Balzer 4

abstract syntax linear session types

$$A, B, C ::= A \otimes B \mid A \multimap B \mid \& \{\overline{l : A}\} \mid \oplus \{\overline{l : A}\} \mid\ !A$$

structural context (also has weakening, contraction)

$$\Psi ; \Delta \vdash P :: (x : A)$$

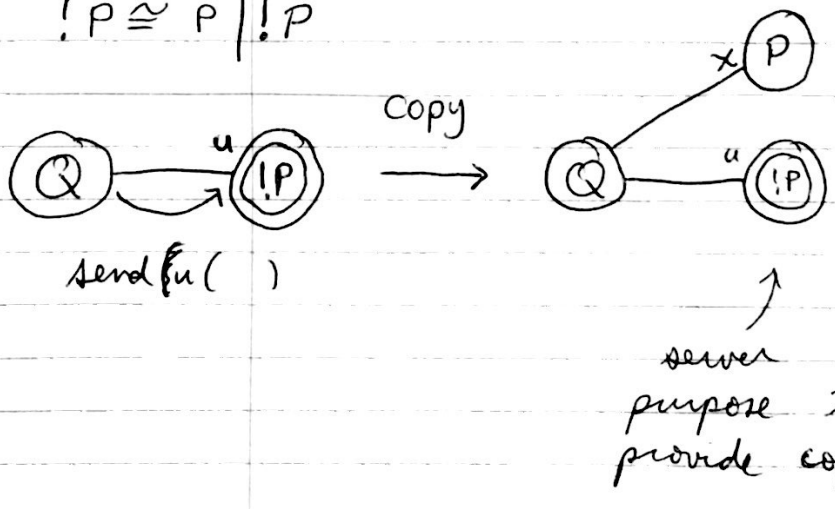
linear context

cut! copy !R !L rules

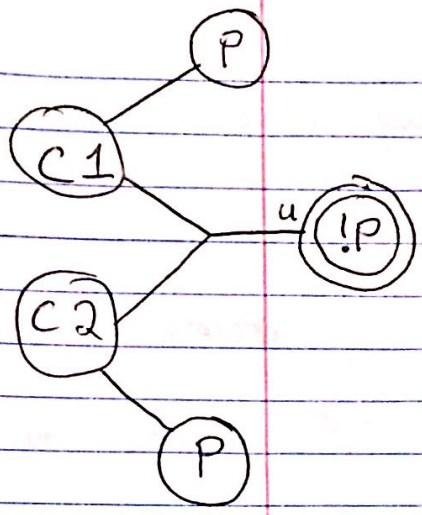
! corresponds to <u>process replication</u> : !P

in π calculus (usu. defined by structural congruence)

$$!P \cong P \mid !P$$



this is not right construct if you want sharing

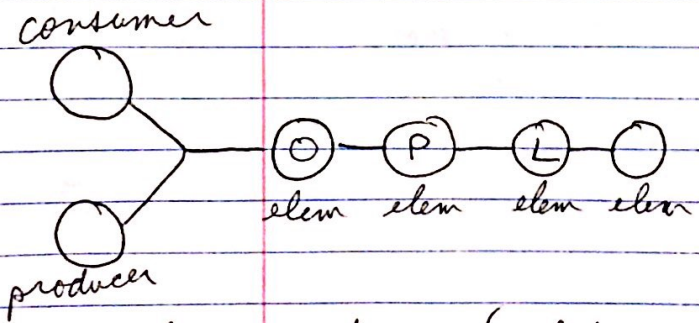


each client gets its own copy

goal:

introduce sharing in a logically moderated way

Manifest Sharing [Balzer & Pfenning 2017]



multi-producer (although not in above example)
multi-consumer queue

What do we do with preservation?

what did linearity give us?

exactly one client per provider

ensures client interacts w/ provider in isolation

Can we use another concept to achieve same thing?

ensure client communicate w/ mutual exclusion

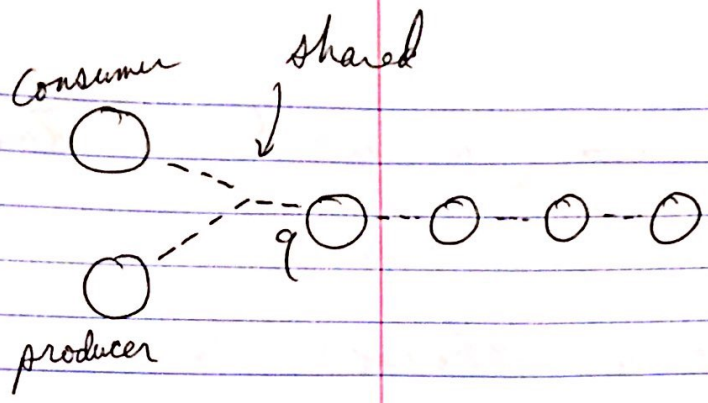
introduce notion of:

- shared
- linear

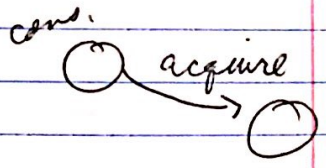
} channel

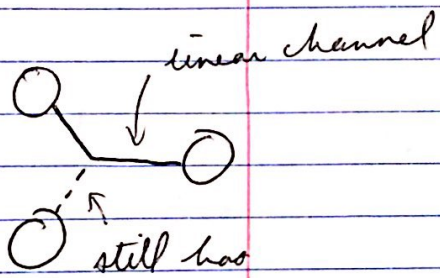
shared - ensure who have exclusive ~~role~~ access

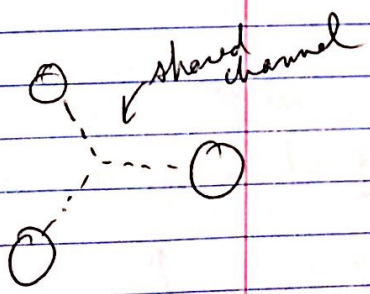
Then it becomes linear available to interact



can acquire q (queue)





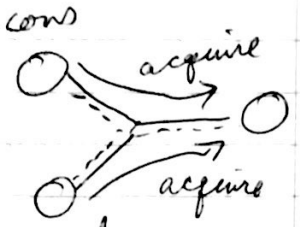


what if cons & prod. acquire @ same time?

one will wait

non-deterministically chosen

(only source of non-det in lang)



non-determinism

(now can simulate π-calc)

couldn't in linear setting

What is the type of q?

need to enrich notion of type structure with shared session

$$\text{queue } A = \downarrow\,\&\,\{\underbrace{\text{enq}}_{\text{acquire}} : A \multimap \uparrow \text{queue } A,\ \underbrace{\text{deq}}_{\text{release}} : \oplus\{\text{none} : \uparrow \text{queue } A,\ \text{some} : A \otimes \uparrow \text{queue } A\}\}$$

shared

linear

}

process
~~faced~~ phased
(linear / shared)

expand type system; intro shared

2 kinds of types
↙ linear proposition (↑ returns shared)

$A_s ::= \uparrow_L^S A_L$

$A_L, B_L ::= \oplus\{\overline{l : A_L}\} \mid A_L \otimes B_L \mid 1$ pos. conn

$\&\{\overline{l : A_L}\} \mid A_L \multimap B_L \mid \downarrow_L^S A_S$ neg conn.

↖ shared proposition

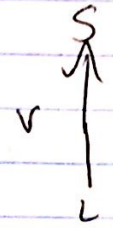
(could keep ! which
would give copying
and sharing semantics
in same language)

all acquire does is take us
from one layer to
another
release (linear → shared)

express acquire/release as modalities

stratified
can generalize adjoint formulation, could add affine, etc.

set up an ordering between layers



shared cannot rely on something lower in hierarchy
other ~~not~~ way is possible

(higher level, eg. has weakening, sharing which would mess with linear props, so not allowed)

queue $A_S = \uparrow^S_L \& \{$ enq : $A \multimap \downarrow^S_L$ queue A_S,

deq : $\oplus \{$ none : $\downarrow^S_L$ queue A_S,

some : $A \otimes \downarrow^S_L$ queue A_S $\}$ $\}$

can only store shared resources inside queue

need to send/rec shared channels along linear ones

$A_L, B_L ::= \ldots \mid \exists x : A_S . B_L$

$\mid \Pi x : A_S . B_L$

$\text{queue}\, A_S = \uparrow_L^S \& \{ \text{enq} : \Pi x : A_S . \downarrow_L^S \text{queue}\, A_S,$

$\text{deq} : \oplus \{ \text{none} : \downarrow_L^S \text{queue}\, A_S,$

$\text{some} : \exists x : A_S . \downarrow_L^S \text{queue}\, A_S \}$

$\}$

to type a linear process

use Γ (has sharing semantics)

linear $\Gamma, \Delta \vdash P :: (x_L : A_L)$

shared $\Gamma \vdash P :: (x_S : A_S)$

Δ linear

Γ structural w/ sharing semantics

have weakening, contraction

so far linear references

now we get aliasing with its problems

$$\frac{\Gamma, x_S : \uparrow_L^S A_L \;;\; \Delta, x_L : A_L \vdash Q_{x_L} :: (z_L : C_L)}{\Gamma, x_S : \uparrow_L^S A_L \;;\; \Delta \vdash x_L \leftarrow \text{acquire } x_S ; Q_{x_L} :: (z_L : C_L)} \; (\uparrow_L^S - L)$$

contraction at work (x_S not consumed)

point of view of ~~producer~~ provider; empty linear context

$$\frac{\Gamma ; \cdot \vdash P_{x_L} :: (x_L : A_L)}{\Gamma \vdash x_L \leftarrow \text{accept } x_S ; P_{x_L} :: (x_S : \uparrow_L^S A_L)} \; (\uparrow_L^S - R)$$

processes go back & forth between modes: linear & shared

release)

$$\frac{\Gamma, x_S : A_S ; \Delta \vdash Q_{x_S} :: (z_L : C_L)}{\Gamma ; \Delta, x_L : \downarrow_L^S A_S \vdash x_S \leftarrow \text{release } x_L ; Q_{x_S} :: (z_L : C_L)} \; (\downarrow_L^S - L)$$

↑ (on top)

gone, because of linearity

(w/o this everything would break down)

$$\frac{\Gamma \vdash P_{x_S} :: (x_S : A_S)}{\Gamma ; \cdot \vdash x_S \leftarrow \text{detach } x_L ; P_{x_S} :: (x_L : \downarrow_L^S A_S)} \; (\downarrow_L^S - R)$$

linearity requires us to consume all resources before detaching

not preservation (... yet)

one thing missing

what if (I acquire, use, release queue)

someone else releases queue at different spot

get a failure of preservation

not coordinated

need <u>well-formed condition</u> on session type

if I acquire shared resource, have to release at the same type I acquired it

called ⇔ <u>equisynchronizing</u>

(like equirecursive types)

then we get <u>preservation</u> ✓

(can relax a bit by adding subtyping)

What about <u>progress</u>? w/ sharing, get weaker form of progress ⇝ can get cycles (can have reference to self (cf. aliasing))

example: process linear

tries to acquire itself

deadlock

has to permit for a process to be blocking

current research to get rid of it

back in full π-calculus

(eg. can encode dining philosophers)

Paper present Sept at Concur

- encoding asynchronous untyped π-calculus into this language (shared session types, process calculi)

Conclude

curry - Howard correspondence?

linear prop	session types
proofs	processes
interleaving of	
· proof construction	acquire
· proof reduction	communication
· proof deconstruction	release

deadlock; failure of proof construction