Balzer 1

Session typed concurrent Programming

parallel program - reason sequentially
deterministic, w/ cost model

here deal with non-determinism

Roadmap (top-down
iterative deepening)

- message-passing concurrent prog.
- session types
- linear logic - session types (connection)
- manifest sharing (relax linear restriction)

learning objectives

(make linear logic less abstract)

- how can we program using
  message-passing concurrency?
- what session types are about
- benefits of linear logic to programming
  (more than inference rules)

adds restrictions, some too limiting

trying to derive a logically-motivated notion of sharing in prog lang

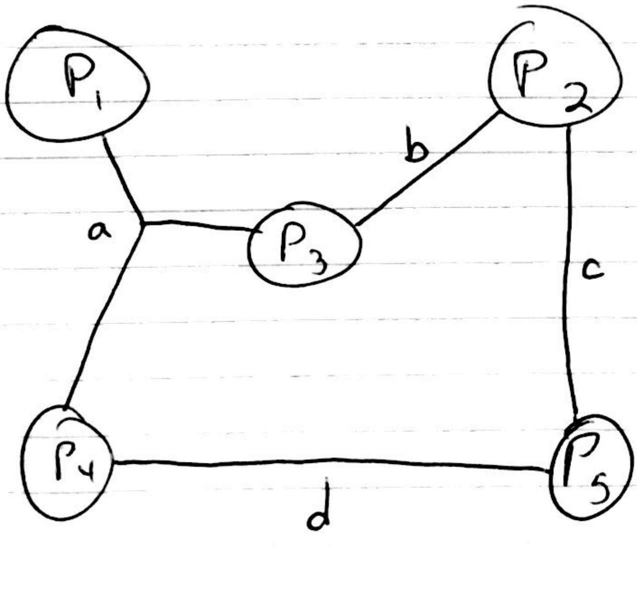
- how to accomodate sharing in a logically motivated lang.

Message-passing concurrency.

think comp.

bunch of processes exchang messages w/ each other

processes that compute by exchanging messages along channels



processes $P_1, \dots, P_5$

channels $a, \dots, d$ connect (can have more than 2 processes)

<u>n-ary channels</u> (eg. channel a shared among P_1, P_3, P_4)

we have <u>non-determinism</u>
(no longer parallel)
different model

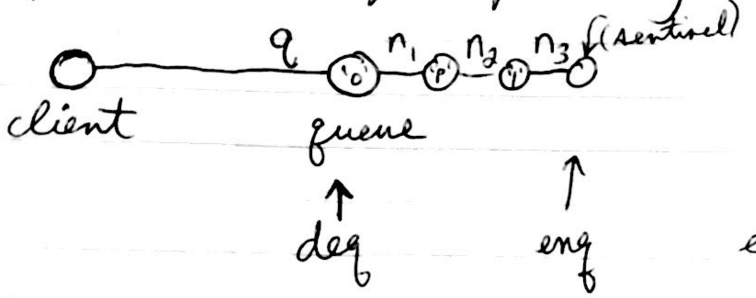
if P_1 sends along a,

either P_4 or P_3 can receive
↖(exclusive)

formal underpinning are process calculi
in particular the <u>π-calculus</u>
(Robin Milner) 1992

shown to be <u>universal</u>
(can encode λ-calculus in π-calculus)

example: <u>message - passing queue</u>



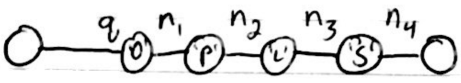
assume queue stores characters

each character has its own process

enqueue

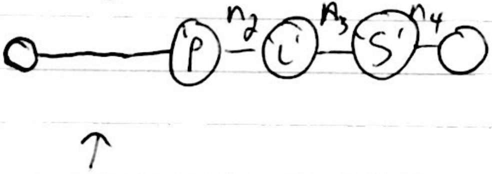
linked structure of processes

↓ enq 'S'



internally in queue, links processes w/ each other

↓ deq.



done by forwarding (form of identity logic)

create new channel q'

equate q w/ q'

- everywhere we have occurrence of q w/ q' $[q'/q]P$

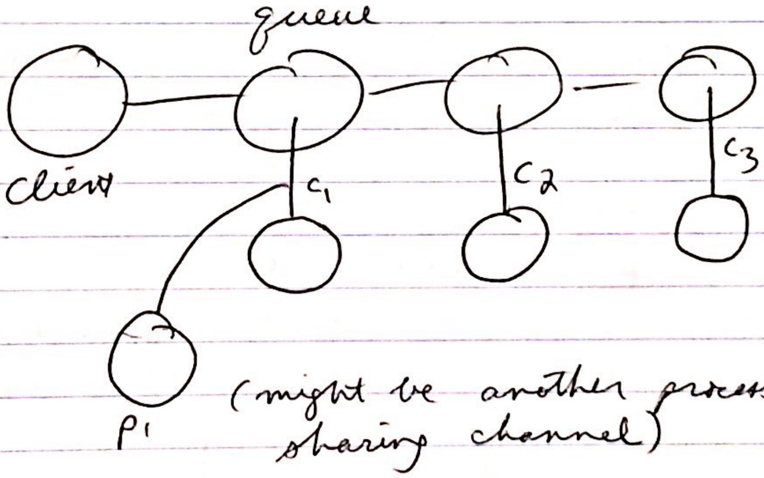
- another possibility - keep process alive, keep forwarding message sending labels eng, deg

values 'S'

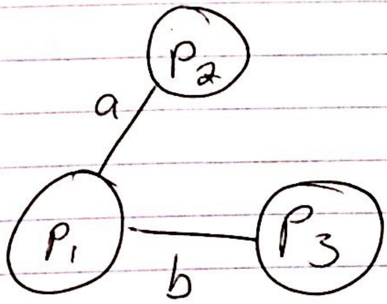
in π-calc, no support of basic values

messages are channels themselves or references to channels (more precisely)

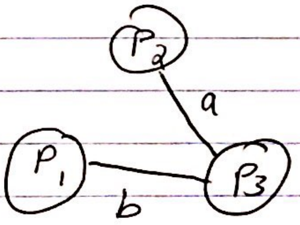
similarly to passing reference to object



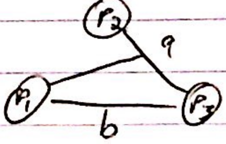
(might be another process sharing channel)



send a along b } send b a



if P_1 keeps reference to q:



"mobility" in π - calculus

"higher order channels" - channels can transport channels

How can we <u>type</u> message-passing protocols? <u>session</u> <u>types</u>

<u>Types for protocols of message exchange</u>

session types (Kohei Honda 1993)

$A ::= ?[T].A'$ ← (willing to receive input type T, continue to be type A') ABCD session types

$\mid\ ![T].A'$ ← (willing to send type T, afterwards type A') need send, receive, indicate choice (eg enq, deq)

external (to process) $\mid\ \&\{l_1{:}A_1, \dots, l_n{:}A_n\}$ ("external" choice (client chooses)) pick any labels $l_1, \dots, l_n$ afterward continues as $A_1, \dots, A_n$

internal (by process itself) $\mid\ \oplus\{l_1{:}A_1, \dots, l_n{:}A_n\}$ dual of &; "internal" choice. client must be prepared to deal w/ choice

$T ::= A \mid \text{int} \mid \text{char} \mid \dots$ (depends on particular formalization)

$\mid\ \text{end}$ (terminate / kill process)

$\mid\ X$ type argument (for polymorphism for example)

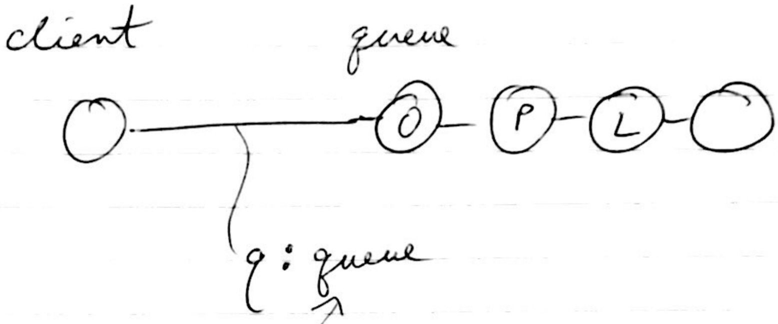
$\mid\ \mu.X.A'$ (recursive process)

What is type of queue

$\text{queue} = \&\{\text{enq} : ?[\text{char}].\,\text{queue},$

$\quad \text{deq} : \oplus\{\text{none} : \text{end},\ \text{some} : ![\text{char}].\,\text{queue}\}$

$\}$



$q : \&\{\text{enq} : \ldots,\ \text{deq} : \ldots\}$ send 'enq' along q

$q : ?[\text{char}].\,\text{queue}$ send 's' along q

$q : \text{queue}$

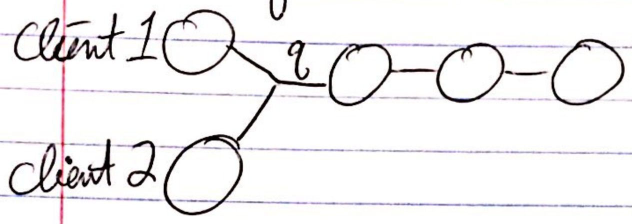
takeaway/observation: type of channel/process changes with message exchange

session types tell us about protocol
protocol change
session types change as we compute

note type safety

preservation in particular

What if ?



q : queue

client 1 sends 'enq' along q

q : ?[char]. queue

how does client 2 know about this?

client2 wants to send 'deq' along q

(presewation compromised here)

What does presewation mean in session typed setting?

view or expectations of client
align with ~~presentation~~ expectation of provider

"session fidelity"
guarantees ~~t~~that expectation of client matches with the one of provider if they do match initially

as it, is, current state is not
type safe

Can we do something about this?

what about linearity?
treating something as a resource
don't drop
consume everything

what are our resources?
channel

base session types on linear logic
(cast into π)

treat channels as a resource
what does it mean?

linear logic rejects:

- weakening
- contraction

what does it mean for sessions?
process graphs

what kind of structures do we get?