Session 7
Communication by channels

Contents

Parallelism and Communication

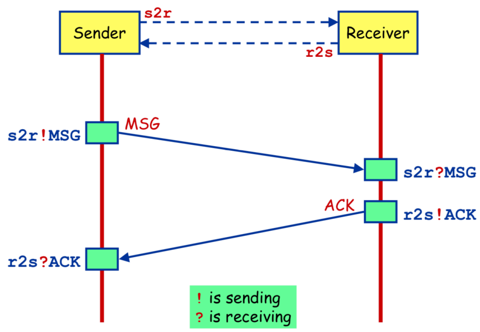
In general, hardware and software systems are parallel and may communicate among them. We want to define the system

$$T_1 || T_2 \cdots || T_n$$

Parallelism can be modelled in several ways:

- Interleaving processes (asynchronous).
- Communication by shared variables
- Synchronous product
- *Handshaking* (actions allow to synchronise processes)
- *Message passing - communication by channels*

Communication by channels



- a channel has two operations:*send* and *receive*.

1 Channels Systems

Communication by channels

- A channel can be a buffer FIFO (shared variable)
- Models communication in networks and communication protocols
- Also, basic for concurrency modelling formalisms
- *channel system*
- has n processes $P_1, \ldots, P_n$, each one with a program graph PG_i with
- conditional transitions $g : \alpha$ or *communication actions*:
- $g : c!v$ transmit the value v along channel c
- $g : c?x$ receive a message via channel c and assign it to variable x.
- $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$, $\ell \overset{g:c!v}{\hookrightarrow} \ell'$, or $\ell \overset{g:c?x}{\hookrightarrow} \ell'$.
- communication actions can be synchronous or asynchronous.

Channels

- Let c be a buffer.
- $c!v$ puts v in the end of the buffer c
- $c?x$ gets the element in the top of the buffer c and assigns it to x

- *channel capacity*,

$$cap(c) \in \mathbb{N} \cup \{\infty\},$$

indicates the maximum number of messages that c can store (can be finite or infinite)

- *channel type*, indicates the type of messages that can be transmitted over c, $dom(c)$.
- Let $Chan$ be the set of channels, the seset of *communication actions* is

$$\begin{aligned} Comm &= \{c!v, c?x \mid c \in Chan \wedge v \in dom(c) \wedge \\ &\quad x \in Var \wedge dom(x) \subseteq dom(c)\} \end{aligned}$$

1.1 Synchronous and Asynchronous

Synchronous and Asynchronous

- Ex: a channel c that transmits bits has $dom(c) = \{0, 1\}$
- If $cap(c) = 0$ the system corresponds to *Handshaking*: simultaneous transmission and receipt, *synchronous message passing*
- If $cap(0) > 0$ there is a delay between the transmission and the receipt of a message: *asynchronous message passing*

Channel System (CS)

$CS = [PG_1|PG_2|\cdots|PG_n]$ over $(Var, Chan)$ where PG_i are program graphs over $(Var_i, Chan)$

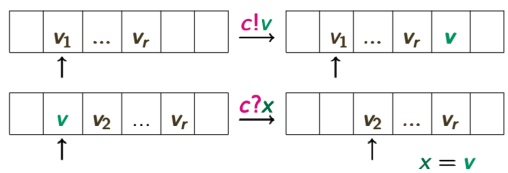
- $Var = \bigcup_{1\leq i\leq n} Var_i$ set of typed variables
- $Chan$ set of typed channels with capacities $cap(\cdot)$ and domains $dom(\cdot)$
- $PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$

$$\hookrightarrow_i \subseteq Loc_i \times (Cond(Var_i) \times (Act_i \cup Comm_i)) \times Loc_i$$

- $\ell \overset{g:\alpha}{\hookrightarrow}_i \ell'$, g guard
- $\ell \overset{g:c!v}{\hookrightarrow}_i \ell'$, sends the value v over the channel c
- $\ell \overset{g:c?x}{\hookrightarrow}_i \ell'$, receives a message along c and stores it in x
- If $g = True$ we can omit $g:$ in the communication actions.

Communication if $cap(c) > 0$

- P_i can perform the conditional transition $\ell_i \overset{c!v}{\hookrightarrow}_i \ell_i'$ iff *channel c is not full* and v is stored in the end of the channel c ($add(c, v)$)
- P_j can execute $\ell_j \overset{c?x}{\hookrightarrow}_j \ell_j'$ if *channel c is not empty*



Communication if $cap(c) = 0$ (*rendezvous*)

- process P_i can transmit a value v over a channel c,

$$\ell_i \overset{c!v}{\hookrightarrow}_i \ell_i'$$

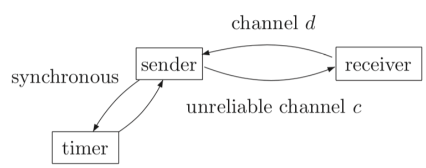
  if there is another process P_j that offers a complementary receive action

$$\ell_j \overset{c?x}{\hookrightarrow}_j \ell_j'$$

- being the effect of message passing equivalent to $x := v$.

2 Example: Alternating Bit Protocol

***Alternating Bit Protocol* (ABP)**



- channel c is not perfect and can lose sent messages (e.g. large data packets)
- channel d is perfect and sends "acknowledgment" (e.g. small data packets)
- We want a communication protocol that
- ensures that all distinct transmitted data by S are delivered to R.
- For that S may have to retransmit messages (if timer timeouts)
- and a new message only is sent when it is warranted that the previous one was received (this is called *send and wait*)

Alternating Bit Protocol

- S sends a message, one extra bit y and activates the *timer*.
- if a *timeout* occurs the same message is sent again
- if R sent y then S restarts the timer and sets $y = \neg y$ (sending a new message)
- Without real-time, the timeout is implemented with nondeterminism

Sender

```
y ← 0
while True do
    (1) send message + bit y (or lose it) and activate timer
    (2) await timeout or ack x
    do
    if timeout then
        goto (1)
    else if x==y then
        turn off timer; y ← ¬y
    else
        ignore x
    od
```

Receiver

```
x ← 0
while True do
     await receive message + bit y
    if x == y then
        send ack x; x ← ¬x
    else
        ignore y
```

Alternating Bit Protocol

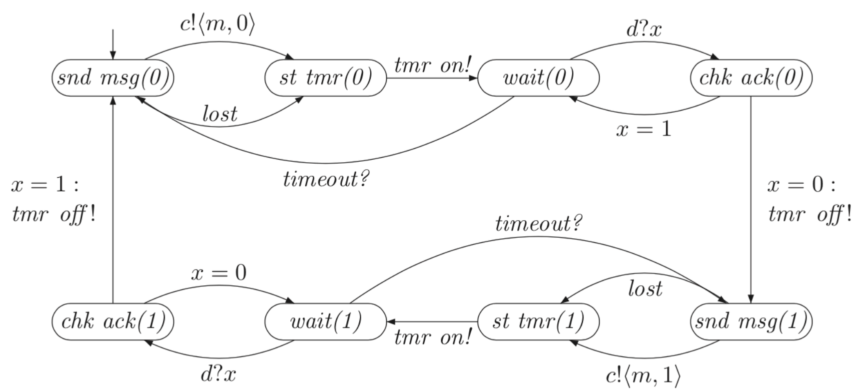
- S sends a message along c

$$\langle m_0, b_0 \rangle, \langle m_1, b_1 \rangle, \ldots$$

  e $b_0 = 0$, $b_1 = 1$, $b_2 = 0$, ...
- when R receives $\langle m, b \rangle$ sends the control bit b that receives from the channel d
- when S receives b, S transmits a new message m' with the bit $\neg b$.

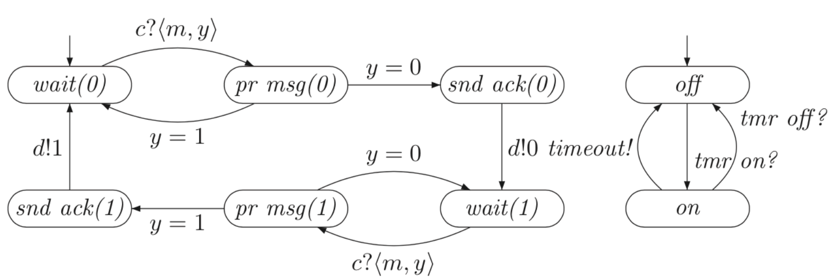
- If, however, S has to wait too long for a message from R, S timeouts and retransmits $\langle m, b\rangle$ (here the simulation is done using nondeterminism)
- b is the *alternating bit*

PG **for *Sender***



$$Chan = \{c, d, tmr\_on, tmr\_off, timeout\}$$
$$Var = \{x, y, m_i\}$$

PG **for *Receiver* and *Timer***



$$ABP = [S|Timer|R]$$

- rendezvous (synchronous message passing) between S and $Timer$
- asynchronous message passing between S and R

3 Transition Systems for Channel Systems

Transition System for CS

Let $CS = [PG_1|PG_2|\cdots|PG_n]$ over $(Var, Chan)$.

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$$

One can define the associated transition system $T(CS)$ where

- states are $\langle \ell_1, \ldots, \ell_n, \eta, \zeta \rangle$
- ℓ_i location in PG_i
- $\eta \in Eval(Var)$ current values of the variables
- $\zeta : Chan \to \bigcup_{c \in Chan} dom(c)^\star$ current content of the various channels
- for $c \in Chan$, $\zeta(c) \in dom(c)^\star$
- and $len(\zeta(c)) \leq cap(c)$
- $Eval(Chan)$ is the set of all ζ.

Transition System for CS

Let $CS = [PG_1|PG_2|\cdots|PG_n]$ over $(Var, Chan)$.

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$$

- initial states: components $\ell_i \in Loc_{0,i}$
- initially all channels are empty ($\zeta_0(c) = \varepsilon,\ c \in Chan$) and $len(\varepsilon) = 0$.
- $\zeta(c) = v_1 v_2 \cdots v_k$, with v_1 the channel *top*
- $len(\zeta) = k$
- $\zeta[c := w_1, w_2, \ldots, w_k]$ is the environment equal to ζ but with $\zeta(c) = w_1 w_2 \cdots w_k$

$$\zeta[c := w_1, w_2, \ldots, w_k](c') = \begin{cases} \zeta(c') \text{ se } c' \neq c \\ w_1 w_2 \cdots w_k \text{if } c' = c \end{cases}$$

$T(CS)$

$$T(CS) = (S, Act, \longrightarrow, I, AP, L)$$

- $S = (Loc_1 \times \cdots \times Loc_n) \times Eval(Var) \times Eval(Chan)$

- $Act = \bigoplus_{0<1\leq n} Act_i \oplus \{\tau\}$, disjoin union
- $I = \{ \langle \ell_1, \ldots, \ell_n, \eta, \zeta_0 \rangle \mid \forall 0 < i \leq n(\ell_i \in Loc_{0,i} \land \eta \models g_{0,i}) \}$
- $AP = \bigoplus_{0<1\leq n} Loc_i \oplus Cond(Var)$, onde could added conditions over channels: **emptyP**(c), **fullP**(c), etc
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \zeta \rangle) = \{\ell_1, \ldots, \ell_n\} \cup \{ g \in Cond(Var) \mid \eta \models g \}$
- transition relation $\longrightarrow$ with the rules for actions $\alpha \in Act_i$ and messages passing.

Interleaving for $\alpha \in Act_i$

$$\frac{\ell_i \overset{g:\alpha}{\hookrightarrow}_i \ell_i' \land \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta \rangle \overset{\alpha}{\longrightarrow} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \zeta \rangle}$$

with $\eta' = Effect(\alpha, \eta)$.

Message Passing for $c \in Chan$ **and** $cap(c) > 0$

- receive a value along c and store in x

$$\frac{\ell_i \overset{g:c?x}{\hookrightarrow}_i \ell_i' \land \eta \models g \land \zeta(c) = v_1 \cdots v_k \land k > 0}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta \rangle \overset{\tau}{\longrightarrow} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \zeta' \rangle}$$

  with $\eta' = \eta[x := v_1]$ e $\zeta' = \zeta[c := v_2 \cdots v_k]$.

- transmit a message $v \in dom(c)$ over c

$$\frac{\ell_i \overset{g:c!v}{\hookrightarrow}_i \ell_i' \land \eta \models g \land \zeta(c) = v_1 \cdots v_k \land k < cap(c)}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta \rangle \overset{\tau}{\longrightarrow} \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta', \zeta' \rangle}$$

  with $\zeta' = \zeta[c := v_1 \cdots v_k v]$.

Message passing synchronous for $c \in Chan$ **and** $cap(c) = 0$

$$\frac{\ell_i \overset{g_1:c?x}{\hookrightarrow}_i \ell_i' \land \eta \models g_1 \land \eta \models g_2 \land \ell_j \overset{g_2:c!v}{\hookrightarrow}_j \ell_j' \land i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots \ell_n, \eta, \zeta \rangle \overset{\tau}{\longrightarrow} \langle \ell_1', \ldots, \ell_i', \ldots, \ell_j', \ldots \ell_n', \eta', \zeta \rangle}$$

with $\eta' = \eta[x := v]$.

4 State-Space Explosion Problem

How many states has a transistion system

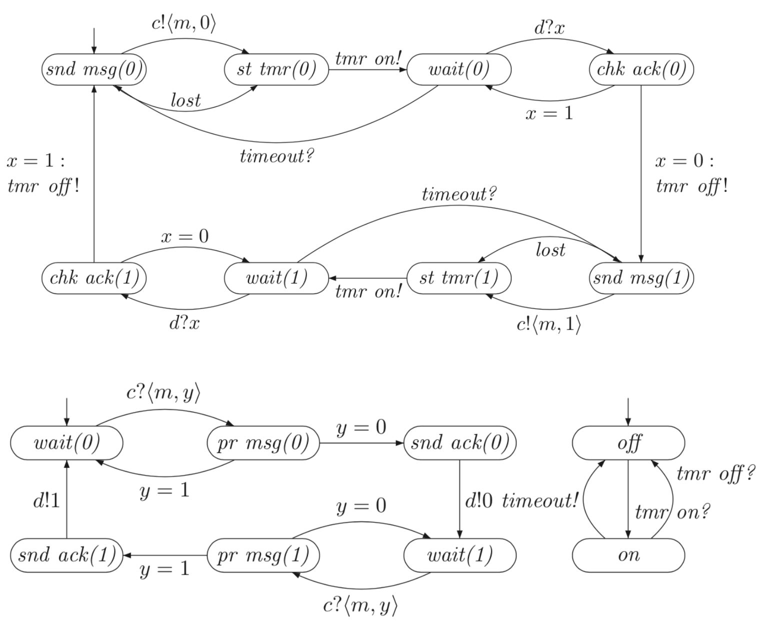
... of a channel system with:

- 2 processes with 2 locations
- 2 Boolean variables
- 2 channels of capacity 10 of type Boolean
- ?

$$2 \times 2 \times 2 \times 2 \times (1 + 2 + 2^2 + \cdots + 2^{10}) = 2^4(2^{11} - 1)^2 > 2^{24}$$

if the channels are unbound, $cap(c) = \infty$, the number of states is ∞.

ABP



$T(ABP)$

- Timer can *timeout* on each transmission of data by S thus the number of messages over c can be infinite,

- thus $T(ABP)$ can be infinite
- fragment of execution where a message is lost

sender S	timer	receiver R	channel c	channel d	event
$snd\ msg(0)$	*off*	$wait(0)$	$\varnothing$	$\varnothing$	
$st\ tmr(0)$	*off*	$wait(0)$	$\varnothing$	$\varnothing$	loss of message
$wait(0)$	*on*	$wait(0)$	$\varnothing$	$\varnothing$	
$snd\ msg(0)$	*off*	$wait(0)$	$\varnothing$	$\varnothing$	timeout
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Ignoring retransmissions

sender S	timer	receiver R	channel c	channel d	event
$snd\ msg(0)$	*off*	$wait(0)$	$\varnothing$	$\varnothing$	
$st\ tmr(0)$	*off*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	message with bit 0 transmitted
$wait(0)$	*on*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	
$snd\ msg(0)$	*off*	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	timeout
$st\ tmr(0)$	*off*	$wait(0)$	$\langle m, 0\rangle\ \langle m, 0\rangle$	$\varnothing$	retransmission
$st\ tmr(0)$	*off*	$pr\ msg(0)$	$\langle m, 0\rangle$	$\varnothing$	receiver reads first message
$st\ tmr(0)$	*off*	$snd\ ack(0)$	$\langle m, 0\rangle$	$\varnothing$	
$st\ tmr(0)$	*off*	$wait(1)$	$\langle m, 0\rangle$	0	receiver changes into mode-1
$st\ tmr(0)$	*off*	$pr\ msg(1)$	$\varnothing$	0	receiver reads retransmission
$st\ tmr(0)$	*off*	$wait(1)$	$\varnothing$	0	and ignores it
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

State-Space Explosion Problem

A transition system can be very large

- *infinite* if the variables has infinite domains (e.g.$\mathbb{N}$) or infinite data structures as stacks)
- *finite* with an exponential growth of the state space in terms of the number of components or the number of variables and channels
- $|Loc_1| \cdots |Loc_2| \prod_{x \in Var} |dom(x)|. \prod_{c \in Chan} |dom(c)|^{cap(c)}$
- L locations per component K channels of bits with capacity k and M variables with $|dom(x)| \leq m$ the number of states is
- $$L^n \cdot m^M \cdot 2^{K \cdot k}$$
- Example: ABP if $cap(c) = cap(d) = 10$, $dom(c) = dom(m) = \{0, 1\}$ e $|Loc_T| = 2$, $|Loc_R| = 6$, $|Loc_S| = 8$ the number of states is
$$2 \times 6 \times 8 \times 4^{10} \times (2^{11} - 1) > 3.2^{25}.$$

5 Channels in Promela

Channels in Promela

```
chan ch = [capacity] of { typename, ..., typename }
```

- allows the definition of channels where each message has several fields each one of a certain type.
- `capacity = 0` for synchronous channels
- *ch*!1 sends 1 (blocks if *ch* is full)
- *ch*?*x* receives a value and stores in *x* (blocks if *ch* is empty)
- normally declared globally
- if local they disappear when the process terminates
- can be passed as parameters of processes
- for receiving the variable *x* can be anonymous *ch*?_
- *arrays* of channels: **chan** [2] = [3] of {**byte**, **bool**}
- **full**, **nfull**, **empty**, **nempty** are Boolean functions to test the state of the channels
- **len** number of messages in a channel

Rendezvous

- Client-Server: cs1.pml , cs2.pml, cs3.pml, cs4.pml

```
chan request = [0] of {byte }

active proctype Server() {
byte client;
end:
do
:: request ? client ->
   printf(client)
od
}
active [2] proctype Client() {
request ! _pid
}
```

Buffers

- check if the channels are full or empty: cs5.pml

```
chan request = [0] of { byte, chan  };
chan reply [2]  = [2] of { byte };

active [2] proctype  Server() {
byte client;
chan replyChannel;
do
:: empty(request) ->  printf("No requests for %d\n",_pid)
:: request ? client, replyChannel ->
 printf("Client %d to  server %d\n",client, _pid);
   replyChannel ! _pid
od
}
```

Buffers

```
active [2] proctype  Client() {
       byte server;
       do
  :: full(request) ->
  printf("Client %d waiting for channel \n", _pid);
  ::  request ! _pid, reply[_pid-2];
       reply[_pid-2] ? server;
       printf("Response received from the server %d for the
     client %d\n",server, _pid);
       od
}
```

Conditional

`chan ch1 = [16] of { byte, int, chan, byte }`

- `ch1!exp1,exp2,exp3`
- `ch1?var1,var2,var3`
- `ch1!exp1(exp2,exp3)`
- `ch1?var1(var2,var3)`
- `ch1?[var1,var2,var3]` : eval to 1 if matches the values of the channel and 0 otherwise; no side effect (so no race conditions in case $var1, var2, var3$ shared by other processes).

Alternating bit protocol - `abp1.pml`

```
mtype ={msg,ack};
chan to_sender = [2] of { mtype, bit };
chan to_receiver = [2] of {mtype, bit};

active proctype Sender(){
       bool y, x;
       do
       :: true ->
send: to_receiver!msg(y);
      to_sender?ack(x);
       if
       :: y==x -> y= 1-y;
       :: timeout
       fi
       od
}
```

`timeout` boolean predefined global variable that is true if no statement is executable in aany activ process

Alternating bit protocol

```
active proctype Receiver(){
       bool x;
       do
       :: true ->
rec:   to_receiver?msg(x);
        to_sender!ack(x);
       :: timeout -> to_sender!ack(x);
       od
}

#define sent  Sender@send
#define recv  Receiver@rec

ltl A1 { []<> sent }
ltl A2 { []<> recv }
ltl A3 {[](recv -> ( recv U (!recv &&(( ! recv) U sent))))}
```