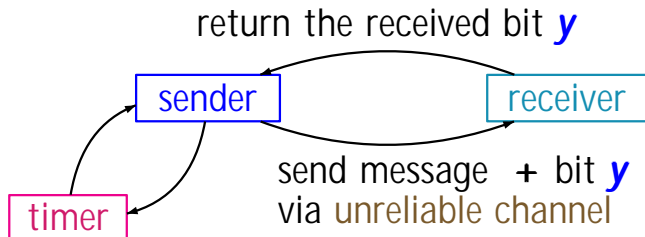
... with an unbounded channel ?

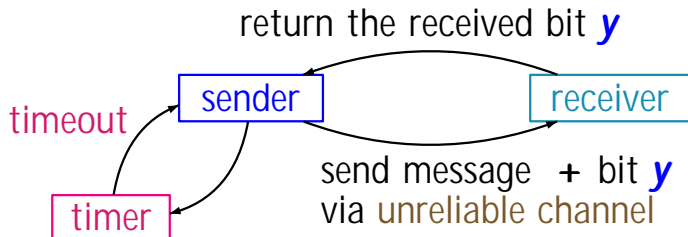
answer:











LOOP FOREVER

(1) send message + bit **y** and activate timer

(2) AWAIT **timeout** or **acknowledgement** DO

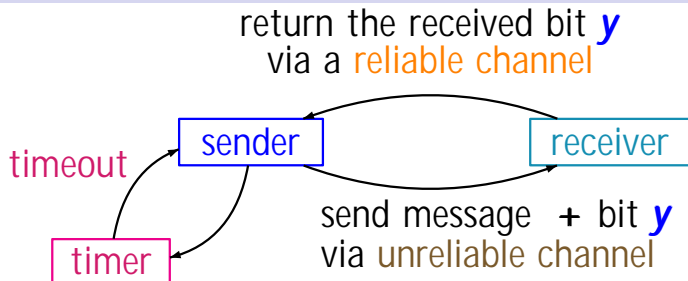
IF **timeout** THEN goto (1)

ELSE turn o timer; **y** := ~~Pop~~

OD FI

Protocol for the sender

pc2.2-32



LOOP FOREVER

(1) send message + bit y and activate timer

(2) AWAIT **timeout** or **acknowledgement** DO

IF **timeout** THEN goto (1)

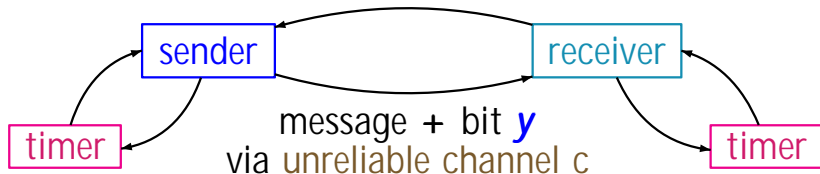
ELSE turn off timer; $y := \text{Pop}$

OD FI

If both channels are unreliable ...

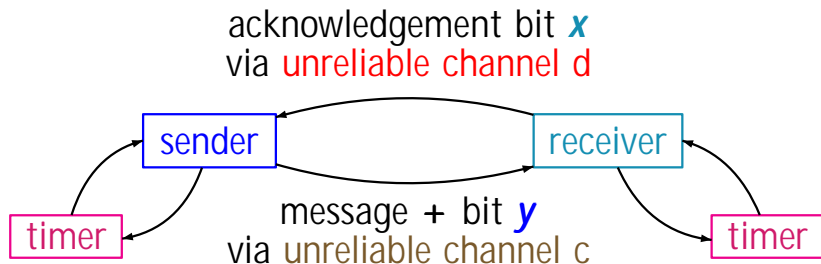
pc2.2-33

acknowledgement bit **x**
via **unreliable channel d**



If both channels are unreliable ...

pc2.2-33

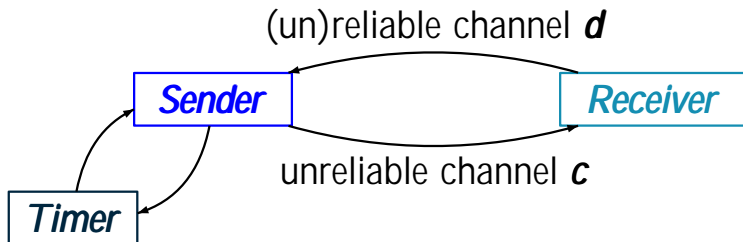


LOOP FOREVER

- (1) send message + bit y and activate timer
 - (2) AWAIT **timeout** or **acknowledgement x** DO
 - IF **timeout** THEN goto (1)
 - ELSE IF $x = y$ THEN turn off timer; $y :=$ ~~By~~ $y + 1$
 - ELSE ignore x
- OD FI FI

Alternating bit protocol (ABP)

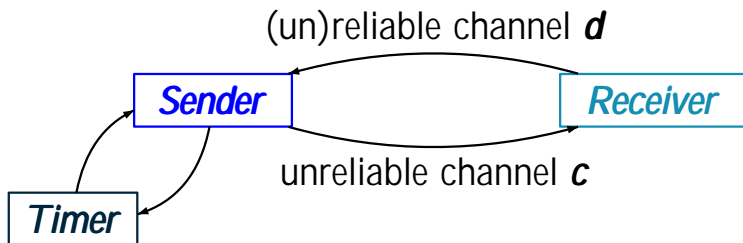
pc2.2-34



channel system: **Sender** / **Timer** / **Receiver**

Alternating bit protocol (ABP)

pc2.2-34



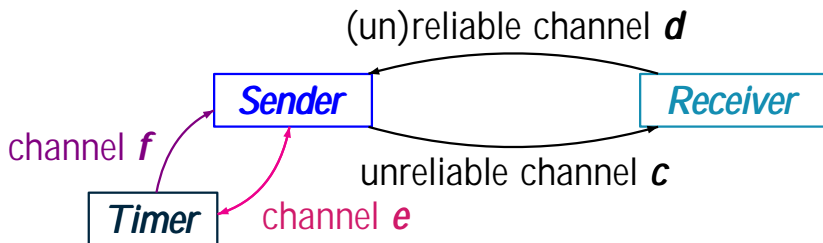
channel system: ***Sender*** / ***Timer*** / ***Receiver***

synchronous message passing between
Timer and ***Sender***

asynchronous message passing between
Receiver and ***Sender***

Alternating bit protocol (ABP)

pc2.2-34



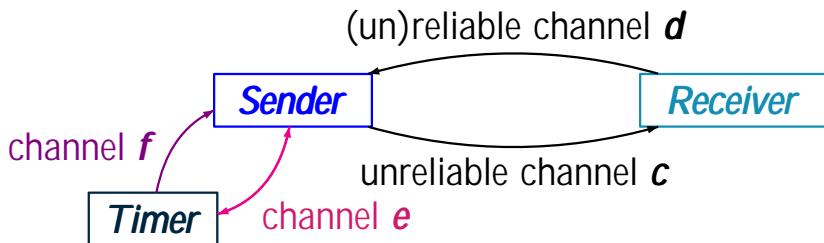
channel system: ***Sender*** / ***Timer*** / ***Receiver***

synchronous message passing between
Timer and ***Sender*** channels ***e*** and ***f***

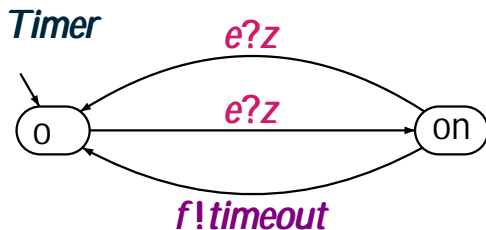
asynchronous message passing between
Receiver and ***Sender*** channels ***c***, ***d***

Alternating bit protocol (ABP)

pc2.2-34

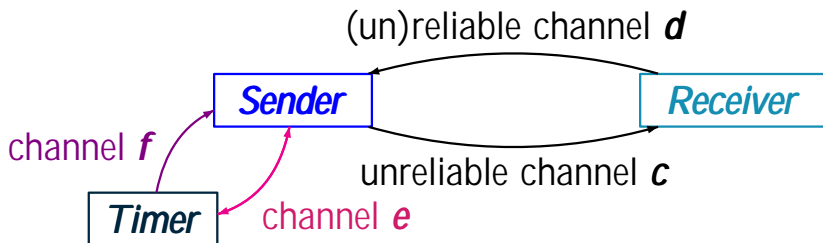


channel system: **Sender** / **Timer** / **Receiver**

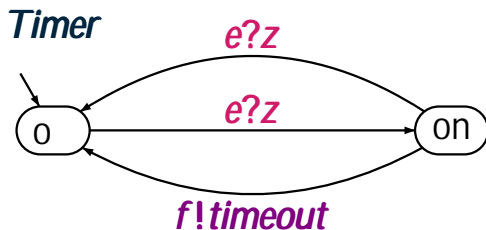


Alternating bit protocol (ABP)

pc2.2-34

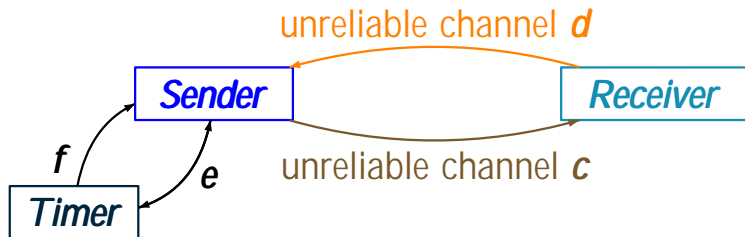


channel system: **Sender** / **Timer** / **Receiver**



actions of **Sender**:

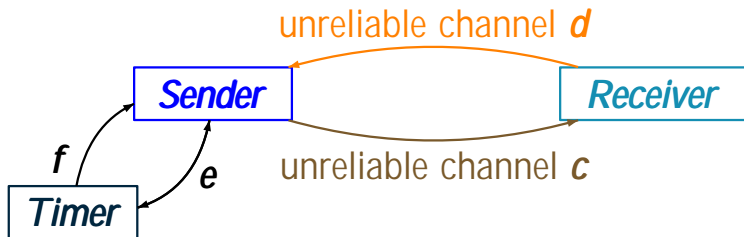
\vdots
 $e!timer_on$
 $e!timer_o$
 $f?z$
 \vdots



specify the **sender** by a program graph using

$\not\hookrightarrow$ asynchronous channels **c** and **d**

\hookrightarrow synchronous channels **e** and **f**

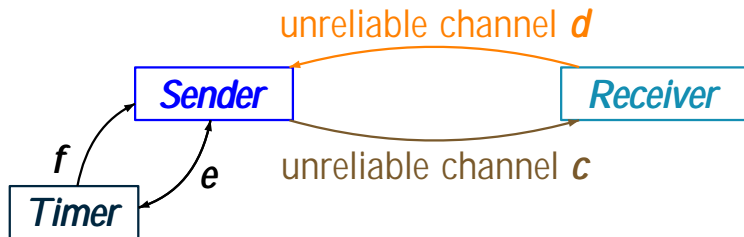


specify the **sender** by a program graph using

ℓ_{oo} asynchronous channels **c** and **d**

ℓ_{oo} synchronous channels **e** and **f**

simply write **!timeout**
?timer_on
?timer_o

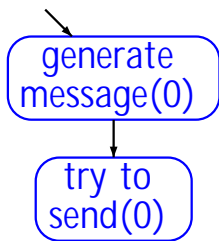


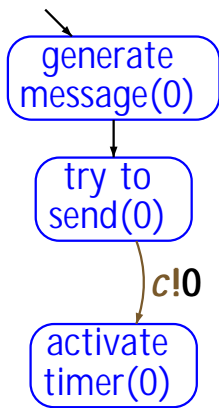
specify the **sender** by a program graph using

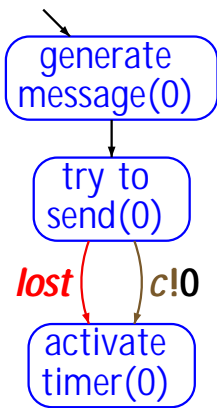
- asynchronous channels **c** and **d**

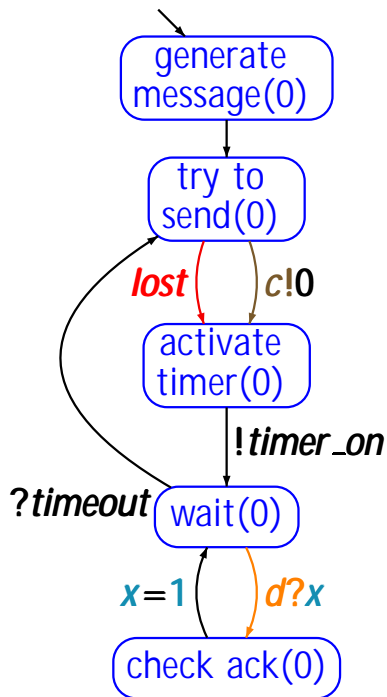
- synchronous channels **e** and **f**

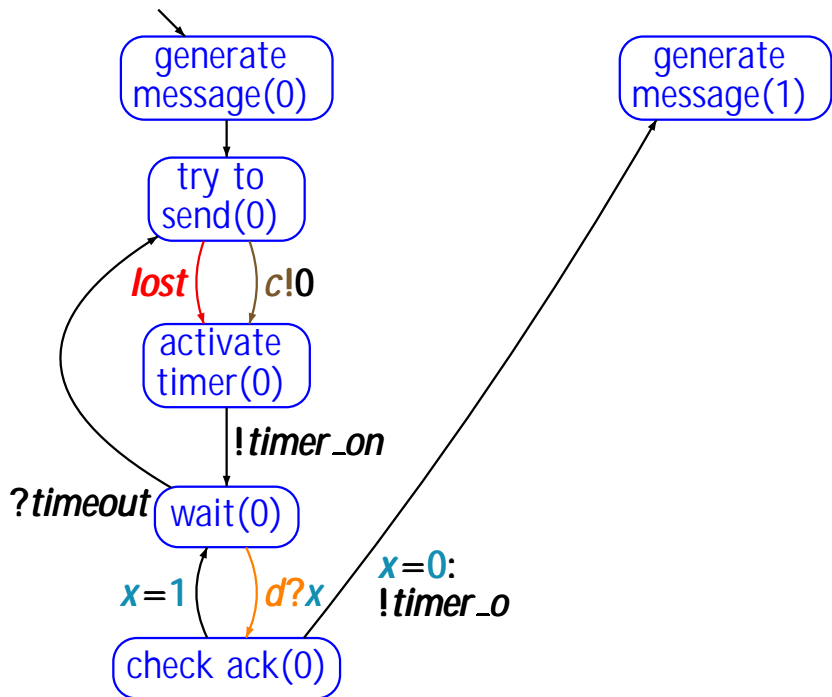
- Boolean variable **x** for the acknowledgement bit sent by the receiver

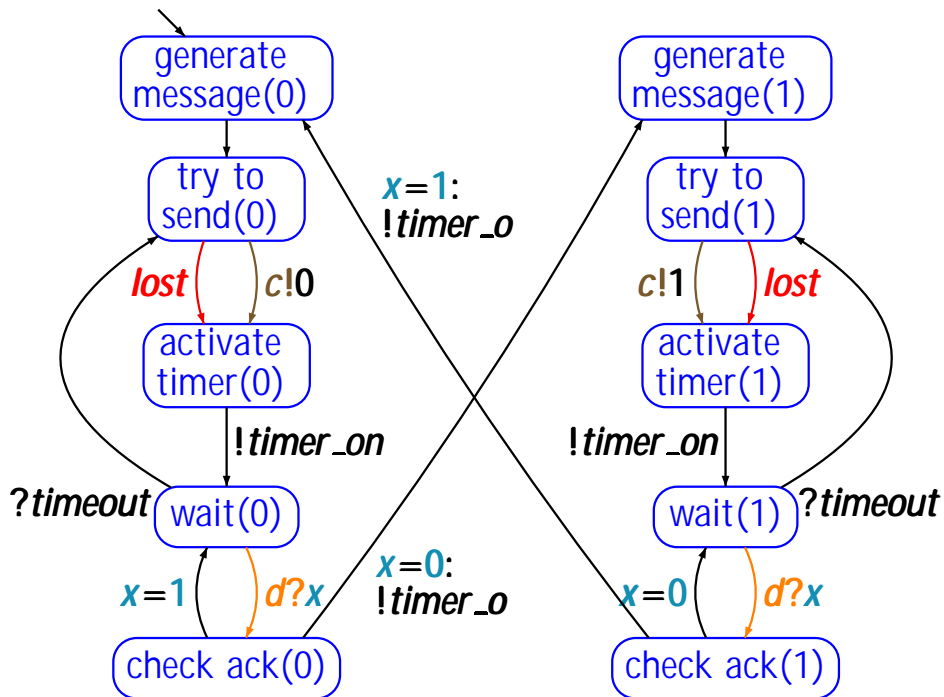






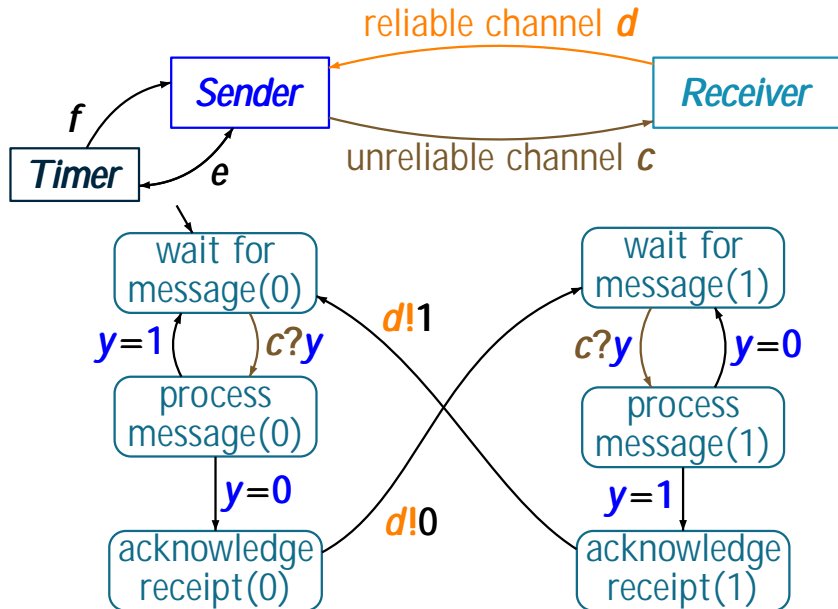


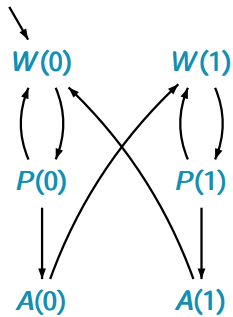
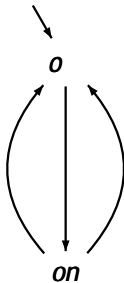
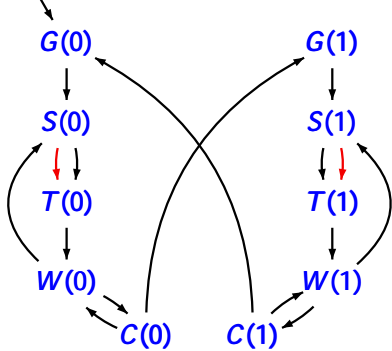


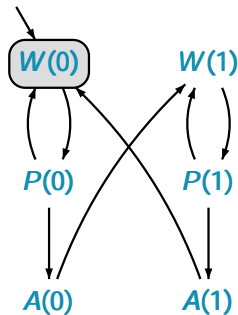
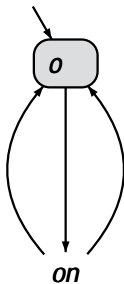
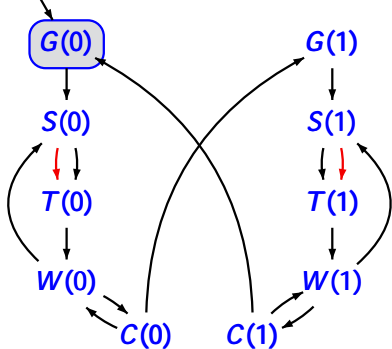


Program graph for the receiver

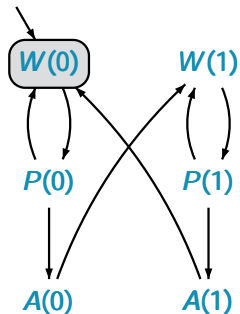
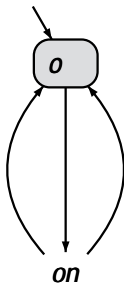
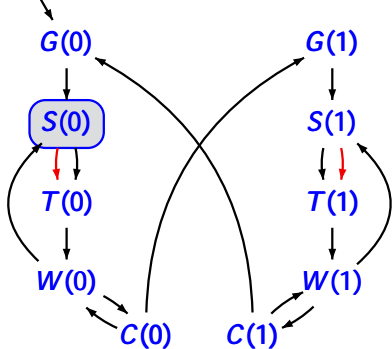
pc2.2-36



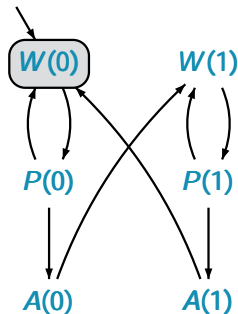
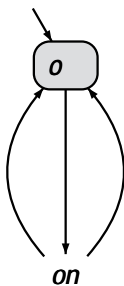
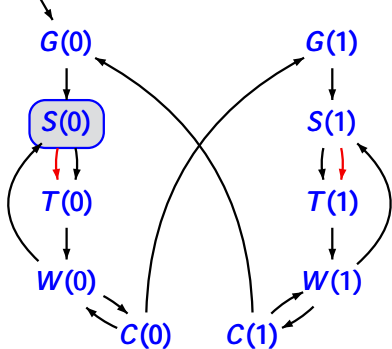




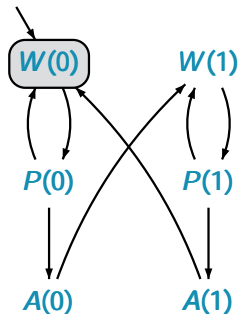
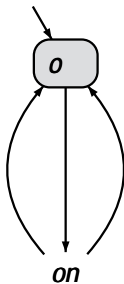
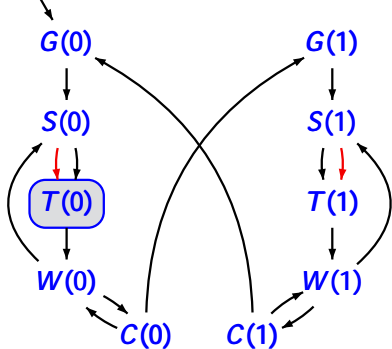
Generate(0) 0 Wait(0) $c =$ $d =$



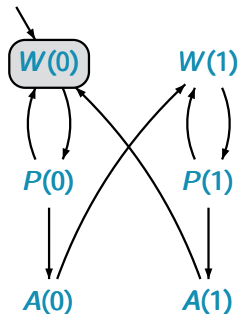
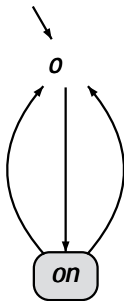
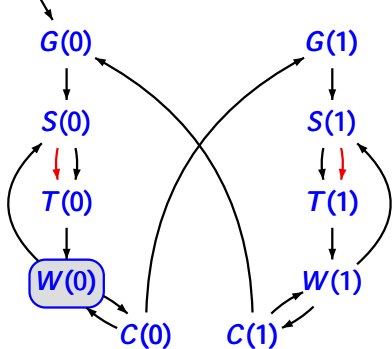
Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$



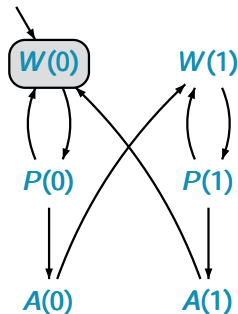
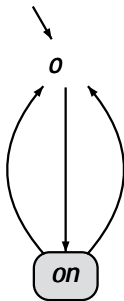
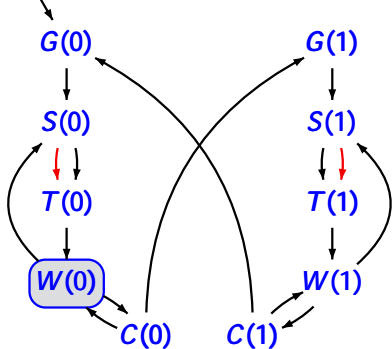
Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message lost



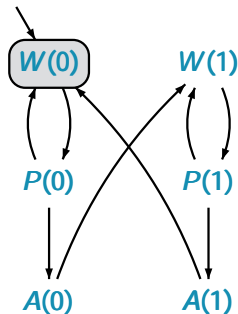
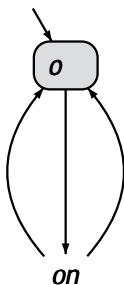
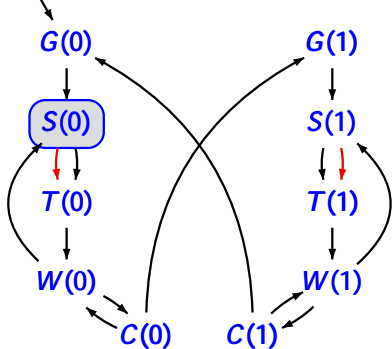
Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message lost
Timer_on(0)	0	Wait(0)	$c =$	$d =$	



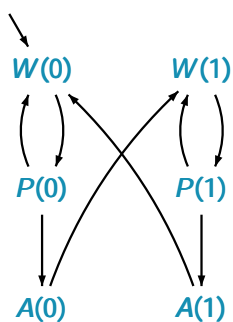
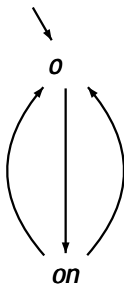
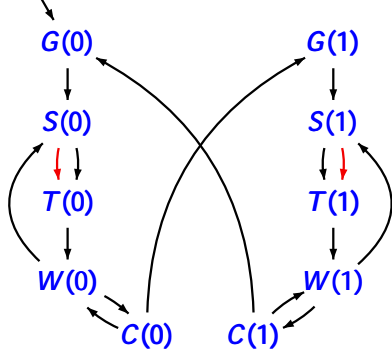
Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message lost
Timer_on(0)	0	Wait(0)	$c =$	$d =$	
Wait(0)	on	Wait(0)	$c =$	$d =$	

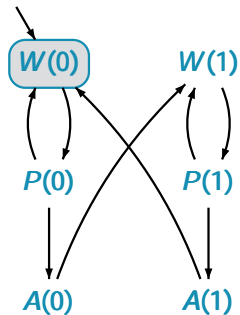
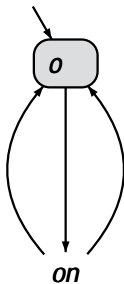
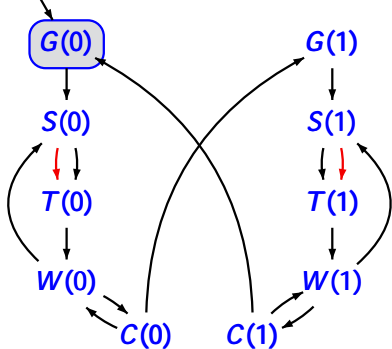


Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message lost
Timer_on(0)	0	Wait(0)	$c =$	$d =$	
Wait(0)	on	Wait(0)	$c =$	$d =$	timeout

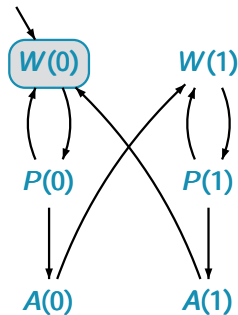
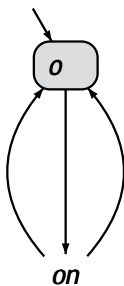
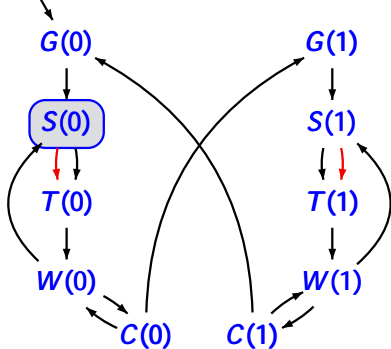


Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message lost
Timer_on(0)	0	Wait(0)	$c =$	$d =$	
Wait(0)	on	Wait(0)	$c =$	$d =$	timeout
Send(0)	0	Wait(0)	$c =$	$d =$	
	\vdots				try again

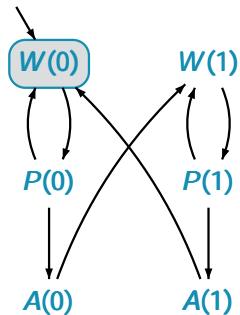
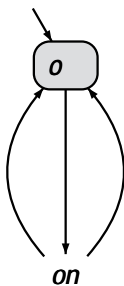
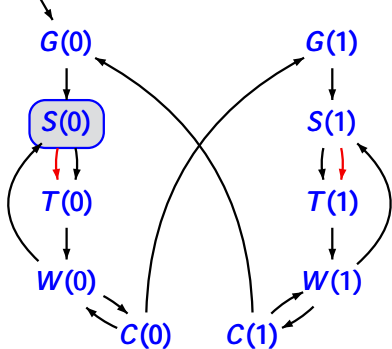




Generate(0) 0 Wait(0) $c =$ $d =$



Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$



Generate(0)
Send(0)

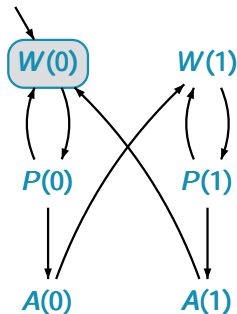
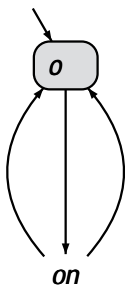
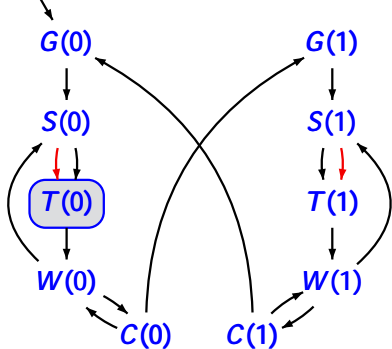
0
0

Wait(0)
Wait(0)

c =
c =

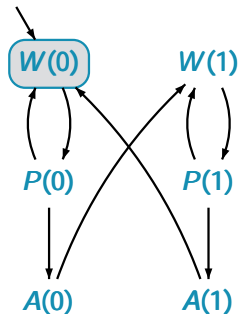
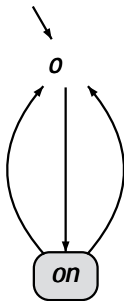
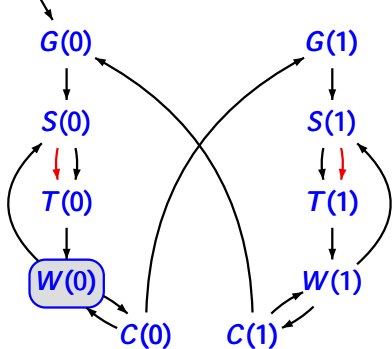
d =
d =

message 0 sent



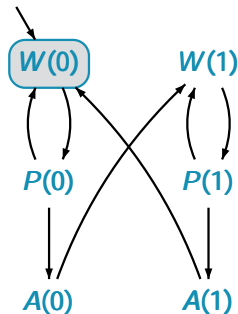
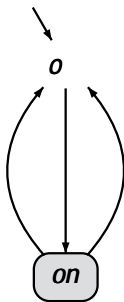
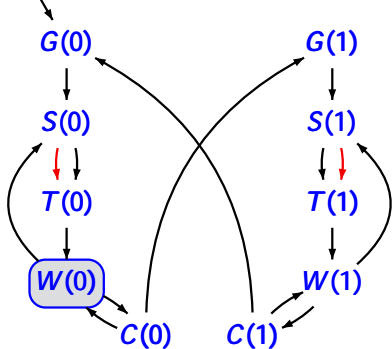
Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$

message **0** sent

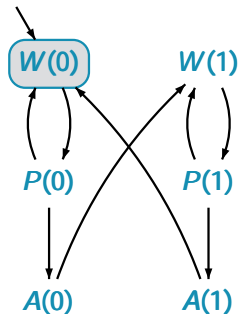
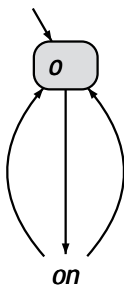
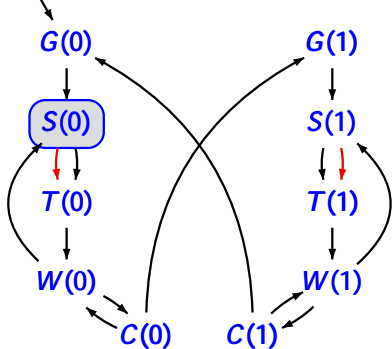


Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$
Wait(0)	on	Wait(0)	$c = 0$	$d =$

message **0** sent



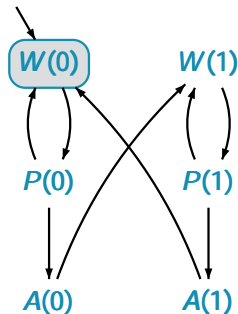
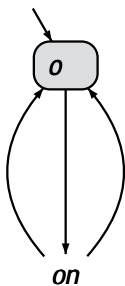
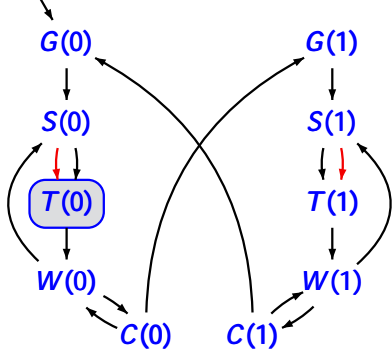
Generate(0)	0	Wait(0)	$c =$	$d =$	
Send(0)	0	Wait(0)	$c =$	$d =$	message 0 sent
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$	
Wait(0)	on	Wait(0)	$c = 0$	$d =$	timeout



Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$
Wait(0)	on	Wait(0)	$c = 0$	$d =$
Send(0)	0	Wait(0)	$c = 0$	$d =$

message **0** sent

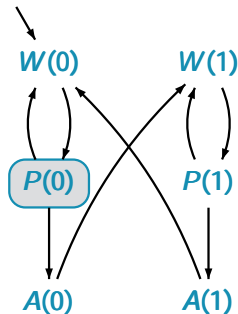
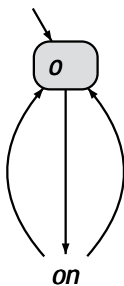
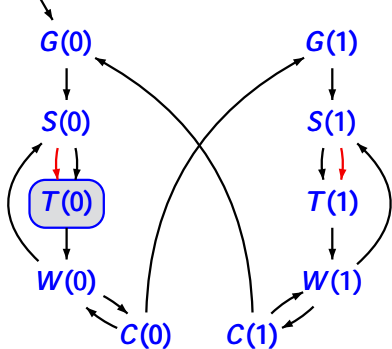
timeout



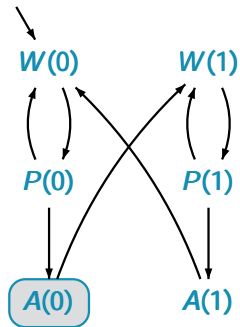
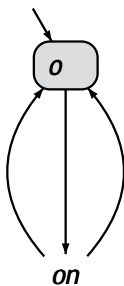
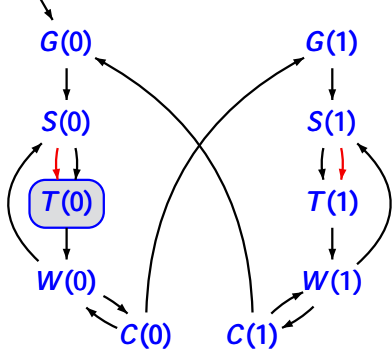
Generate(0)	0	Wait(0)	<i>c</i> =	<i>d</i> =
Send(0)	0	Wait(0)	<i>c</i> =	<i>d</i> =
Timer_on(0)	0	Wait(0)	<i>c</i> =0	<i>d</i> =
Wait(0)	on	Wait(0)	<i>c</i> =0	<i>d</i> =
Send(0)	0	Wait(0)	<i>c</i> =0	<i>d</i> =
Timer_on(0)	0	Wait(0)	<i>c</i> =00	<i>d</i> =

message 0 sent

timeout
0 sent again



Generate(0)	0	Wait(0)	<i>c</i> =	<i>d</i> =	
Send(0)	0	Wait(0)	<i>c</i> =	<i>d</i> =	message 0 sent
Timer_on(0)	0	Wait(0)	<i>c</i> =0	<i>d</i> =	
Wait(0)	on	Wait(0)	<i>c</i> =0	<i>d</i> =	timeout
Send(0)	0	Wait(0)	<i>c</i> =0	<i>d</i> =	0 sent again
Timer_on(0)	0	Wait(0)	<i>c</i> =00	<i>d</i> =	
Timer_on(0)	0	Proc(0)	<i>c</i> =0	<i>d</i> =	message received

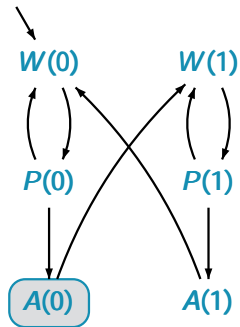
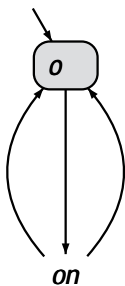
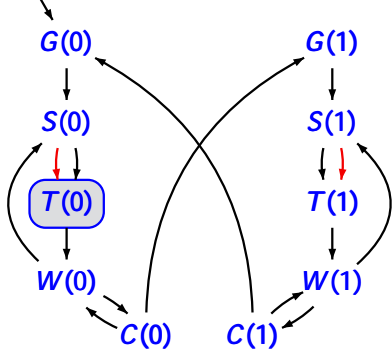


Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$
Wait(0)	on	Wait(0)	$c = 0$	$d =$
Send(0)	0	Wait(0)	$c = 0$	$d =$
Timer_on(0)	0	Wait(0)	$c = 00$	$d =$
Timer_on(0)	0	Proc(0)	$c = 0$	$d =$
Timer_on(0)	0	Ack(0)	$c = 0$	$d =$

message 0 sent

timeout
0 sent again

message received

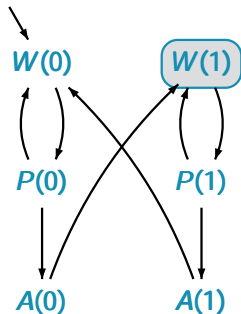
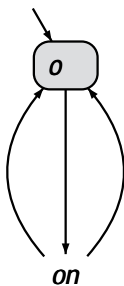
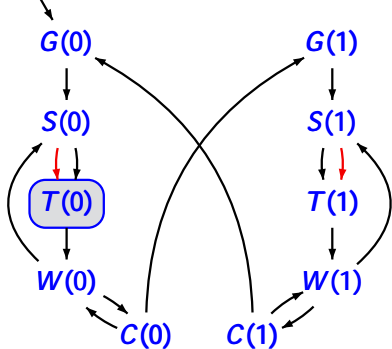


Generate(0)	0	Wait(0)	$c =$	$d =$
Send(0)	0	Wait(0)	$c =$	$d =$
Timer_on(0)	0	Wait(0)	$c = 0$	$d =$
Wait(0)	on	Wait(0)	$c = 0$	$d =$
Send(0)	0	Wait(0)	$c = 0$	$d =$
Timer_on(0)	0	Wait(0)	$c = 00$	$d =$
Timer_on(0)	0	Proc(0)	$c = 0$	$d =$
Timer_on(0)	0	Ack(0)	$c = 0$	$d =$

message 0 sent

timeout
0 sent again

message received
send ack via d



⋮
 Wait(0)
 Send(0)
 Timer_on(0)
 Timer_on(0)
 Timer_on(0)
 Timer_on(0)

⋮
 on
 0
 0
 0
 0
 0

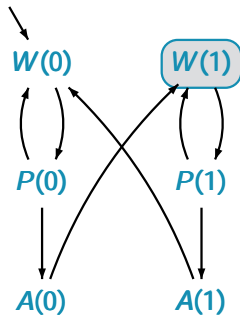
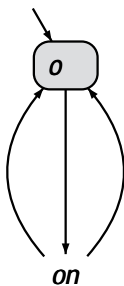
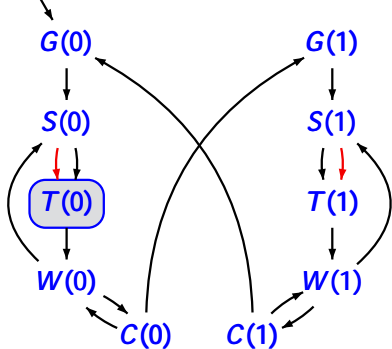
⋮
 Wait(0)
 Wait(0)
 Wait(0)
 Proc(0)
 Ack(0)
 Wait(1)

⋮
 $c=0$
 $c=0$
 $c=00$
 $c=0$
 $c=0$
 $c=0$

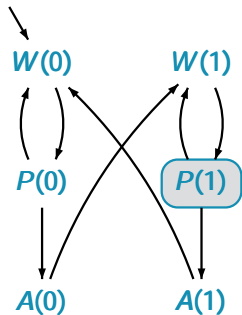
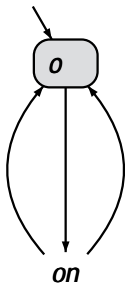
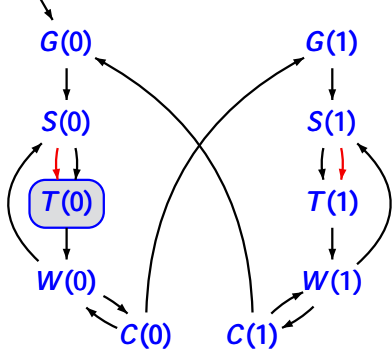
⋮
 $d=$
 $d=$
 $d=$
 $d=$
 $d=$
 $d=0$

timeout
 0 sent again

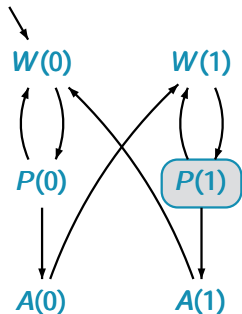
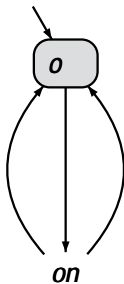
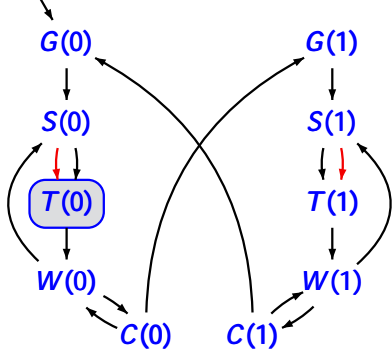
message received
 send ack via d
 receiver changes
 its mode



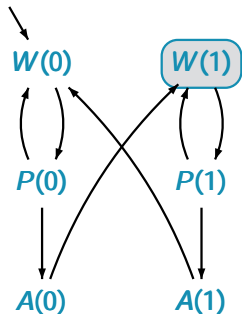
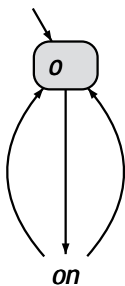
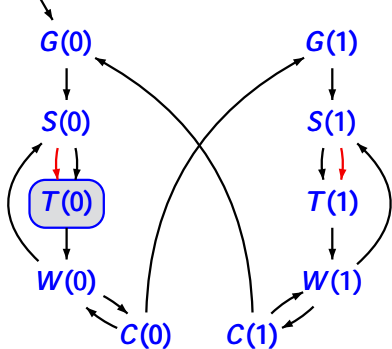
\vdots
 Timer_on(0) 0 \vdots Wait(1) \vdots $c=0$ $d=0$



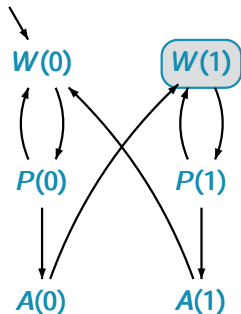
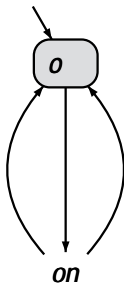
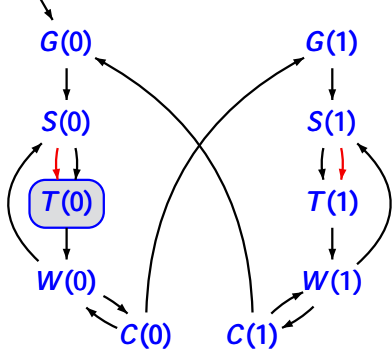
⋮	⋮	⋮	⋮	⋮	receiver reads the same message again
Timer_on(0)	o	Wait(1)	$c=0$	$d=0$	
Timer_on(0)	o	Proc(1)	$c=$	$d=0$	



⋮	⋮	⋮	⋮	⋮	receiver reads the same message again
Timer_on(0)	0	Wait(1)	$c=0$	$d=0$	
⋮	⋮	⋮	⋮	⋮	receiver discards the message
Timer_on(0)	0	Proc(1)	$c=$	$d=0$	



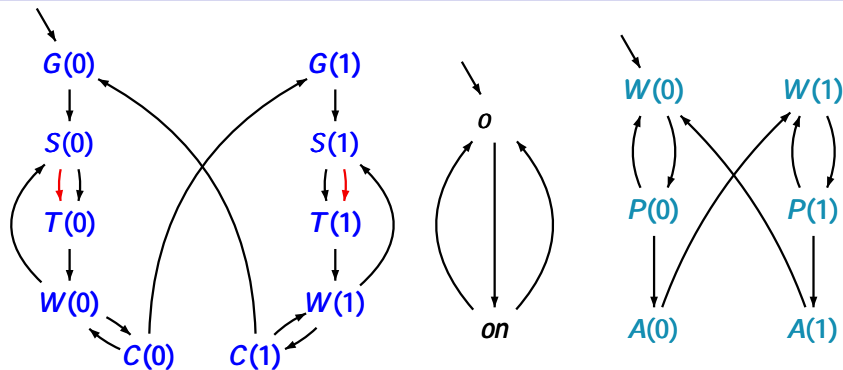
⋮	⋮	⋮	⋮	⋮	
Timer_on(0)	0	Wait(1)	$c=0$	$d=0$	receiver reads the
Timer_on(0)	0	Proc(1)	$c=$	$d=0$	same message again
Timer_on(0)	0	Wait(1)	$c=$	$d=0$	receiver discards
					the message



⋮	⋮	⋮	⋮	⋮	
Timer_on(0)	0	Wait(1)	$c=0$	$d=0$	receiver reads the same message again
Timer_on(0)	0	Proc(1)	$c=$	$d=0$	receiver discards the message
Timer_on(0)	0	Wait(1)	$c=$	$d=0$	
⋮	⋮	⋮	⋮	⋮	

Alternating bit protocol (ABP)

pc2.2-37c

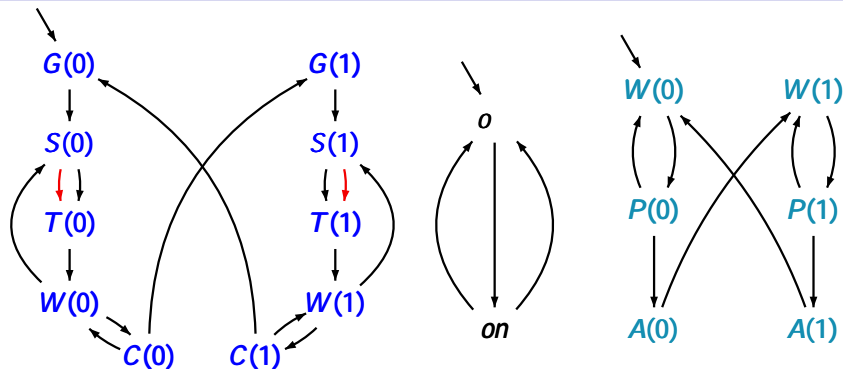


number of states in the TS:

10 ~~20~~ ~~20~~ ~~20~~ ~~20~~ channel evaluations

Alternating bit protocol (ABP)

pc2.2-37c



number of states in the TS:

10 ~~P_{do}~~ ~~P_{ob}~~ ~~P_{off}~~ channel evaluations

$> 10^8$ for FIFOs with capacity 10

$\not\in$ conditional communication actions

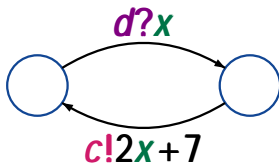
$g:c?x$

ℓ_{oo} conditional communication actions

$g:c?x$

ℓ_{oo} generalized sending instructions $c!expr$ instead of $c!v$

e.g.

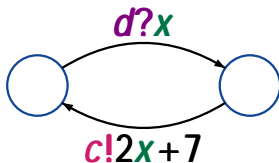


ℙ_{oo} conditional communication actions

$g:c?x$

ℙ_{oo} generalized sending instructions $c!expr$ instead of $c!v$

e.g.



ℙ_{oo} communication as conditions

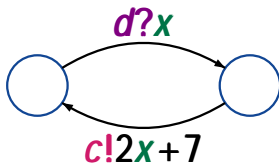
$c?x:$

ℳ conditional communication actions

$g:c?x$

ℳ generalized sending instructions $c!expr$ instead of $c!v$

e.g.



ℳ communication as conditions

$c?x:$

more compact TS-representations

- conditional communication actions $g:c?x$
- generalized sending instructions $c!expr$ instead of $c!v$
- communication as conditions $c?x:$
- open channel systems $P_1 / \dots / P_n$
instead of closed channel systems $[P_1 / \dots / P_n]$

ℳ conditional communication actions

$g:c?x$

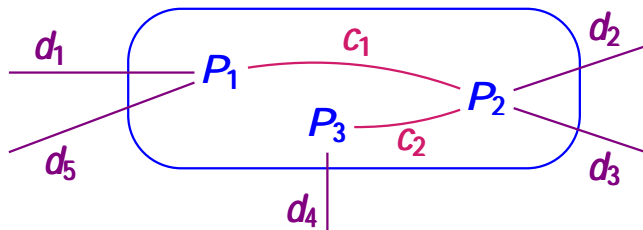
ℳ generalized sending instructions $c!expr$ instead of $c!v$

ℳ communication as conditions

$c?x:$

ℳ open channel systems $P_1 / \dots / P_n$

instead of closed channel systems $[P_1 / \dots / P_n]$



(*pure*) interleaving for TS $T_1 \parallel T_2$

$\not\propto$ only concurrency, no communication

$\not\propto$ not applicable for competing systems

(pure) interleaving for TS $T_1 \parallel T_2$

$\not\propto$ only concurrency, no communication

$\not\propto$ not applicable for competing systems

synchronous message passing for TS $T_1 \text{ Syn } T_2$

$\not\propto$ interleaving for concurrent actions

$\not\propto$ synchronization via actions in *Syn*

(pure) interleaving for TS $T_1 \parallel T_2$

$\not\propto$ only concurrency, no communication

$\not\propto$ not applicable for competing systems

synchronous message passing for TS $T_1 \text{ Syn } T_2$

$\not\propto$ interleaving for concurrent actions

$\not\propto$ synchronization via actions in **Syn**

interleaving for program graphs $P_1 \parallel P_2$

$\not\propto$ interleaving for concurrent actions

$\not\propto$ communication via shared variables

(pure) interleaving for TS $T_1 \parallel T_2$

⊄ only concurrency, no communication

⊄ not applicable for competing systems

synchronous message passing for TS $T_1 \text{ Syn } T_2$

⊄ interleaving for concurrent actions

⊄ synchronization via actions in **Syn**

interleaving for program graphs $P_1 \parallel P_2$

⊄ interleaving for concurrent actions

⊄ communication via shared variables

channel systems: open $P_1 / \dots / P_n$ or closed $[P_1 / \dots / P_n]$

⊄ interleaving, shared variables, message passing

(pure) interleaving for TS $T_1 \parallel T_2$

⊂ only concurrency, no communication

synchronous message passing for TS $T_1 \text{ syn } T_2$

⊂ interleaving, synchronization via actions in **Syn**

interleaving for program graphs $P_1 \parallel P_2$

⊂ interleaving, shared variables

channel systems: open $P_1 / \dots / P_n$ or closed $[P_1 / \dots / P_n]$

⊂ interleaving, shared variables

⊂ synchronous and asynchronous message passing

synchronous product for TS $T_1 \text{ syn } T_2$

⊂ no interleaving, pure synchronization

for parallel systems with fully synchronized processes

$$\begin{aligned} T_1 &= (S_1, Act_1, \%_1, \dots) \\ T_2 &= (S_2, Act_2, \%_2, \dots) \end{aligned} \quad \text{two TS}$$

synchronous product:

$$T_1 \quad T_2 = (S_1 \% S_2, Act, \% , \dots)$$

for parallel systems with fully synchronized processes

$$\begin{aligned} T_1 &= (S_1, Act_1, \%_1, \dots) \\ T_2 &= (S_2, Act_2, \%_2, \dots) \end{aligned} \quad \text{two TS}$$

synchronous product:

$$T_1 \parallel T_2 = (S_1 \% S_2, Act, \% , \dots)$$

where the action set Act is given by a function

$$Act_1 \% Act_2 \% Act, (,)$$

action name for the concurrent
execution of and

for parallel systems with fully synchronized processes

$$\begin{aligned} T_1 &= (S_1, Act_1, \%_1, \dots) \\ T_2 &= (S_2, Act_2, \%_2, \dots) \end{aligned} \quad \text{two TS}$$

synchronous product:

$$T_1 \quad T_2 = (S_1 \% S_2, Act, \% , \dots)$$

where the action set Act is given by a function

$$Act_1 \% Act_2 \% Act, (,)$$

action name for the concurrent
execution of and

if action names are irrelevant: $Act_1 = Act_2 = Act = \{ \}$

for parallel systems with fully synchronized processes

$$\begin{aligned} T_1 &= (S_1, Act_1, \rightarrow_1, \dots) \\ T_2 &= (S_2, Act_2, \rightarrow_2, \dots) \end{aligned} \quad \text{two TS}$$

synchronous product:

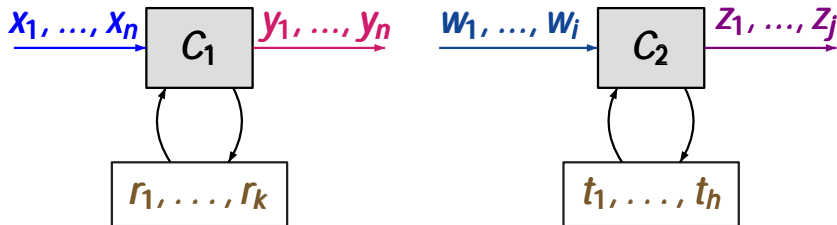
$$T_1 \quad T_2 = (S_1 \rightarrow S_2, Act, \rightarrow, \dots)$$

transition relation \rightarrow :

$$\frac{S_1 \rightarrow_1 S_1 \quad S_2 \rightarrow_2 S_2}{S_1, S_2 \rightarrow S_1, S_2}$$

Synchronous product for composing circuits

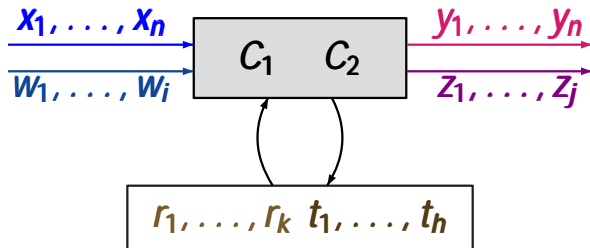
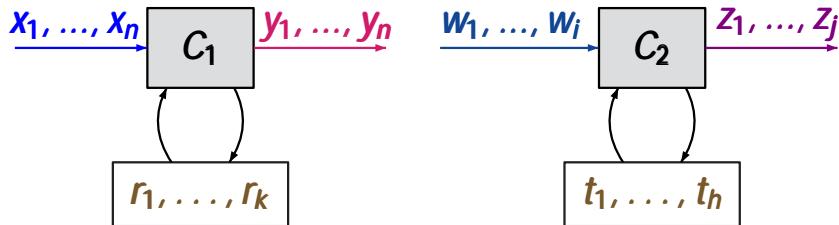
pc2.2-40



2 sequential circuits

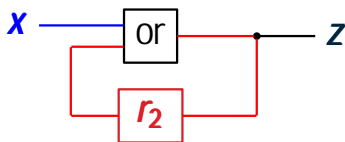
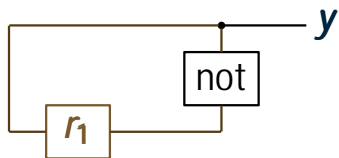
Synchronous product for composing circuits

pc2.2-40



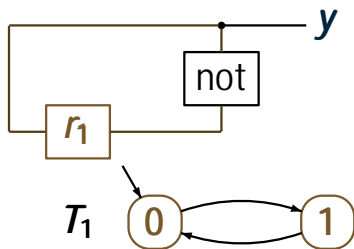
Synchronous product: example

pc2.2-52



Synchronous product: example

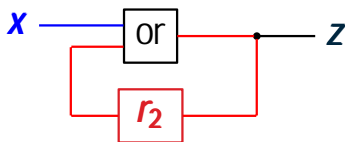
pc2.2-52



initially:
 $r_1 = 0$

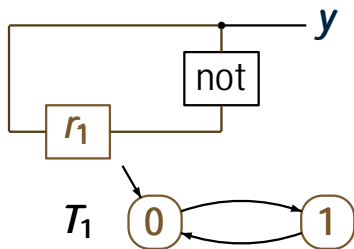
transition function:

$$r_1 = \neg r_1$$



Synchronous product: example

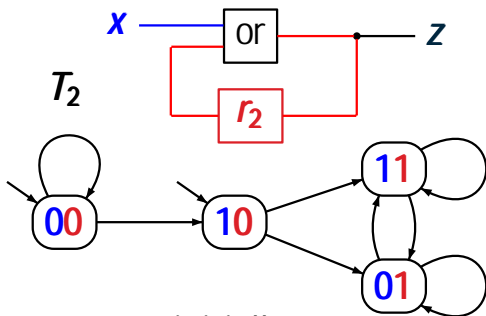
pc2.2-52



initially:
 $r_1 = 0$

transition function:

$$r_1 = \neg r_1$$



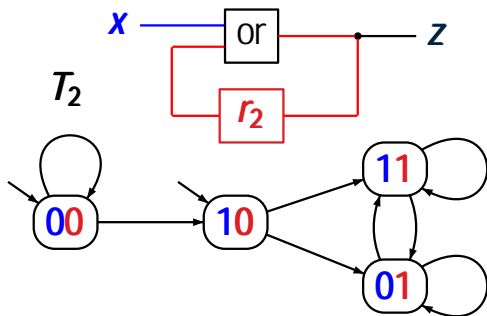
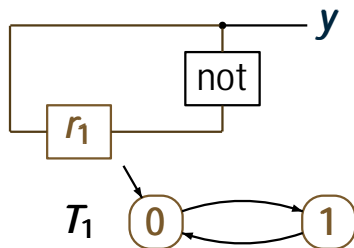
initially:
 $r_2 = 0$

transition function:

$$r_2 = r_2 \vee x$$

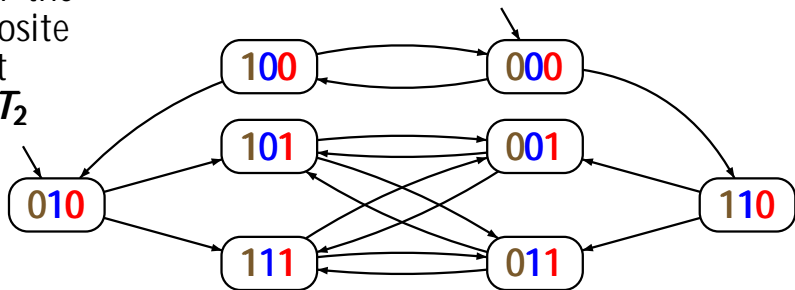
Synchronous product: example

pc2.2-52



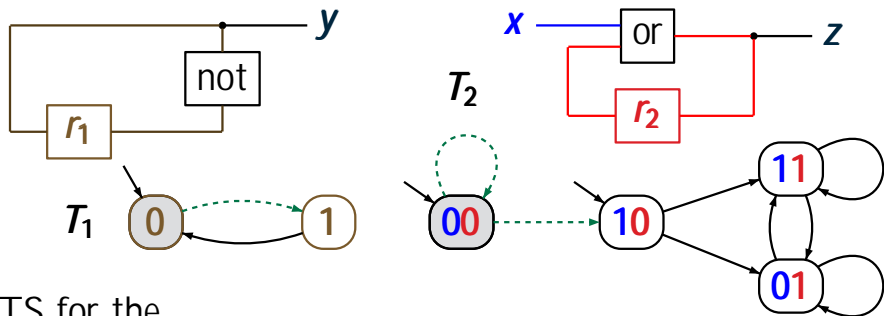
TS for the
composite
circuit

T_1 T_2



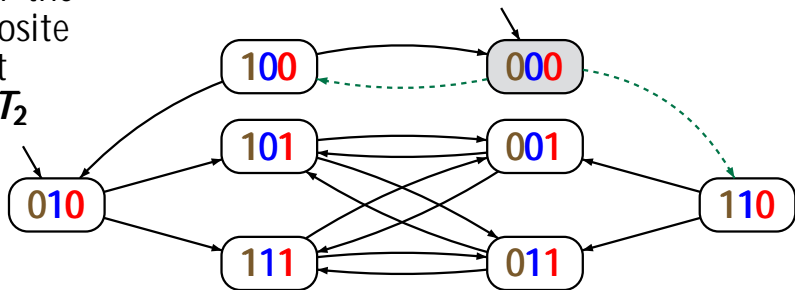
Synchronous product: example

pc2.2-52



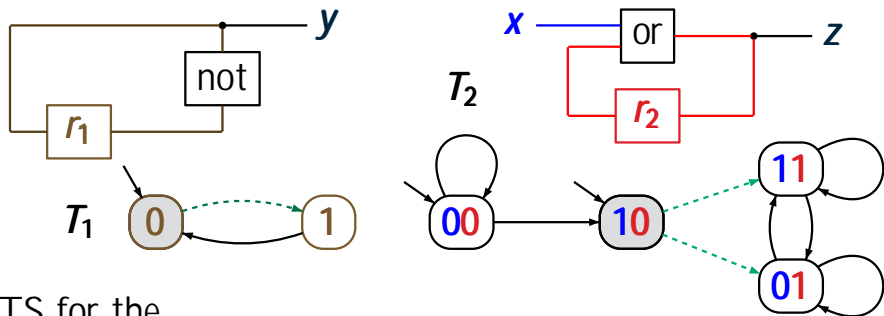
TS for the
composite
circuit

T_1 T_2



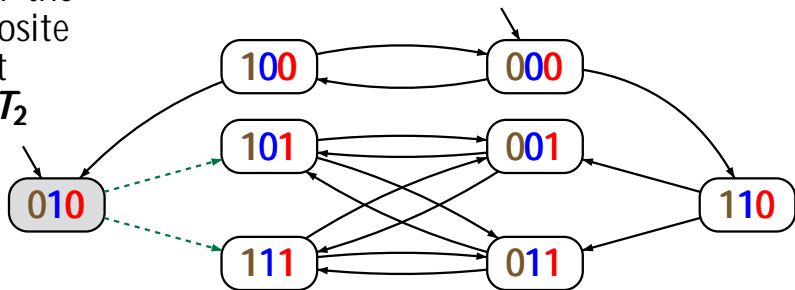
Synchronous product: example

pc2.2-52



TS for the
composite
circuit

T_1 T_2



TS for reactive systems can be enormously large

TS for reactive systems can be enormously large

~~Too~~ ~~infinite~~ for systems with

variables of infinite domains, e.g., \mathbb{N}

infinite data structures, e.g., stacks, queues, lists,...

TS for reactive systems can be enormously large

~~Too~~ ~~large~~ for systems with

variables of infinite domains, e.g., N

infinite data structures, e.g., stacks, queues, lists,...

~~Too~~ if ~~large~~: **exponential growth** in

TS for reactive systems can be enormously large

~~Too~~ ~~infinite~~ for systems with

~~Too~~ variables of infinite domains, e.g., \mathbb{N}

~~Too~~ infinite data structures, e.g., stacks, queues, lists,...

~~Too~~ if ~~Too~~: **exponential growth** in

number of parallel components,

e.g., state space of $T_1 \dots T_n$ is $S_1 \dots S_n$

TS for reactive systems can be enormously large

~~Exponential~~ ~~infinite~~ for systems with

variables of infinite domains, e.g., \mathbb{N}

infinite data structures, e.g., stacks, queues, lists,...

~~Exponential~~ ~~infinite~~: **exponential growth** in

number of parallel components,

e.g., state space of $T_1 \dots T_n$ is $S_1 \dots S_n$

number of variables and channels

State explosion problem

pc2.2-42

TS for reactive systems can be enormously large

~~Exponential~~ ~~infinite~~ for systems with

variables of infinite domains, e.g., \mathbb{N}

infinite data structures, e.g., stacks, queues, lists,...

~~Exponential~~ ~~infinite~~ if ~~infinite~~: **exponential growth** in

number of parallel components,

e.g., state space of $T_1 \dots T_n$ is $S_1 \dots S_n$

number of variables and channels

e.g., for channel systems: size of the state space is

$$\prod_{x \text{ Var}} |Loc_1| \dots |Loc_n| \prod_{c \text{ Chan}} |Dom(x)| \prod_{c \text{ Chan}} |Dom(c)|^{cap(c)}$$

