

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

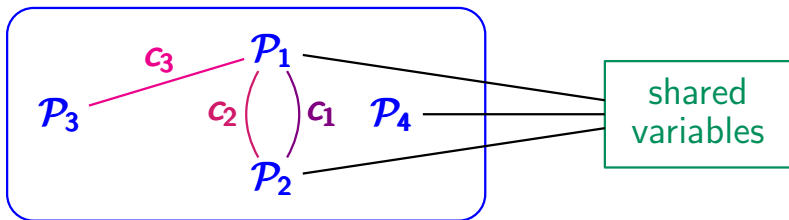
Equivalences and Abstraction

representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**

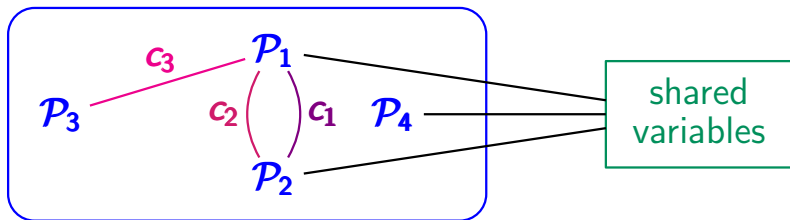
representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**



representation of data-dependent parallel systems with

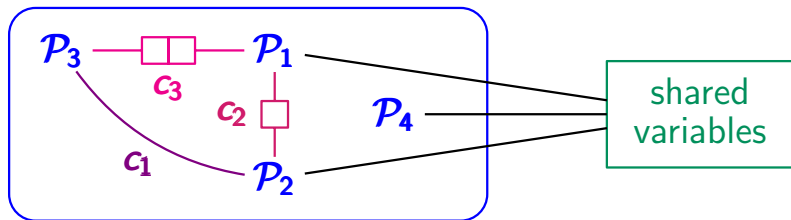
- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**



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

representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**

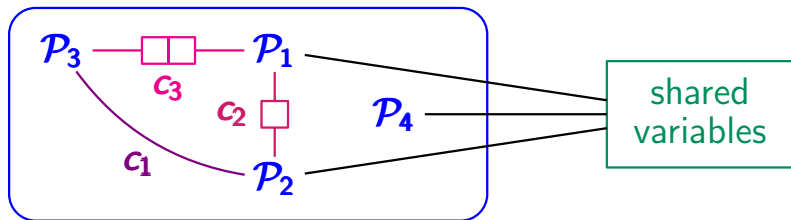


channel types: **synchronous** or **FIFO**

↑
capacity $\hat{=}$ number of buffer cells

representation of data-dependent parallel systems with

- communication over **shared variables**
- **synchronous** message passing \leftarrow **capacity 0**
- **asynchronous** message passing \leftarrow **capacity ≥ 1**



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

capacity 0

capacity

$\hat{=}$ number of buffer cells

representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**

formalization through **program graphs** for $\mathcal{P}_1, \dots, \mathcal{P}_n$

representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**

formalization through **program graphs** for $\mathcal{P}_1, \dots, \mathcal{P}_n$

- with conditional transitions $\ell_i \xrightarrow{g:\alpha} \ell'_i$ (as before)

representation of data-dependent parallel systems with

- communication over **shared variables**
 - **synchronous** message passing
 - **asynchronous** message passing
- } communication over **channels**

formalization through **program graphs** for $\mathcal{P}_1, \dots, \mathcal{P}_n$

- with conditional transitions $\ell_i \xrightarrow{g:\alpha} \ell'_i$ (as before)
- and **communication actions**

$\ell_i \xrightarrow{c!v} \ell'_i$	sending value v via channel c
$\ell_i \xrightarrow{c?x} \ell'_i$	receiving a value for variable x via channel c

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

evaluation for a set Var of typed variables:

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

↑
i.e., $\eta(x) \in Dom(x)$

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

evaluation for a set Var of typed variables:

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

\uparrow
i.e., $\eta(x) \in Dom(x)$

typed channel: channel c with

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

evaluation for a set $Chan$ of typed channels:

type-consistent function $\xi : Chan \rightarrow Values^*$

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

$[\mathcal{P}_1 | \mathcal{P}_2 | \dots | \mathcal{P}_n]$ where \mathcal{P}_i are program graphs

$[\mathcal{P}_1 \mid \mathcal{P}_2 \mid \dots \mid \mathcal{P}_n]$ where \mathcal{P}_i are program graphs
over a pair (*Var*, *Chan*)

$[\mathcal{P}_1 \mid \mathcal{P}_2 \mid \dots \mid \mathcal{P}_n]$ where \mathcal{P}_i are program graphs
over a pair $(\text{Var}, \text{Chan})$

Var set of typed variables

Chan set of typed channels with
capacities $\text{cap}(\cdot)$ and domains $\text{Dom}(\cdot)$

$[\mathcal{P}_1 \mid \mathcal{P}_2 \mid \dots \mid \mathcal{P}_n]$ where \mathcal{P}_i are program graphs over a pair $(\text{Var}, \text{Chan})$

Var set of typed variables

Chan set of typed channels with capacities $\text{cap}(\cdot)$ and domains $\text{Dom}(\cdot)$

program graphs $\mathcal{P}_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \hookrightarrow_i, \text{Loc}_{0,i}, g_0)$ with conditional transitions

$\ell \xrightarrow{g:\alpha}_i \ell'$ guarded command

$\ell \xrightarrow{c!v}_i \ell'$ sending value v via channel c

$\ell \xrightarrow{c?x}_i \ell'$ receiving a value for variable x via channel c

asynchronous message passing via channels of capacity ≥ 1

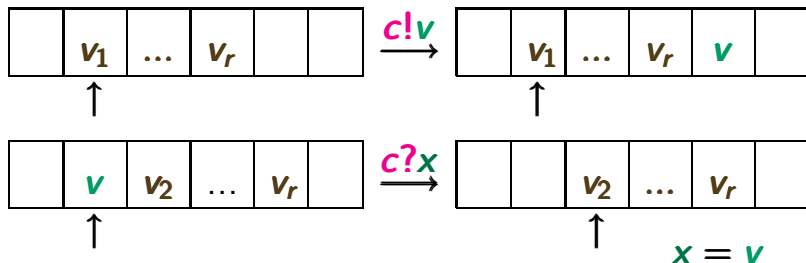
	enabled if ...	effect
sending $c!v$		
receiving $c?x$		

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 $v = front(c)$	$x := v$ $remove(c)$



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 $v = front(c)$	$x := v$ $remove(c)$

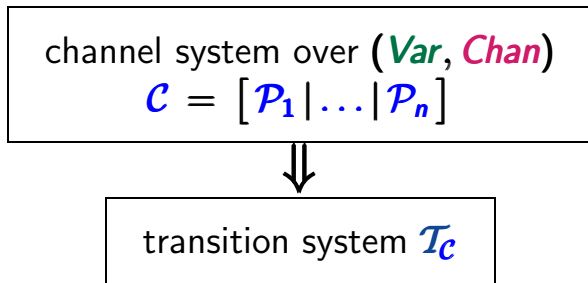
synchronous message passing via channels of capacity 0

- $c!v$ and $c?x$ are executed at the same time
- effect $x := v$

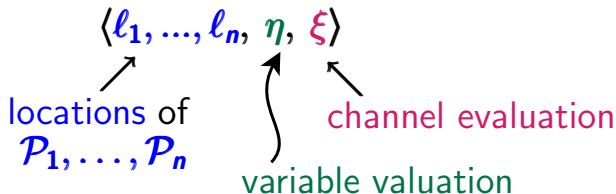
channel system over (*Var*, *Chan*)
 $\mathcal{C} = [\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n]$



transition system $\mathcal{T}_{\mathcal{C}}$



states of $\mathcal{T}_{\mathcal{C}}$ have the form



states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where

ℓ_i location of program graph \mathcal{P}_i ,

$\eta \in \text{Eval}(\text{Var})$ variable evaluation

$\xi \in \text{Eval}(\text{Chan})$ channel evaluation

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where

ℓ_i location of program graph \mathcal{P}_i ,

$\eta \in \text{Eval}(\text{Var})$ variable evaluation

$\xi \in \text{Eval}(\text{Chan})$ channel evaluation

variable evaluation:

$$\eta : \text{Var} \longrightarrow \bigcup_{x \in \text{Var}} \text{Dom}(x) \quad \text{with } \eta(x) \in \text{Dom}(x)$$

channel evaluation:

$$\xi : \text{Chan} \longrightarrow \bigcup_{c \in \text{Chan}} \text{Dom}(c)^* \quad \text{with } \xi(c) \in \text{Dom}(c)^* \\ \text{and } |\xi(c)| \leq \text{cap}(c)$$

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where

ℓ_i location of program graph \mathcal{P}_i ,


$\eta \in \text{Eval}(\text{Var})$ variable evaluation

$\xi \in \text{Eval}(\text{Chan})$ channel evaluation

variable evaluation:

$$\eta : \text{Var} \longrightarrow \bigcup_{x \in \text{Var}} \text{Dom}(x) \quad \text{with } \eta(x) \in \text{Dom}(x)$$

channel evaluation:

$$\xi : \text{Chan} \rightarrow \bigcup_{c \in \text{Chan}} \text{Dom}(c)^* \quad \text{with } \xi(c) \in \text{Dom}(c)^* \\ \text{and } |\xi(c)| \leq \text{cap}(c)$$


only channels c with $\text{cap}(c) \geq 1$ are relevant

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval}(\text{Var})$,
 $\xi \in \text{Eval}(\text{Chan})$

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval}(\text{Var})$,
 $\xi \in \text{Eval}(\text{Chan})$

transition relation \longrightarrow is given by SOS-rules:

- interleaving rules for $\alpha \in \text{Act}_i$
- rules for message passing along channels

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval}(\text{Var})$,
 $\xi \in \text{Eval}(\text{Chan})$

transition relation \longrightarrow is given by SOS-rules:

- interleaving rules for $\alpha \in \text{Act}_i$
- rules for message passing along channels

interleaving rule for actions $\alpha \in \text{Act}_i$:

$$\frac{\ell_i \xrightarrow{g:\alpha}_i \ell'_i \wedge \eta \models g}{\langle \ell_1, \dots, \ell_i, \dots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \dots, \ell'_i, \dots, \ell_n, \text{Effect}_i(\alpha, \eta), \xi \rangle}$$

states $\langle \ell_1, \dots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval}(\text{Var})$,
 $\xi \in \text{Eval}(\text{Chan})$

transition relation \longrightarrow is given by SOS-rules:

- interleaving rules for $\alpha \in \text{Act}_i$
- rules for message passing along channels

interleaving rule for actions $\alpha \in \text{Act}_i$:

$$\frac{\ell_i \xrightarrow{g:\alpha}_i \ell'_i \wedge \eta \models g}{\langle \ell_1, \dots, \ell_i, \dots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \dots, \ell'_i, \dots, \ell_n, \text{Effect}_i(\alpha, \eta), \xi \rangle}$$

↑
does not affect the channel evaluation ξ

SOS-rules for asynchronous message passing

PC2.2-29

for channel c with $\text{cap}(c) \geq 1$

SOS-rules for asynchronous message passing

PC2.2-29

for channel c with $\text{cap}(c) \geq 1$

receiving a message:

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge \xi(c) = v_1 v_2 \dots v_k \wedge k \geq 1}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{\tau} \langle l_1, \dots, l'_i, \dots, l_n, \eta', \xi' \rangle}$$

SOS-rules for asynchronous message passing

PC2.2-29

for channel c with $\text{cap}(c) \geq 1$

receiving a message:

$$\frac{\ell_i \xrightarrow{c?x}_i \ell'_i \wedge \xi(c) = v_1 v_2 \dots v_k \wedge k \geq 1}{\langle \ell_1, \dots, \ell_i, \dots, \ell_n, \eta, \xi \rangle \xrightarrow{T} \langle \ell_1, \dots, \ell'_i, \dots, \ell_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$

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

SOS-rules for asynchronous message passing

PC2.2-29

for channel c with $\text{cap}(c) \geq 1$

receiving a message:

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge \xi(c) = v_1 v_2 \dots v_k \wedge k \geq 1}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \dots v_k]$

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

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

for channel c with $\text{cap}(c) \geq 1$

receiving a message:

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge \xi(c) = v_1 v_2 \dots v_k \wedge k \geq 1}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \dots v_k]$

sending a message:

$$\frac{l_i \xrightarrow{c!v}_i l'_i \wedge \xi(c) = v_1 \dots v_k \wedge k < \text{cap}(c)}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l_n, \eta, \xi[c := v_1 \dots v_k v] \rangle}$$

for synchronous channel c :

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge l_j \xrightarrow{c!v}_j l'_j \wedge i \neq j}{\langle l_1, \dots, l_i, \dots, l_j, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l'_j, \dots, l_n, \eta', \xi' \rangle}$$

for synchronous channel c :

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge l_j \xrightarrow{c!v}_j l'_j \wedge i \neq j}{\langle l_1, \dots, l_i, \dots, l_j, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l'_j, \dots, l_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v]$

for synchronous channel c :

$$\frac{l_i \xrightarrow{c?x}_i l'_i \wedge l_j \xrightarrow{c!v}_j l'_j \wedge i \neq j}{\langle l_1, \dots, l_i, \dots, l_j, \dots, l_n, \eta, \xi \rangle \xrightarrow{T} \langle l_1, \dots, l'_i, \dots, l'_j, \dots, l_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x:=v]$ and $\xi' = \xi$

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:

$$2 * 2 * 2 * 2 * (2^{11} - 1) * (2^{11} - 1)$$

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

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:

$$2 * 2 * 2 * 2 * (2^{11} - 1) * (2^{11} - 1) > 2^{24} > 25 \text{ mio}$$

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

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:

$$2 * 2 * 2 * 2 * (2^{11} - 1) * (2^{11} - 1) > 2^{24} > 25 \text{ mio}$$

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

... 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

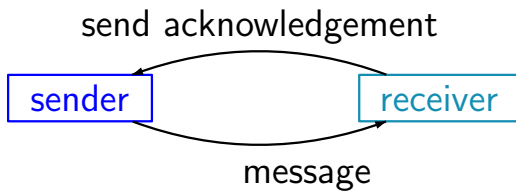
answer:

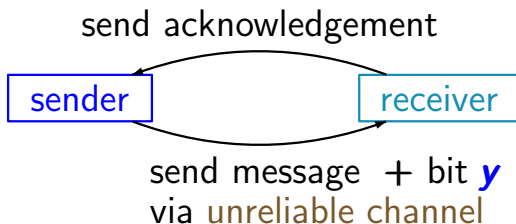
$$2 * 2 * 2 * 2 * (2^{11} - 1) * (2^{11} - 1) > 2^{24} > 25 \text{ mio}$$

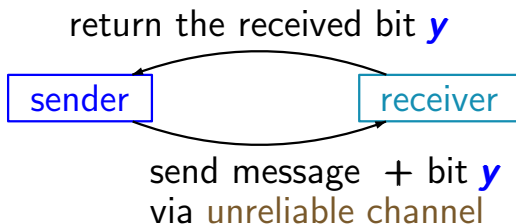
$$\text{note: } 2^{11} - 1 = 1 + 2 + 2^2 + \dots + 2^{10}$$

... with an unbounded channel ?

answer: ∞



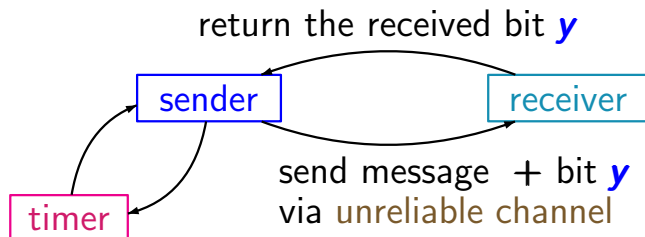


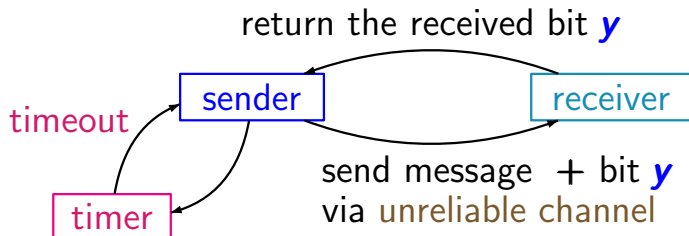


Goal : design a comm. protocol that ensures any distinct transmitted datum by S to be delivered to R

Alternating bit protocol (ABP)

PC2.2-32





LOOP FOREVER

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

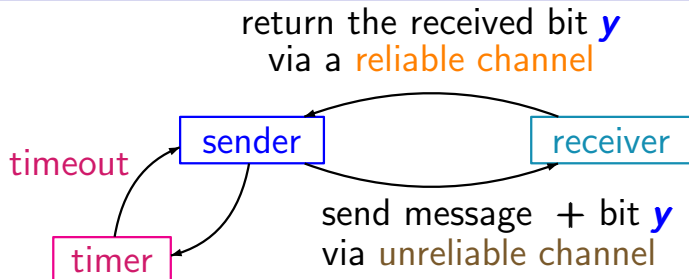
(2) AWAIT **timeout** or **acknowledgement** DO

IF **timeout** THEN goto (1)

ELSE turn off timer; **y** := $\neg y$

FI

OD



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; $y := \neg y$

FI

OD

If both channels are unreliable ...

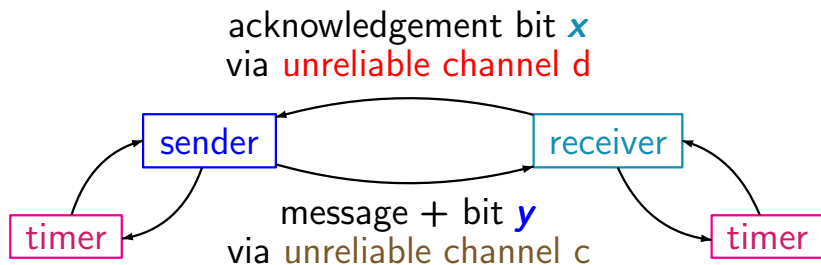
PC2.2-33

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



If both channels are unreliable ...

PC2.2-33

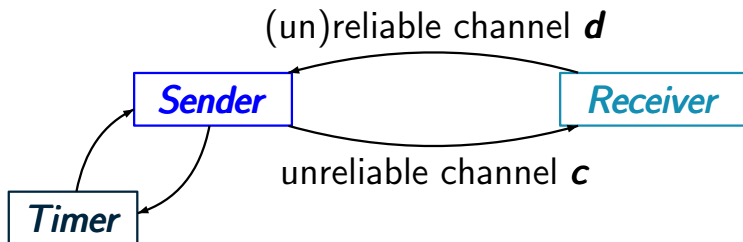


LOOP FOREVER

- (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; $y:=\neg y$
 - ELSE ignore x
- OD FI FI

Alternating bit protocol (ABP)

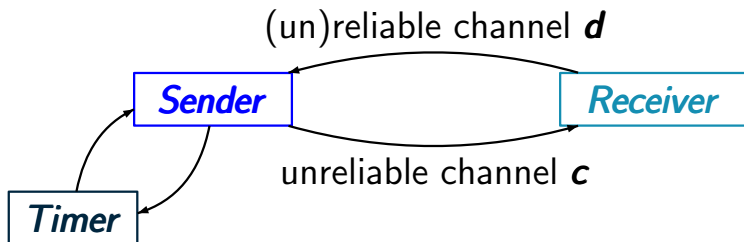
PC2.2-34



channel system: [**Sender** | **Timer** | **Receiver**]

Alternating bit protocol (ABP)

PC2.2-34



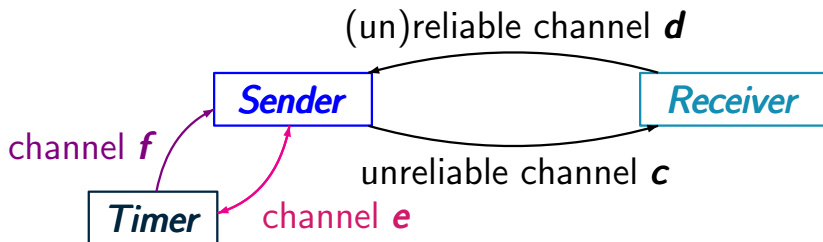
channel system: [**Sender** | **Timer** | **Receiver**]

synchronous message passing between
Timer and **Sender**

asynchronous message passing between
Receiver and **Sender**

Alternating bit protocol (ABP)

PC2.2-34



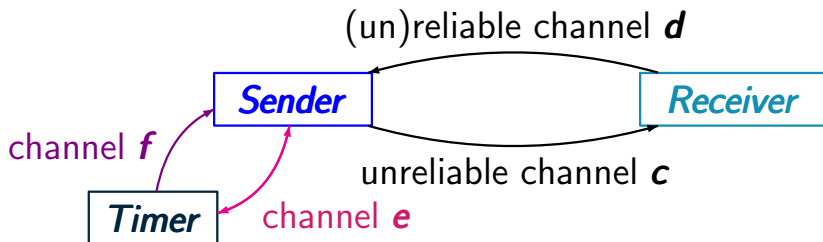
channel system: [**Sender** | **Timer** | **Receiver**]

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

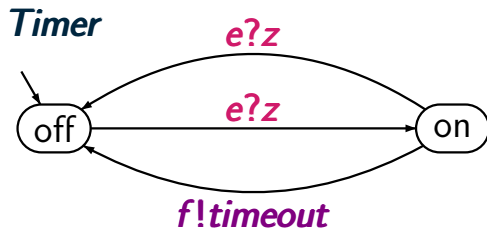
asynchronous message passing between
Receiver and **Sender** ← channels **c**, **d**

Alternating bit protocol (ABP)

PC2.2-34

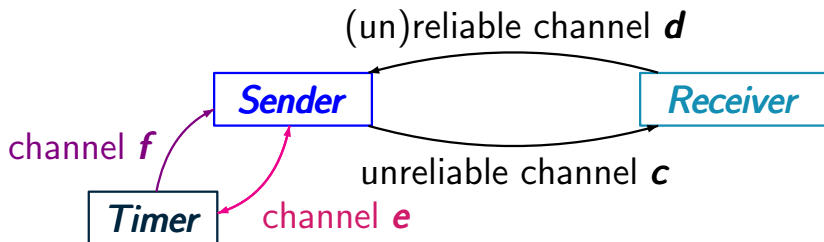


channel system: $[\text{Sender} \mid \text{Timer} \mid \text{Receiver}]$

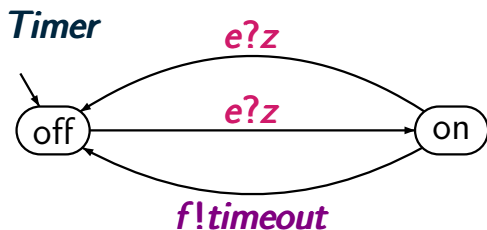


Alternating bit protocol (ABP)

PC2.2-34

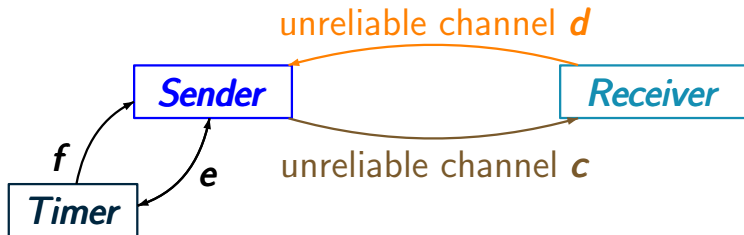


channel system: $[\text{Sender} \mid \text{Timer} \mid \text{Receiver}]$



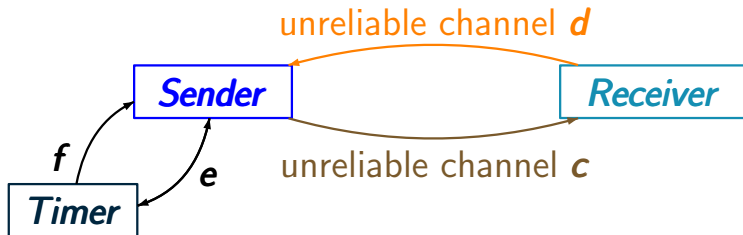
actions of **Sender**:

\vdots
 $e!timer_on$
 $e!timer_off$
 $f?z'$
 \vdots



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

- asynchronous channels ***c*** and ***d***
- synchronous channels ***e*** and ***f***

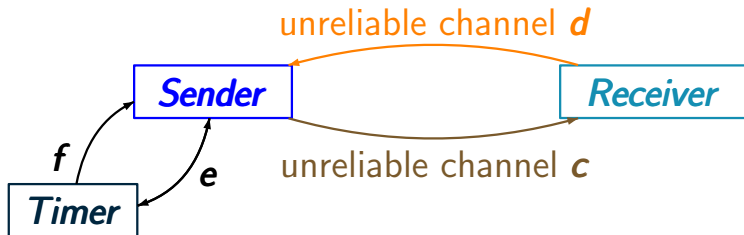


specify the **sender** by a program graph using

- asynchronous channels **c** and **d**
- synchronous channels **e** and **f**

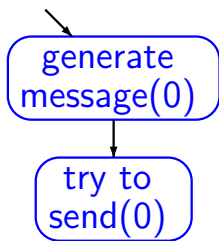
simply write

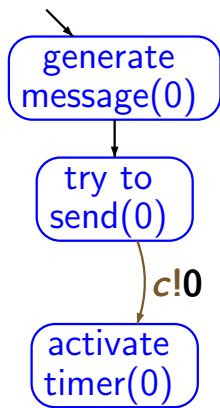
$\begin{array}{c} \nwarrow \nearrow \\ \text{!}timer_off \\ \text{?}timer_on \\ \text{?}timer_off \end{array}$

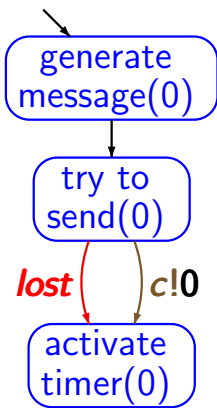


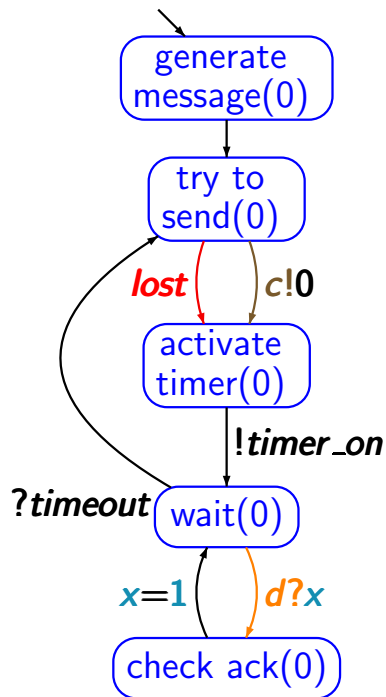
specify the **sender** by a program graph using

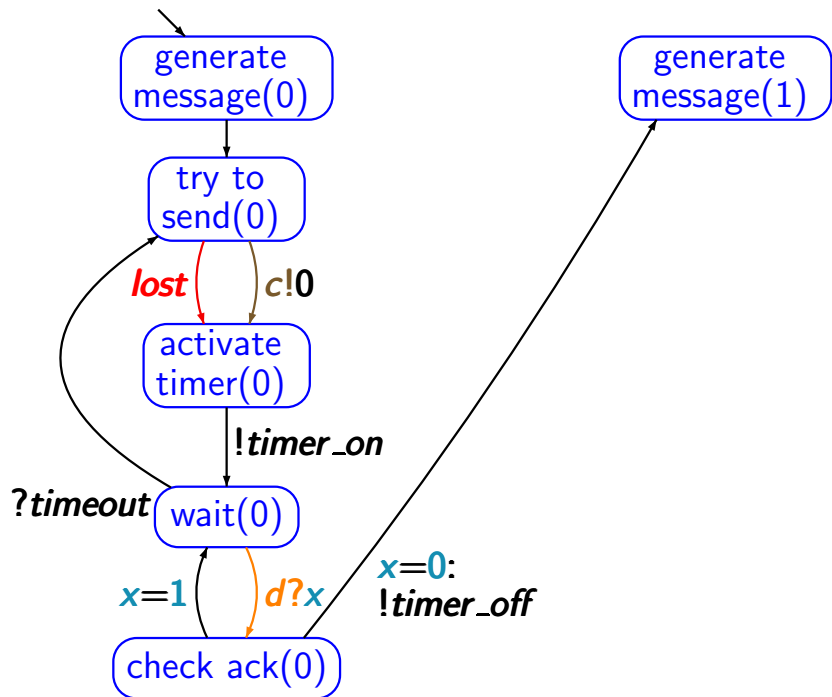
- asynchronous channels ***c*** and ***d***
- synchronous channels ***e*** and ***f***
- Boolean variable ***x*** for the acknowledgement bit sent by the receiver

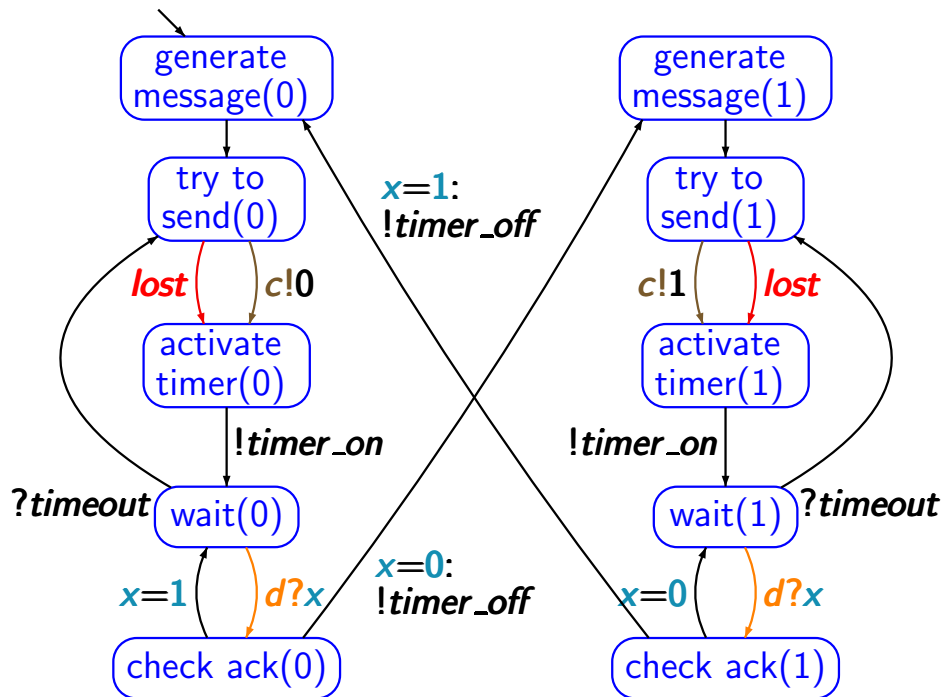






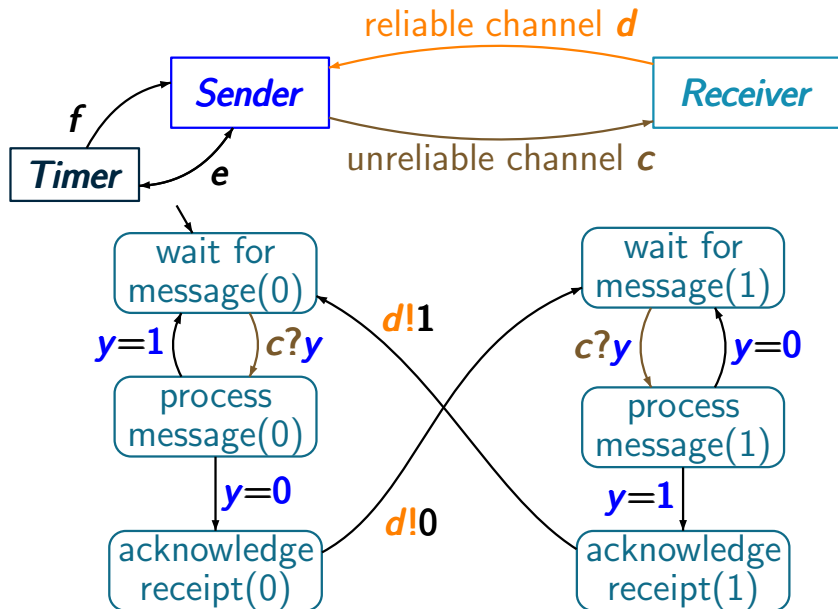


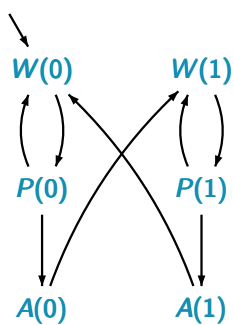
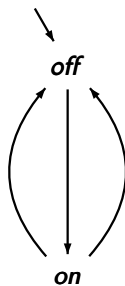
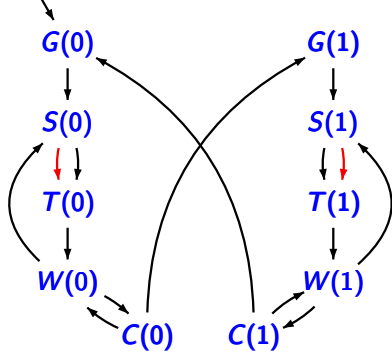


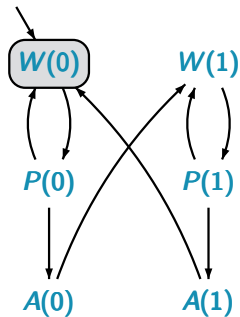
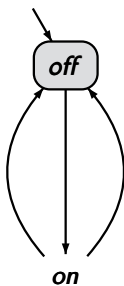
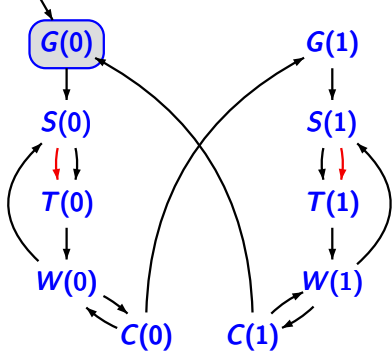


Program graph for the receiver

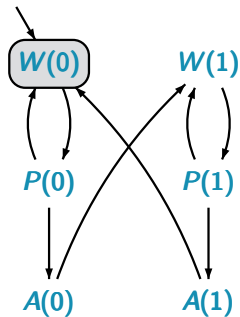
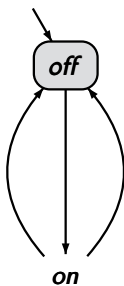
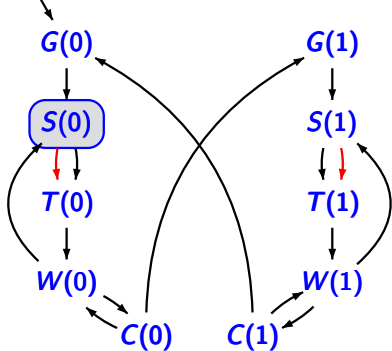
PC2.2-36







Generate(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$



Generate(0)

off

Wait(0)

$c = \varepsilon$

$d = \varepsilon$

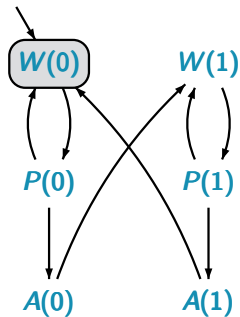
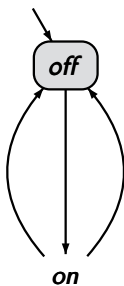
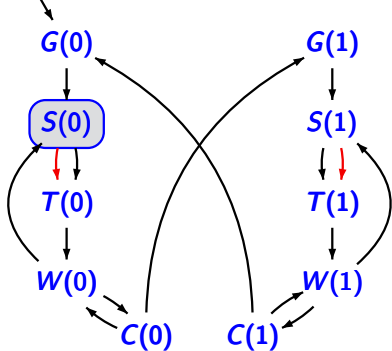
Send(0)

off

Wait(0)

$c = \varepsilon$

$d = \varepsilon$



Generate(0)

off

Wait(0)

$c = \varepsilon$

$d = \varepsilon$

Send(0)

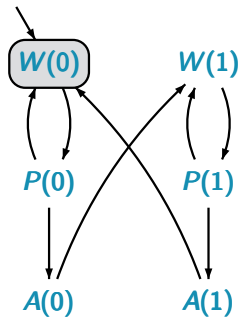
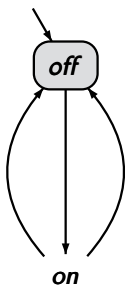
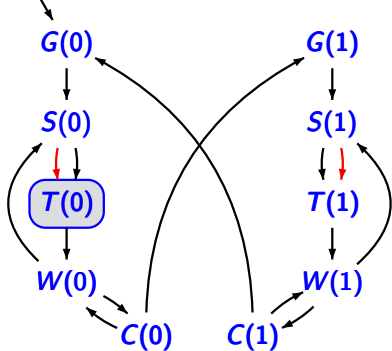
off

Wait(0)

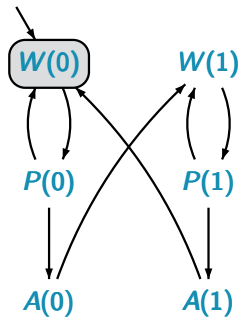
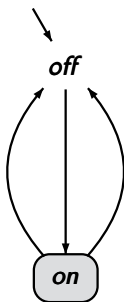
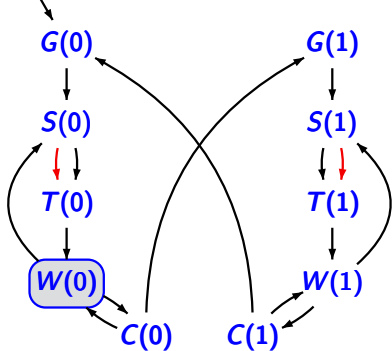
$c = \varepsilon$

$d = \varepsilon$

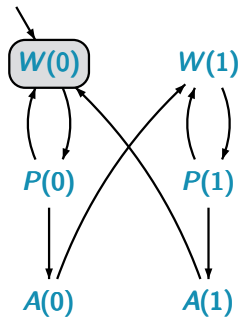
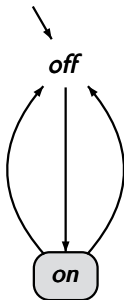
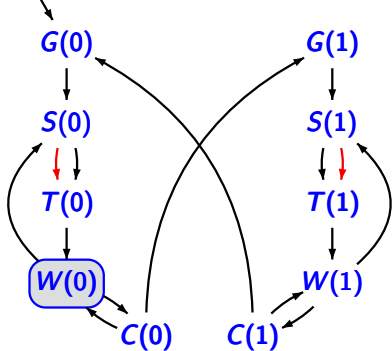
message **lost**



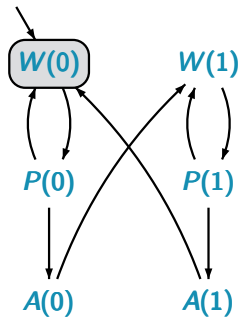
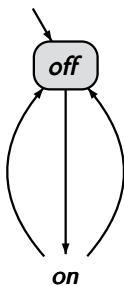
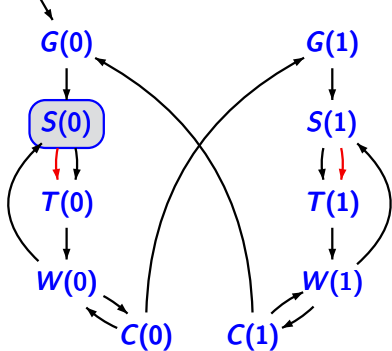
Generate(0)	off	Wait(0)	$c = \varepsilon$	$d = \varepsilon$	
Send(0)	off	Wait(0)	$c = \varepsilon$	$d = \varepsilon$	message lost
Timer_on(0)	off	Wait(0)	$c = \varepsilon$	$d = \varepsilon$	



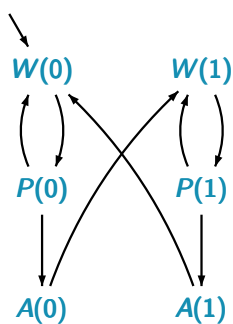
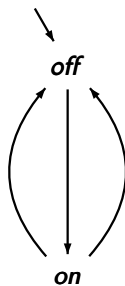
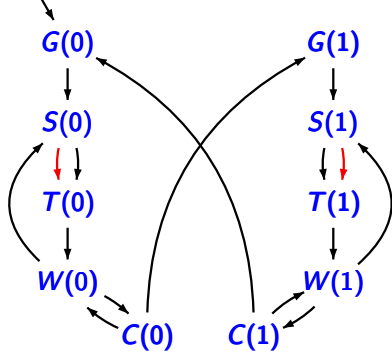
Generate(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	
Send(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	message lost
Timer_on(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	
Wait(0)	on	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	

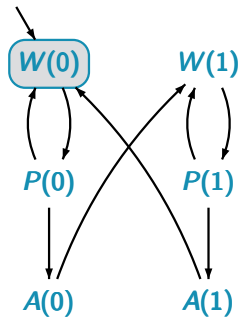
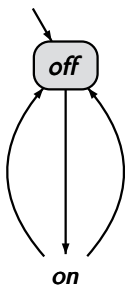
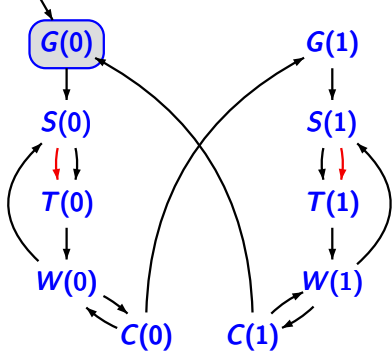


Generate(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	
Send(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	message lost
Timer_on(0)	off	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	
Wait(0)	on	Wait(0)	$c=\varepsilon$	$d=\varepsilon$	timeout

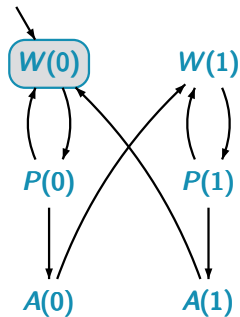
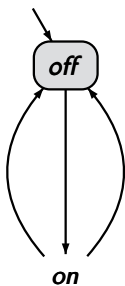
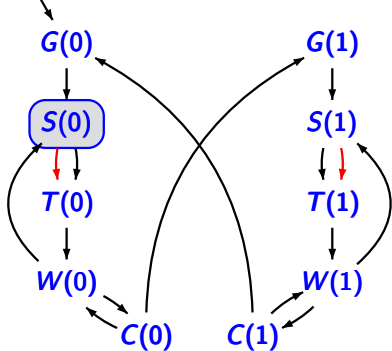


Generate(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	message lost
Timer_on(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Wait(0)	on	Wait(0)	$c=\epsilon$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
	\vdots				try again





Generate(0) off Wait(0) $c=\epsilon$ $d=\epsilon$



Generate(0)
Send(0)

off
off

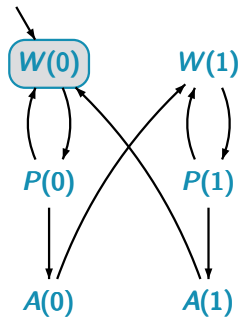
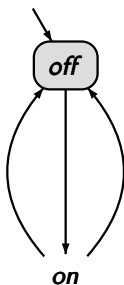
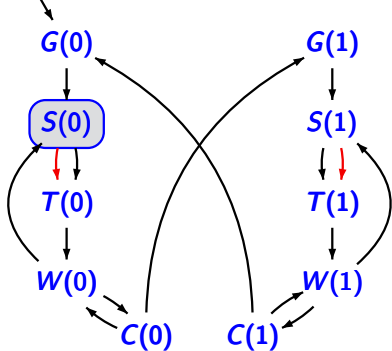
Wait(0)
Wait(0)

$c = \epsilon$

$d = \epsilon$

$c = \epsilon$

$d = \epsilon$



Generate(0)
Send(0)

off
off

Wait(0)
Wait(0)

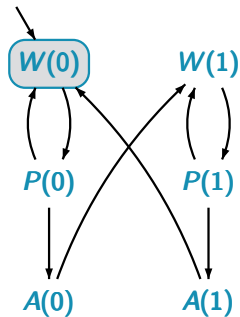
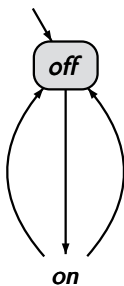
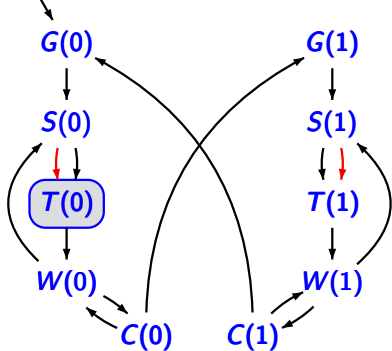
$c = \epsilon$

$c = \epsilon$

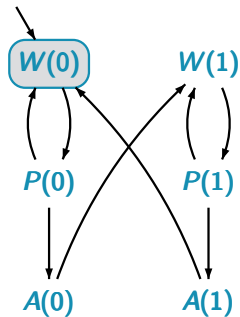
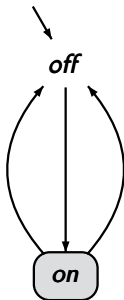
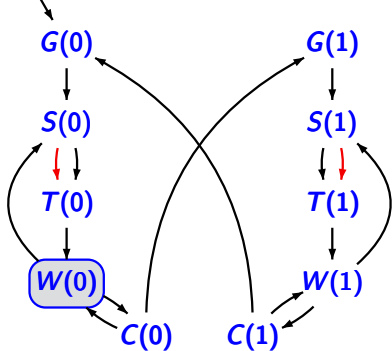
$d = \epsilon$

$d = \epsilon$

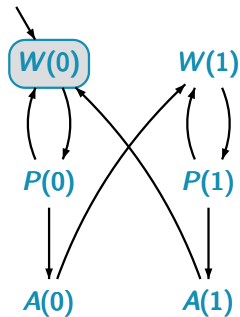
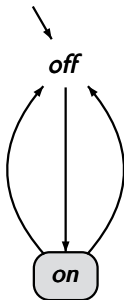
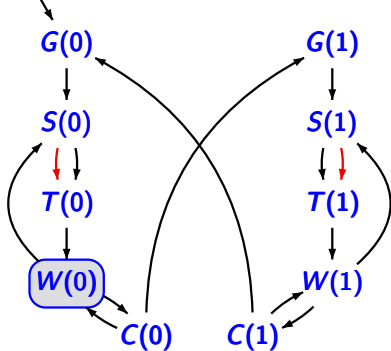
message 0 sent



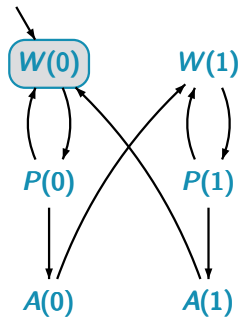
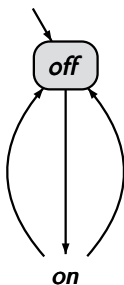
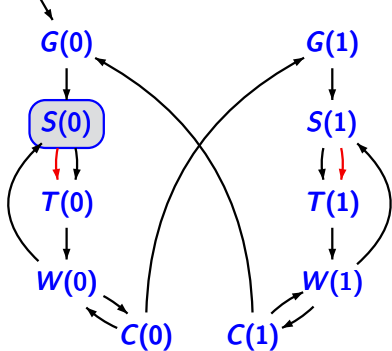
Generate(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	
Send(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c = 0$	$d = \epsilon$	



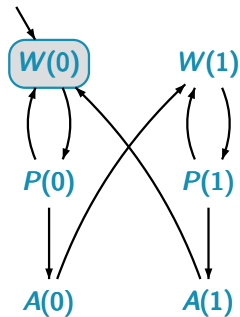
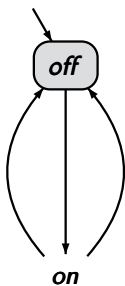
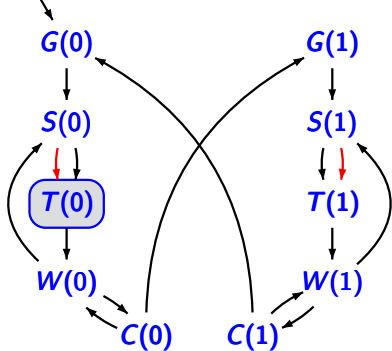
Generate(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	message 0 sent
Send(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	
Timer_on(0)	off	Wait(0)	$c = 0$	$d = \epsilon$	
Wait(0)	on	Wait(0)	$c = 0$	$d = \epsilon$	



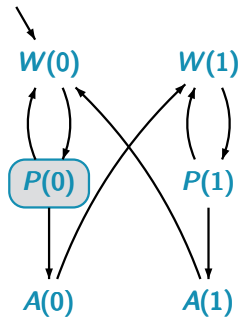
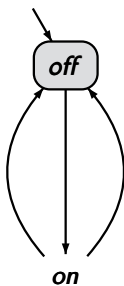
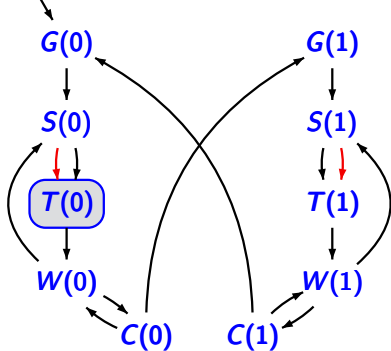
Generate(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	
Send(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c = 0$	$d = \epsilon$	
Wait(0)	on	Wait(0)	$c = 0$	$d = \epsilon$	timeout



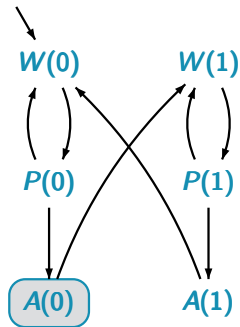
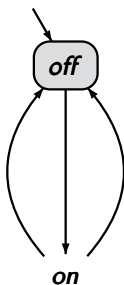
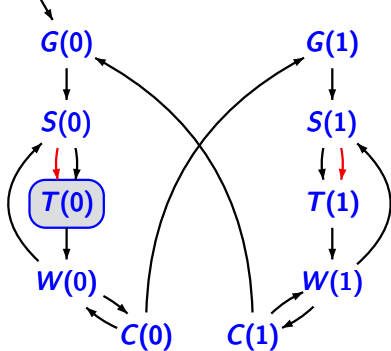
Generate(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	
Send(0)	off	Wait(0)	$c = \epsilon$	$d = \epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c = 0$	$d = \epsilon$	
Wait(0)	on	Wait(0)	$c = 0$	$d = \epsilon$	timeout
Send(0)	off	Wait(0)	$c = 0$	$d = \epsilon$	



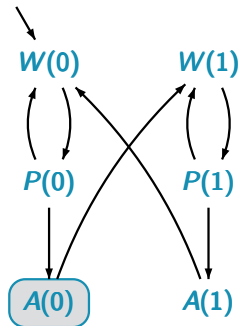
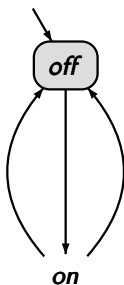
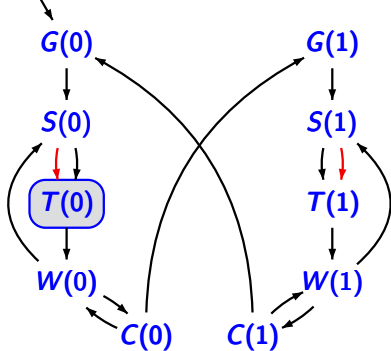
Generate(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c=0$	$d=\epsilon$	
Wait(0)	on	Wait(0)	$c=0$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=0$	$d=\epsilon$	0 sent again
Timer_on(0)	off	Wait(0)	$c=00$	$d=\epsilon$	



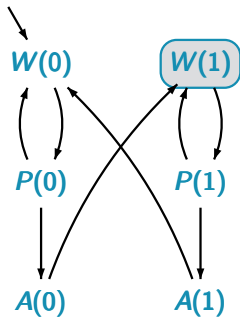
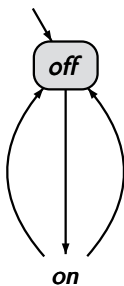
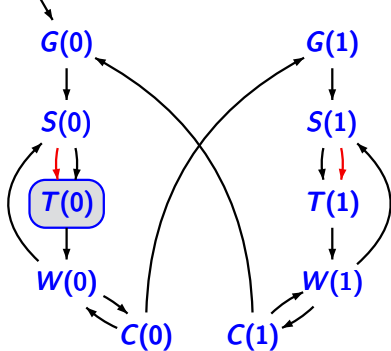
Generate(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c=0$	$d=\epsilon$	
Wait(0)	on	Wait(0)	$c=0$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=0$	$d=\epsilon$	0 sent again
Timer_on(0)	off	Wait(0)	$c=00$	$d=\epsilon$	
Timer_on(0)	off	Proc(0)	$c=0$	$d=\epsilon$	message received



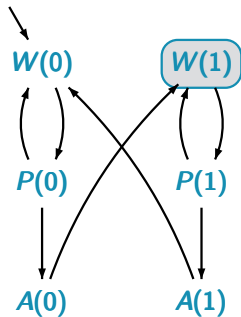
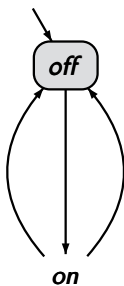
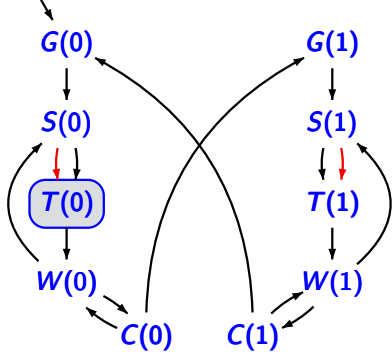
Generate(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c=0$	$d=\epsilon$	
Wait(0)	on	Wait(0)	$c=0$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=0$	$d=\epsilon$	0 sent again
Timer_on(0)	off	Wait(0)	$c=00$	$d=\epsilon$	
Timer_on(0)	off	Proc(0)	$c=0$	$d=\epsilon$	message received
Timer_on(0)	off	Ack(0)	$c=0$	$d=\epsilon$	



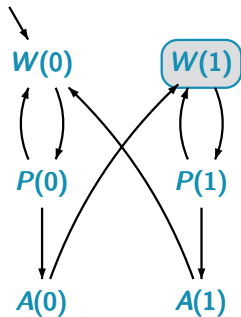
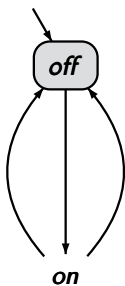
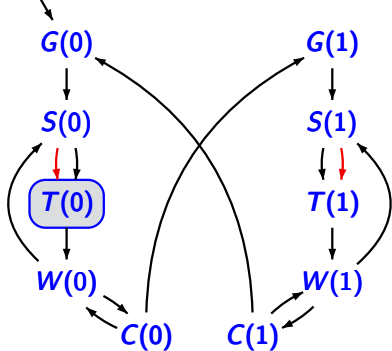
Generate(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	
Send(0)	off	Wait(0)	$c=\epsilon$	$d=\epsilon$	message 0 sent
Timer_on(0)	off	Wait(0)	$c=0$	$d=\epsilon$	
Wait(0)	on	Wait(0)	$c=0$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=0$	$d=\epsilon$	0 sent again
Timer_on(0)	off	Wait(0)	$c=00$	$d=\epsilon$	
Timer_on(0)	off	Proc(0)	$c=0$	$d=\epsilon$	message received
Timer_on(0)	off	Ack(0)	$c=0$	$d=\epsilon$	send ack via d



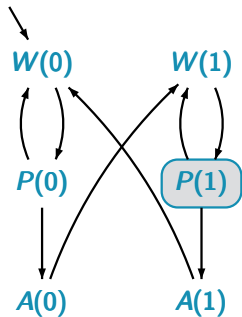
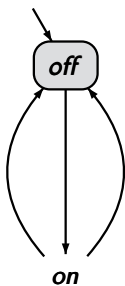
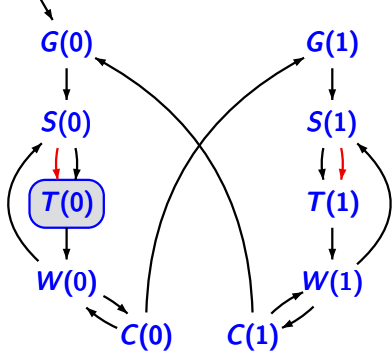
⋮	⋮	⋮	⋮	⋮	
Wait(0)	on	Wait(0)	$c=0$	$d=\epsilon$	timeout
Send(0)	off	Wait(0)	$c=0$	$d=\epsilon$	0 sent again
Timer_on(0)	off	Wait(0)	$c=00$	$d=\epsilon$	
Timer_on(0)	off	Proc(0)	$c=0$	$d=\epsilon$	message received
Timer_on(0)	off	Ack(0)	$c=0$	$d=\epsilon$	send ack via d
Timer_on(0)	off	Wait(1)	$c=0$	$d=0$	receiver changes its mode



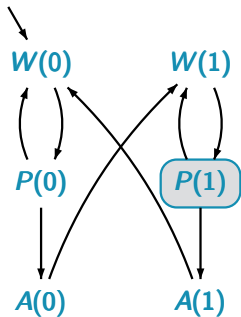
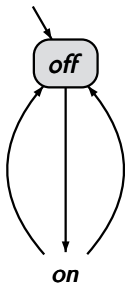
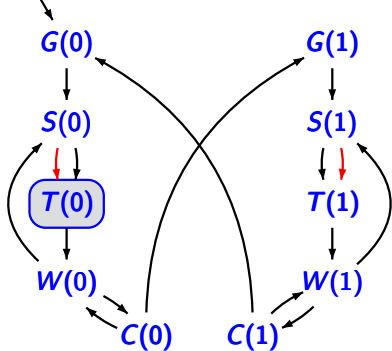
\vdots
 Timer_on(0) \vdots off \vdots Wait(1) \vdots $c=0$ \vdots $d=0$



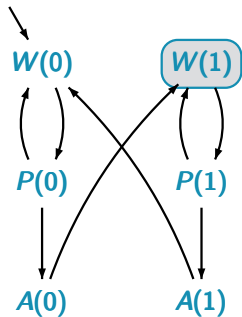
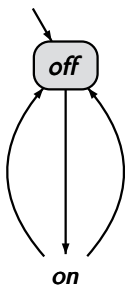
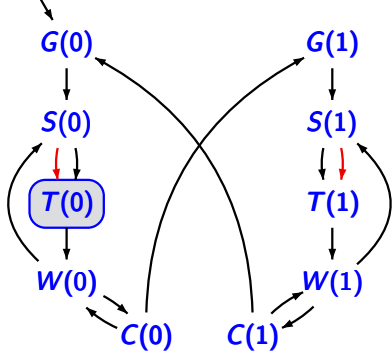
\vdots Timer_on(0) \vdots off \vdots Wait(1) \vdots $c=0$ \vdots $d=0$ receiver reads the same message again



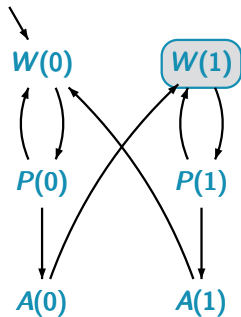
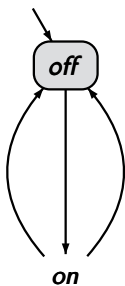
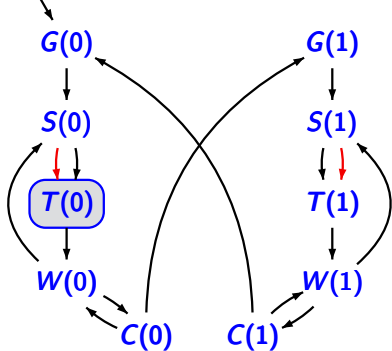
\vdots	\vdots	\vdots	\vdots	
Timer_on(0)	off	Wait(1)	$c=0$	$d=0$ receiver reads the
				same message again
Timer_on(0)	off	Proc(1)	$c=\varepsilon$	$d=0$



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



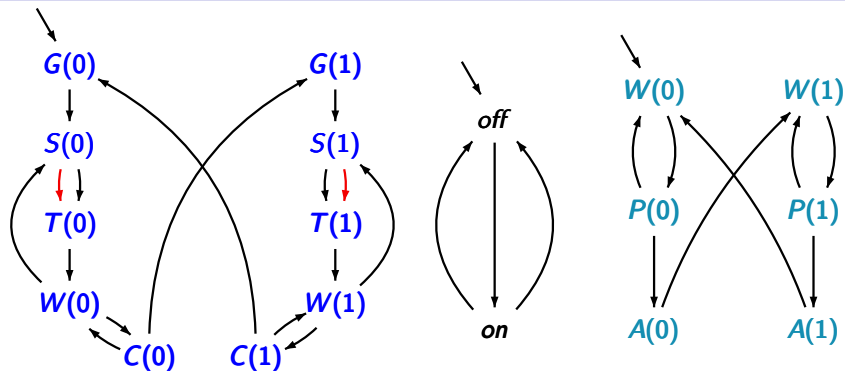
\vdots	\vdots	\vdots	\vdots	
Timer_on(0)	off	Wait(1)	$c=0$ $d=0$	receiver reads the same message again
Timer_on(0)	off	Proc(1)	$c=\varepsilon$ $d=0$	receiver discards the message
Timer_on(0)	off	Wait(1)	$c=\varepsilon$ $d=0$	



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

Alternating bit protocol (ABP)

PC2.2-37C

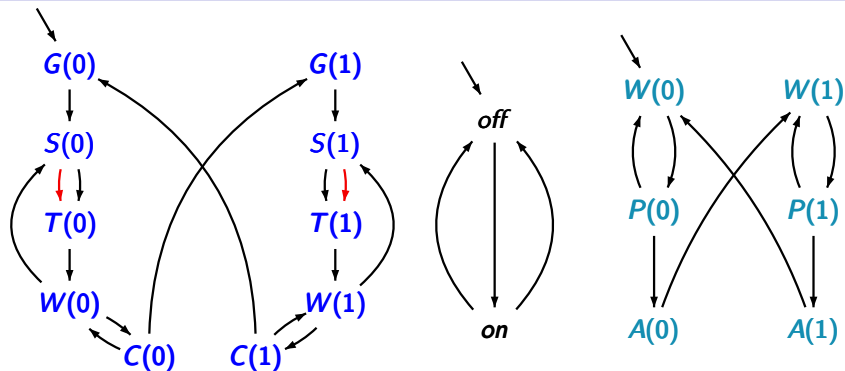


number of states in the TS:

$$10 \cdot 2 \cdot 6 \cdot \text{\#channel evaluations}$$

Alternating bit protocol (ABP)

PC2.2-37C



number of states in the TS:

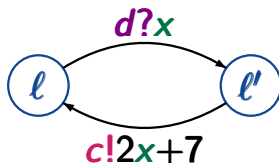
$10 \cdot 2 \cdot 6 \cdot \text{\#channel evaluations}$

$> 10^8$ for FIFOs with capacity 10

- conditional communication actions $\ell \xrightarrow{g:c?x} \ell'$

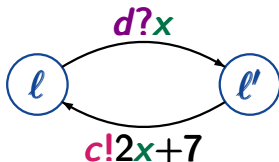
- conditional communication actions $\ell \xrightarrow{g:c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$

e.g.



- conditional communication actions $\ell \xleftrightarrow{g:c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$

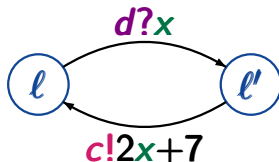
e.g.



- communication as conditions $\ell \xleftrightarrow{c?x:\alpha} \ell'$

- conditional communication actions $\ell \xleftrightarrow{g:c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$

e.g.

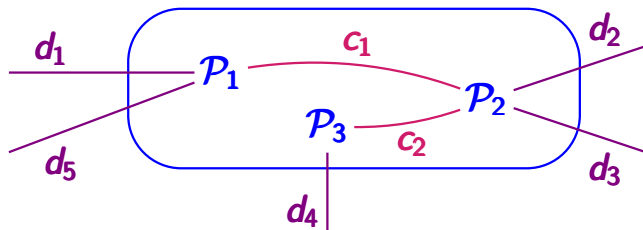


- communication as conditions $\ell \xleftrightarrow{c?x:\alpha} \ell'$

\rightsquigarrow more compact TS-representations

- conditional communication actions $\ell \xleftrightarrow{g:c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$
- communication as conditions $\ell \xleftrightarrow{c?x:\alpha} \ell'$
- *open* channel systems $\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n$
instead of *closed* channel systems $[\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n]$

- conditional communication actions $\ell \xrightarrow{g:c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$
- communication as conditions $\ell \xrightarrow{c?x:\alpha} \ell'$
- *open* channel systems $\mathcal{P}_1 | \dots | \mathcal{P}_n$
instead of *closed* channel systems $[\mathcal{P}_1 | \dots | \mathcal{P}_n]$



(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in *Syn*

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in *Syn*

interleaving for program graphs $\mathcal{P}_1 \parallel \mathcal{P}_2$

- interleaving for concurrent actions
- communication via shared variables

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in *Syn*

interleaving for program graphs $\mathcal{P}_1 \parallel \mathcal{P}_2$

- interleaving for concurrent actions
- communication via shared variables

channel systems: open $\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n$ or closed $[\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n]$

- interleaving, shared variables, message passing

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication

synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

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

interleaving for program graphs $\mathcal{P}_1 \parallel \mathcal{P}_2$

- interleaving, shared variables

channel systems: open $\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n$ or closed $[\mathcal{P}_1 \mid \dots \mid \mathcal{P}_n]$

- interleaving, shared variables
- synchronous and asynchronous message passing

synchronous product for TS $\mathcal{T}_1 \otimes \mathcal{T}_2$

- no interleaving, “pure” synchronization

for parallel systems with fully synchronized processes

$$\left. \begin{array}{l} \mathcal{T}_1 = (\mathcal{S}_1, Act_1, \longrightarrow_1, \dots) \\ \mathcal{T}_2 = (\mathcal{S}_2, Act_2, \longrightarrow_2, \dots) \end{array} \right\} \text{two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 = (\mathcal{S}_1 \times \mathcal{S}_2, Act, \longrightarrow, \dots)$$

for parallel systems with fully synchronized processes

$$\left. \begin{array}{l} \mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \dots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \dots) \end{array} \right\} \text{two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 = (S_1 \times S_2, Act, \longrightarrow, \dots)$$

where the action set Act is given by a function

$$Act_1 \times Act_2 \longrightarrow Act, \quad (\alpha, \beta) \mapsto \alpha * \beta$$



action name for the concurrent
execution of α and β

for parallel systems with fully synchronized processes

$$\left. \begin{array}{l} \mathcal{T}_1 = (\mathcal{S}_1, \text{Act}_1, \longrightarrow_1, \dots) \\ \mathcal{T}_2 = (\mathcal{S}_2, \text{Act}_2, \longrightarrow_2, \dots) \end{array} \right\} \text{ two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 = (\mathcal{S}_1 \times \mathcal{S}_2, \text{Act}, \longrightarrow, \dots)$$

where the action set Act is given by a function

$$\text{Act}_1 \times \text{Act}_2 \longrightarrow \text{Act}, \quad (\alpha, \beta) \mapsto \alpha * \beta$$



action name for the concurrent
execution of α and β

if action names are irrelevant: $\text{Act}_1 = \text{Act}_2 = \text{Act} = \{\tau\}$

for parallel systems with fully synchronized processes

$$\left. \begin{array}{l} \mathcal{T}_1 = (\mathbf{S}_1, \mathbf{Act}_1, \longrightarrow_1, \dots) \\ \mathcal{T}_2 = (\mathbf{S}_2, \mathbf{Act}_2, \longrightarrow_2, \dots) \end{array} \right\} \text{two TS}$$

synchronous product:

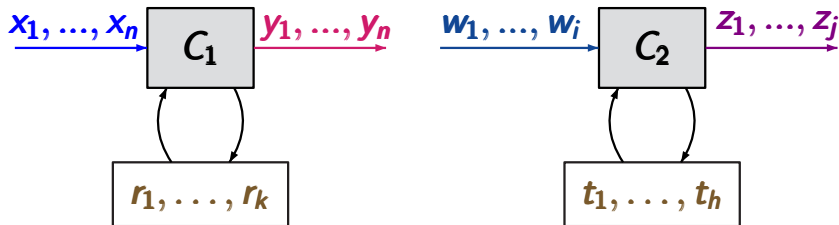
$$\mathcal{T}_1 \otimes \mathcal{T}_2 = (\mathbf{S}_1 \times \mathbf{S}_2, \mathbf{Act}, \longrightarrow, \dots)$$

transition relation \longrightarrow :

$$\frac{\mathbf{s}_1 \xrightarrow{\alpha}_1 \mathbf{s}'_1 \wedge \mathbf{s}_2 \xrightarrow{\beta}_2 \mathbf{s}'_2}{\langle \mathbf{s}_1, \mathbf{s}_2 \rangle \xrightarrow{\alpha * \beta} \langle \mathbf{s}'_1, \mathbf{s}'_2 \rangle}$$

Synchronous product for composing circuits

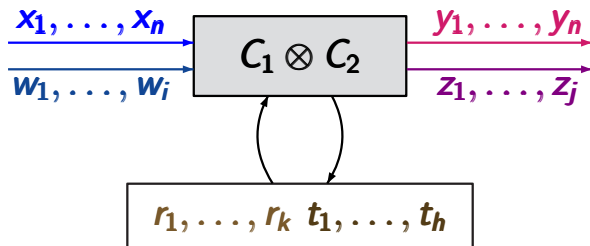
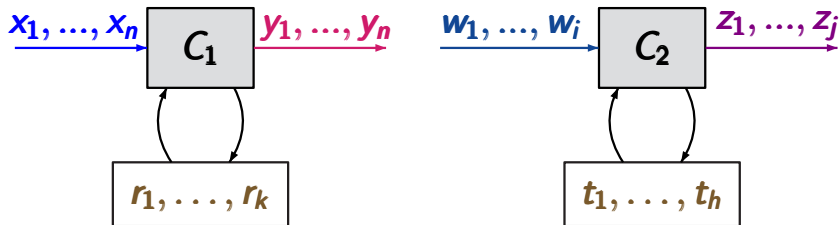
PC2.2-40



2 sequential circuits

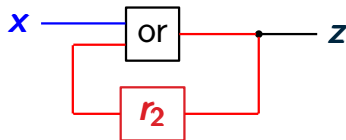
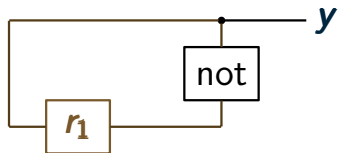
Synchronous product for composing circuits

PC2.2-40



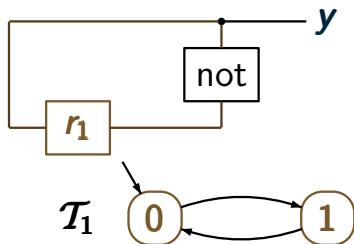
Synchronous product: example

PC2.2-52



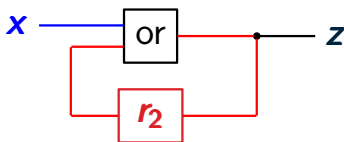
Synchronous product: example

PC2.2-52



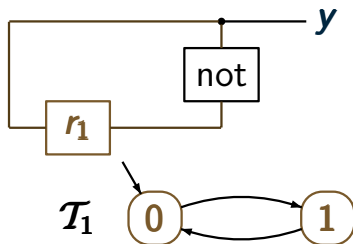
initially:
 $r_1 = 0$

transition function:
 $\delta_{r_1} = \neg r_1$



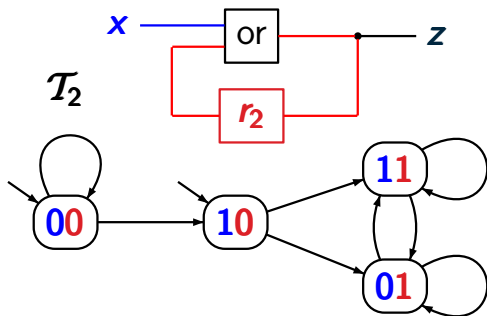
Synchronous product: example

PC2.2-52



initially:
 $r_1 = 0$

transition function:
 $\delta_{r_1} = \neg r_1$

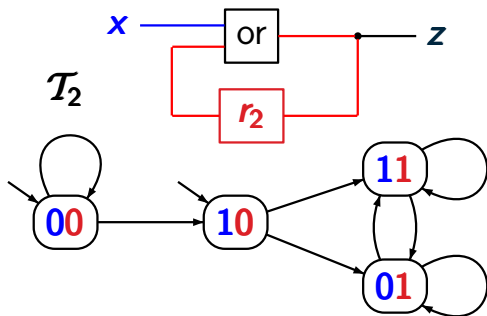
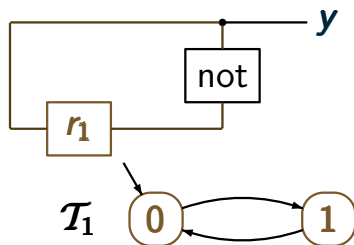


initially:
 $r_2 = 0$

transition function:
 $\delta_{r_2} = r_2 \vee x$

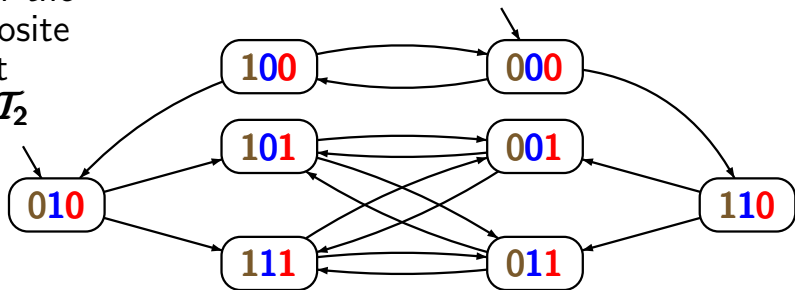
Synchronous product: example

PC2.2-52



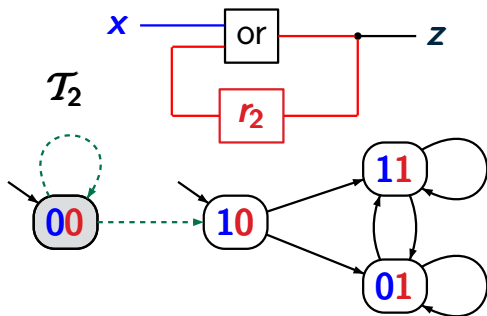
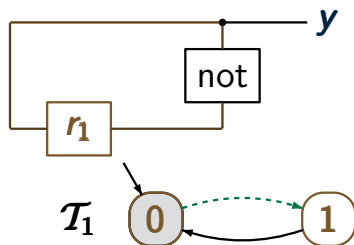
TS for the
composite
circuit

$\mathcal{T}_1 \otimes \mathcal{T}_2$



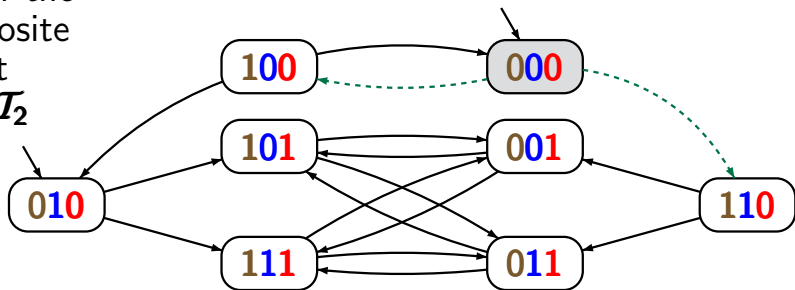
Synchronous product: example

PC2.2-52



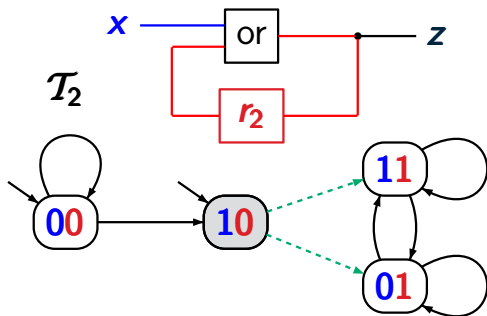
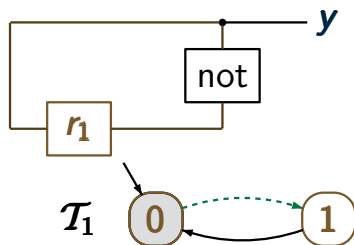
TS for the
composite
circuit

$\mathcal{T}_1 \otimes \mathcal{T}_2$



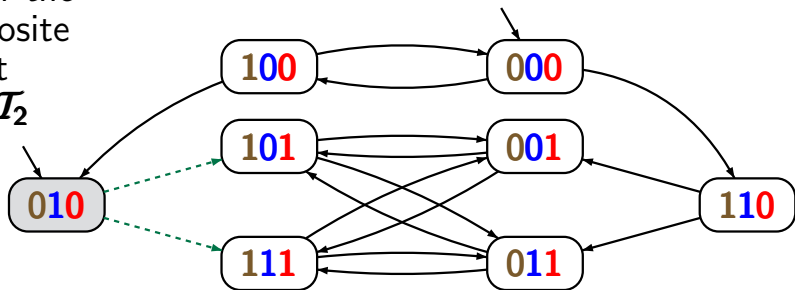
Synchronous product: example

PC2.2-52



TS for the
composite
circuit

$\mathcal{T}_1 \otimes \mathcal{T}_2$



TS for reactive systems can be enormously large

TS for reactive systems can be enormously large

- **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

- **infinite** for systems with
 - * variables of infinite domains, e.g., \mathbb{N}
 - * infinite data structures, e.g., stacks, queues, lists,...
- if finite: **exponential growth** in

TS for reactive systems can be enormously large

- **infinite** for systems with
 - * variables of infinite domains, e.g., \mathbb{N}
 - * infinite data structures, e.g., stacks, queues, lists,...
- if finite: **exponential growth** in
 - * number of parallel components,
e.g., state space of $T_1 \parallel \dots \parallel T_n$ is $S_1 \times \dots \times S_n$

TS for reactive systems can be enormously large

- **infinite** for systems with
 - * variables of infinite domains, e.g., \mathbb{N}
 - * infinite data structures, e.g., stacks, queues, lists,...
- if finite: **exponential growth** in
 - * number of parallel components,
e.g., state space of $T_1 \parallel \dots \parallel T_n$ is $S_1 \times \dots \times S_n$
 - * number of variables and channels

TS for reactive systems can be enormously large

- **infinite** for systems with
 - * variables of infinite domains, e.g., \mathbb{N}
 - * infinite data structures, e.g., stacks, queues, lists,...
- if finite: **exponential growth** in
 - * number of parallel components,
e.g., state space of $\mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n$ is $S_1 \times \dots \times S_n$
 - * number of variables and channels

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

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

