HW6: CSP Channels
Prof. Marko Schutz
INSO 4101

Exercise 21.1. *System Channel Configurations*. Please review Sect. 21.2.9, and suggest formal specifications of P, P_i, Q, and Q_j function definition schema and channel structures to cater to each of the five ([A–E]) system channel configurations.

Hint: You are to assume that the functions P, P_i, Q and Q_j do not interact, i.e., engage in events, with other processes.

System Channel Configuration for [A] Common Channel

A

type
    Info
channel
    c: Info,
value
    P: **Unit** → **out** c **Unit**
    Q: **Unit** → **in** c **Unit**
    P()≡ **let** i = c ! in write_to_sys(i) **end;** P()
    Q()≡ **let** i = c ? in read_from_sys(i) **end;** Q()

    write_to_sys: **Unit** → Info
    read_from_sys:Info → **Unit**

    A: **Unit** → **Unit**; A()≡ P() || (||{Q() | x:{1..n}})

B

type
    Index, Info
channel

    c[1,…,m]:Info
value
    P: **Unit** → **out** {c[idx]:Index} **Unit**
    Q: Index x **Unit** → **in** c[Index] **Unit**
    Q()≡ []{**let** i = write_to_sys(i) in c[idx] ! **end| idx:Index};** Q()

P()≡ **let** i = {c[idx] ? in read_from_sys(i)|idx:Index} **end;** P()

write_to_sys: **Unit** → Info
read_from_sys:Info → **Unit**

B: **Nat x Unit** → **Unit**; B(n)≡|| {P() || Q() x:{1,…,n}}

C

type
Info
channel

c:Info
value
P: **Unit** → **out** {c[idx]:Index} **Unit**
Q: Index x **Unit** → **in** c[Index] **Unit**
Q()≡ **let** i = write_to_sys(i) in c ! **end;** Q()

P()≡ **let** i = c[] ? in read_from_sys(i) **end;** P()

write_to_sys: **Unit** → Info
read_from_sys:Info → **Unit**

C: **Nat x Nat x Unit** → **Unit**; C(m,n)≡|| {P()|x:{1,…,m} || Q() | y:{1,
…,n}}

D

type
Index, Info
channel

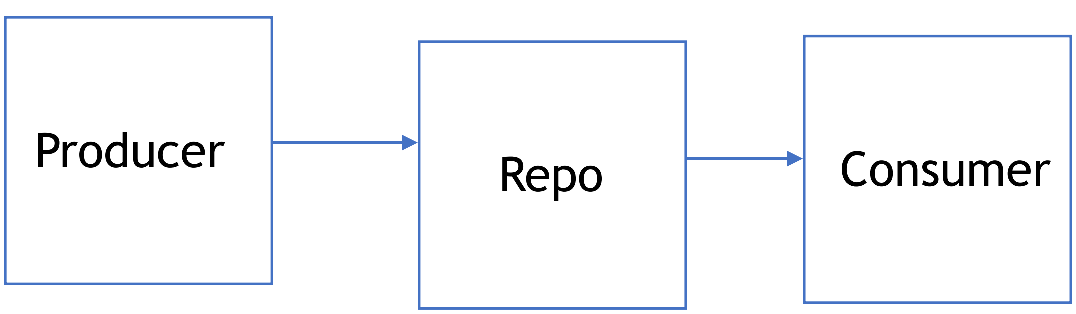
c[1,…,m]:Info
value
P: **Unit** → **out** {c[idx]:Index} **Unit**
Q: Index x **Unit** → **in** c[Index] **Unit**
Q()≡ []{**let** i = write_to_sys(i) in c[idx] ! **end| idx:Index};** Q()
P()≡ **let** i = {c[idx] ? in read_from_sys(i)|idx:Index} **end;** P()

write_to_sys: **Unit** → Info
read_from_sys:Info → **Unit**

```
            D: Nat x Nat x Unit → Unit; B(m,n)≡|| {P()y:{1,…m} || Q() x:{1,…,n}}
E

      type
              Client,Index, Info
      channel

            c[1,…,m]:Info, clt:Client
      value
            P: Unit → out clt Unit
            Q: Index x Unit → in c[Index] Unit
            []{clt||c[idx] | idx:Index}
            Q()≡ []{let i = write_to_sys(i) in c[idx] !  end| idx:Index}; Q()

            P()≡ let i = clt ? in read_from_sys(i) end; P()

            write_to_sys: Unit → Info
            read_from_sys:Info → Unit

            E: Nat x Nat x Unit → Unit; B(m,n)≡|| {P()y:{1,…m} || Q() x:{1,…,n}}
```

Exercise 21.2. *Single Producer-/Consumer-Bounded Repository.* There are given three behaviours: a producer, a repository, and a consumer. The producer (occasionally) produces entities and delivers them to the repository. The repository accepts producer-manufactured entities and, upon request, hands them on to the consumer. The consumer consumes entities by (occasionally) requesting these from the repository. The repository delivers entities in the order in which they were received. The repository can keep at most b entities.

Define types of entities and of entity requests (from consumers), two (or three) channels and the four behaviours: producer, repository, consumer and their aggregation into a system behaviour.



Procuder, Repo and Consumer

type

Stuff

channel

c1: Stuff, c2: Stuff

value

Produce: **Unit** → **out** c1 **Unit**
Repo: **Unit** → **in** c1 **out** c2 **Unit**
Consumer: **Unit** → **in** c2 **Unit**
produce_stuff:**Unit** **—>** Stuff
get_stuff:Stuff **—>** **Unit**
Producer()≡ **let** i = produce_stuff(i) ! **in** c1 **end;** Producer()
Repo()≡ **let** i = c1 ? **in** get_stuff(i) write_stuff(i) **out** c2 **end;** Repo()
Consumer()≡ **let** i = c2 ? **in** read_from_sys(i) **end;** Consumer()

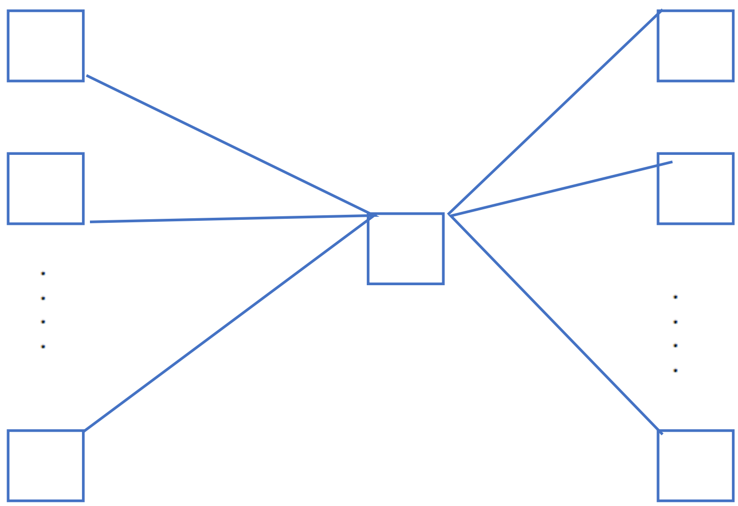
write_to_sys: **Unit** → Info
read_from_sys:Info → **Unit**

Prod_Cons_Sys: **Unit** → **Unit**;
Prod_Cons_Sys()≡ Producer() || Repo() || Consumer()

Exercise 21.3. *Multiple Producer-/Consumer-Bounded Repository.* We refer to Exercise 21.2. All you need, for this exercise, is to read the formulation of that exercise.

There are given $m + n + 1$ behaviours: m producers, p_i, a repository, and n consumers, c_j. Any producer may deposit an entity with the repository, and any consumer may request an entity from the repository. The repository marks every received entity with a unique identity of its producer. The entities delivered to consumers are marked with this identity. The repository otherwise delivers the marked entities in the order of their receipt.

Define types of entities and of entity requests (from consumers), the m channels between producers and the repository, the either n or $2n$ channels between the repository and the consumers, and the four behaviours: producer, repository, consumer and their combined system.



type
Index, Stuff
channel

c1[1,…,m]:Info, c2[1,…,n]
value
send_stuff:**Unit** —> Stuff
get_stuff:Stuff —> **Unit**

Producer: **Unit** → **out** {c1[idx]:Index} **Unit**
Repo: Index x **Unit** → **Unit in** {c1[idx]:Index} **out {c2[idx]:Index} Unit**

Consumer: **Unit → in** {c1[idx]:Index} **Unit**

Producer()≡ []{**let** i = write_to_sys(i) in c1[idx] ! **end| idx:Index};** Producer()

Repo()≡ []{**let** i =c1[idx] ? in get_stuff(i) send_stuff(i) in c2[idx] ! **end| idx:Index};** Repo()

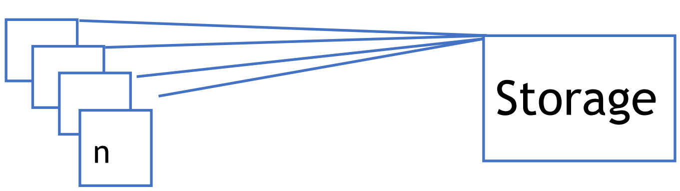
Consumer()≡ []{**let** i = {c2[idx] ? in read_from_sys(i)|idx:Index} **end;** Consumer()

write_to_sys: **Unit →** Info
read_from_sys:Info **→ Unit**

Sys: **Nat x Unit → Unit**; Sys(n)≡|| {Producer x:{1,..m}||Repo() || Consumer() x:{1,…,n}}

Exercise 21.4. *Shared Storage.* A number of computation processes share a common storage. We see this common storage to record, for distinct locations, a value. We see the computation processes performing the following operations on the shared storage: (i) requesting allocation of new storage locations, (ii) storing (initial) values in these, (iii) updating with, i.e., changing existing values to, new given values at identified, i.e., given, locations; (iv) requesting the value at a given (i.e., identified) location, (v) and requesting the deallocation, i.e., the freeing or removal, of an identified location. We finally allow processes to (vi) pass on locations to one another — according to some further unspecified protocol.

Define the type of storages, i.e., of locations and values and their combination into storages. Define the type of channels between computation processes and between these and the storage process — the latter is thus thought of as the only process which "keeps", i.e., maintains, the storage. Finally, define the two kinds of behaviours: computation processes which occasionally perform one of the actions (i–vi), and the storage behaviour.



type

Loc, Value, CompNum,Prot
MSG == reqLoc(Loc)
|storeVal(Value)
| updateVal(Loc)
|reqProtocol(Loc)
|reqDeAlloc(Loc)
|mkVal(Value)

channel

ch[n:CompNum]:MSG

value

value
reqLoc(Loc)—> Loc // request location
storeVal(Val) —> **Bool** // Confirm storage was successful.
updateVal(Loc) —> Val // Update value and returns the value
reqProtocol(Loc) —> CompNum x Prot
reqDeAlloc(Loc) —> Value // De-allocate and get value at location
mkVal:Value —> **Bool** // Make a value and return succesfull
P: **Unit** → **out {**c[idx]:Index} **Unit**
Q: Index x **Unit** → **in** c[Index] **Unit**
Q()≡ **[]**{**let** i = write_to_sys(i) in c[idx] ! **end| idx:Index};** Q()

P()≡ **let** i = {c[idx] ? in read_from_sys(i)|idx:Index} **end;** P()

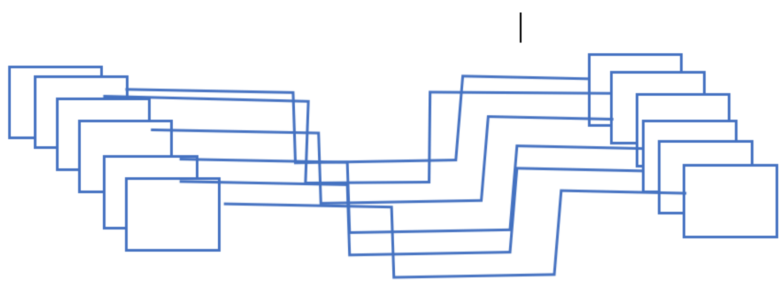
write_to_sys: **Unit** → Info
read_from_sys:Info → **Unit**

Storage_sys: **Nat x Unit** → **Unit**; B(n)≡|| {P() || Q() x:{1,…,n}}

Exercise 21.7. *Synchronous Multiple-Client/Multiple-Server System:* We refer to Exercise 21.5. You need not have solved that exercise, but you need to have read its problem formulation before you now read on.

The only difference between the problem of the present exercise and that of the referenced exercise is that there are now many servers, i.e., more than one. Any server is ready to accept any action request, and all servers serve the same actions.

Define the types of server function catalogues and states, and of client-to-server and server-to-client messages. Also define the system, the client and the server behaviours.



type

Info,Client Server, Index

channel

c:Info

value

Client: **Unit** → **out** {c[idx]:Index} **Unit**
Server: Index x **Unit** → **in** c[Index] **Unit**
Server()≡ **let** i = write_to_sys(i)! out c **end;** Q()

Client()≡ **let** i = c ? in read_from_sys(i) **end;** P()

Client_to_Server: **Unit** → Info
read_from_sys:Info → **Unit**

// Server Comm Room is the overall server communication room and
// final implementation of the CSP
ServerCommRoom: **Nat x Nat x Unit** → **Unit**;
ServerCommRoom(m,n)≡|| {P()|x:{1,…,m} || Q() | y:{1,…,n}}

Time Log

13/Sep/2018

date	start	stop	errupt	net	act	comment	mpleted	nits
10-Sep	7:25	9:00		95	repare	read news, breakfast		
	9:00	9:30		30	park	find parking space		
	9:30	0:20	10	40	class	re and waisting time on twitter		
	0:30	2:10		40	eat	lunch		
	2:30	1:20		50	class	lecture		
	1:30	3:00	10	80	search	read assigned papers	x	2
	3:00	5:00	30	90	study	tructions for HW3, break, phone		
	5:00	6:00	30	30	study	read ch3	x	1
	6:00	7:30		90	class	lecture		
	7:00	8:30	20	70	prog	research team & chat with team	x	2
	8:40	1:30	20	50	study	quiz prep, chat, leisure	x	1
date	start	stop	errupt	net	act	comment	mpleted	nits
11-Sep	7:25	8:30		55	repare	read news, breakfast		
	8:30	8:55	10	15	park	rking time and chat with friends		
	9:00	0:20		80	class	lecture		
	0:30	2:20		10	search	research meeting		
	2:30	1:15		45	eat	lunch		
	1:15	4:30	30	65	study	uiz prep, began reading chapter	x	1
	4:30	4:55	15	15	prog	read requirements for project		
	5:00	6:30		90	class	lecture		
	6:40	7:20		40	eat	supper		
	7:30	0:40	40	50	study	read ppt before class		
	0:50	2:00		70	ercise	go for a run		
date	start	stop	errupt	net	act	comment	mpleted	nits
12-Sep	7:25	9:00	10	85	repare	read news, breakfast		
	9:00	9:25		25	park	find parking space		
	9:30	0:20	2	48	class	lecture		
	0:30	2:05		95	eat	unch with friends and colleagues		
	2:06	2:25	10	9	prog	arch code and chat with friends		
	2:25	1:27		62	class	lecture		
	1:30	4:35	30	55	search	meeting and programming		
	4:35	4:50		15	eat	coffee and waist time on twitter		
	4:50	6:10		80	study	prep, read notes before lecture	x	1
	6:30	7:20		50	class	lecture		
	7:20	0:30	20	70	prog	HW	x	1
	0:30	2:00	5	85	ercise	go for a run		
date	start	stop	errupt	net	act	comment	mpleted	nits

Weekly Activity Summary

week #	Task Date	Class	Prepare	Park	Eat	Study	Prog	Research	Exercise
2	M	180	95	30	40	270	70	80	0
3	T	170	55	15	85	315	70	110	70
4	W	160	85	40	15	80	179	155	85
5	T	170	65	20	40	270	70	0	69
8	Totals	680	300	105	180	935	389	345	224
9	Average	170	75	26.25	45	233.75	97.25	86.25	56
10	Min	160	55	15	15	80	70	0	0
11	Max	180	95	40	85	315	179	155	85

Proposed Schedule for New Tasks

Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
7:00							
8:00							
9:00							
10:00	Code Res		Code Res		Code Res		
11:00	Code Res		Code Res				
12:00	HW	HW	HW	HW	HW		
13:00	HW		HW		HW		
14:00							
15:00		HW		HW			
16:00							
17:00							

Categrory Percentages

Total Est Hr	Time	Class	Prepare	Park	Eat	Study	Prog	Research	Exercise
3980	Total	680	300	870	180	935	389	345	224
	Percentage	17%	8%	22%	5%	23%	10%	9%	6%