Lecture 16

Note Title 3/6/2008

Synchronous communication: sender and receiver must both execute comm. operation at the same time.

Synch comm. is strictly more powerful than asynch.

History

Anthony Hoare — Communicating Sequential Processes - mid '70s
- Ada
- Occam

Robin Milner — Calculus of Communicating Systems CCS
Pi-calculus - mobile processes

Concurrent ML — sync. comm + functional programming

Basic idea:

Channel abstraction

send (c, v) } typed

recv (c)

so α channel (channel<α>)

send : (α channel * α) → unit

recv : (α channel) → α

When a process executes send(c, v) it waits until another process executes recv(c) Then both functions return. and The value of recv(c) is v.

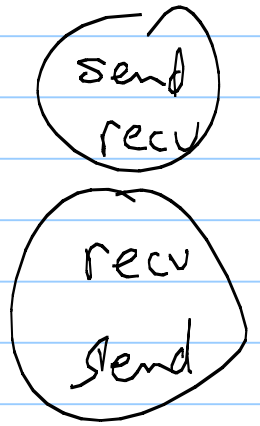
Problm: 1) a program has to predict in what order messages appear on different channels.
2) likely deadlocks

t_1	t_2
send(c1, v1)	send(c2, v2)
v2 = recv(c2)	v1 = recv(c1)

Choice operation

t_1

```
case
    send (c1, v1)
 [] v2 ∈ recv (c2)
end
```



t_2

```
case
    send (c2, v2)
 [] v1 = recv (c1)
end
```

Rendezvous

sendEvt and recvEvt "send event"

- value representing potential to communicate on a channel.

αchan α

se1 = sendEvt(c1, v1)
re1 = recvEvt(c2)

βchan β

se2 = sendEvt(c2, v2)
re2 = recvEvt(c1)

se1 : α sendEvt
re1 : β recvEvt

select [se1, re1]

select [se2, re2]

send(c,v) ≡ sync(sendEvt(c, v)) ≡ select [sendEvt(c,v)]

timeoutEvt(ms)

alwaysEvt()

neverEvt()

event function

wrap (e, f) —

event that when performed does the communication called for by e then calls f on the result.

```
select [
    wrap (recvEvt (c), fn v => .... v ...),
    :
    sendEvt (c2, v2)
]
```

t_1 t_2 t_3

select [c_1 , c_2] select [c_2 , c_3] select [c_3 , c_1]

t_1 t_2 t_3 t_4

select [c_1 , c_2] select [c_1 , c_2] select [c_3 , c_4] select [c_3 , c_4]

Each sent value is received by <u>one receiver</u>.