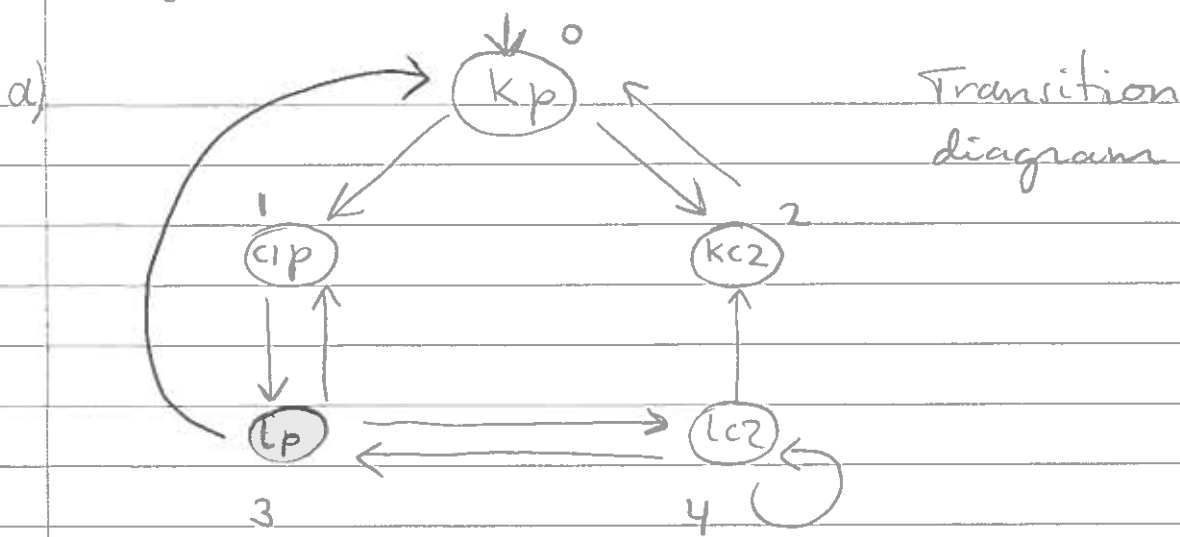


# Assignment 1

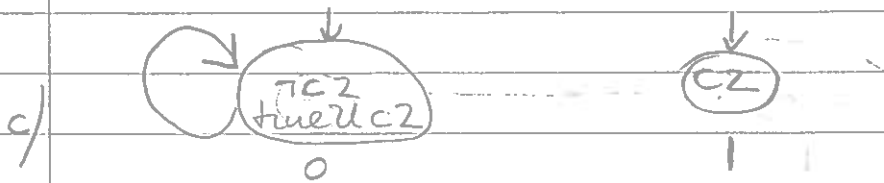


b)  $EFAFC1 = E(\text{true} \cup AFC1)$

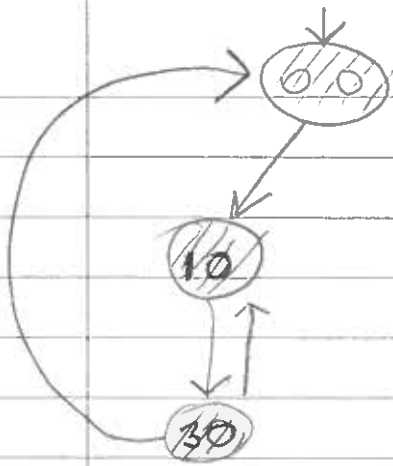
subformula	state	0	1	2	3	4
$c1$			✓			
$AFC1$	$S(\text{start})$		✓			
$AFC1$	$R(\text{repeat/terminate})$		✓			
$E(\text{true} \cup AFC1)$	$S$		✓			
"	$R$	✓	✓		✓	
"	$R$	✓	✓	✓	✓	✓
"	$RT$	✓	✓	✓	✓	✓

when 0 is labelled!  
you may stop

$EFAFC1$  labels state 0, so  
 $EFAFC1$  holds in the starting state.



by application of transition clause 3 on 0,  
 Street automaton  $((L, R) = (0, 1))$   
 because no clauses apply on 1.



$$\{(G, R)\} = \{(\{00, 10, 30\}, \emptyset)\}$$

Product automaton

The product automaton accepts no infinite sequences, as all states have to be avoided after some point. So the (imaginary) formula holds for the system.

d)

```
MODULE main
```

```
VAR
```

```
pr1 : process pr1(pr2.st);
```

```
pr2 : process pr2(pr1.st);
```

```
MODULE pr1(other-st)
```

```
VAR
```

```
st : {k, c1, l};
```

```
ASSIGN
```

```
init(st) := k;
```

```
next(st) :=
```

```
case
```

```
(st = k) & !(other-st=c2) : c1;
```

```
(st = c1) : l;
```

```
(st = l) & !(other-st=c2) : {c1, k}; -- as l to k is also
```

possible

```
(st = l) : k; -- as l to k is always possible
```

```
1 : st;
```

```
esac;
```

```
FAIRNESS running
```

```
MODULE pr2(other-st)
```

```
VAR
```

```
st: {p,c2};
```

```
ASSIGN
```

```
init(st) := p;
```

```
next(st) :=
```

```
case
```

```
(st=p ) & !(other-st=c1) : c2;
```

```
(st=c2) & !(other-st=k) : {c2, p}; -- as c2 to p is always
```

possible

```
(st=c2) : p;
```

```
1 :st;
```

```
esac;
```

```
FAIRNESS running
```

```
FAIRNESS !(st = c2)
```

## Assignment 4

a) To show  $\Box \neg(c_1 \wedge c_2)$  apply INV- $\Box$ .

$\Box 1. k \wedge p \rightarrow \neg(c_1 \wedge c_2)$   $\&$

$\Box 2. \neg(c_1 \wedge c_2) \tilde{\rightarrow} \neg(c_1 \wedge c_2)$

All transitions except  $k \xrightarrow{c_1} c_1, l \xrightarrow{c_2} c_1, p \xrightarrow{c_1} c_2$

and  $c_2 \xrightarrow{k} c_2$  lead to states other than  $c_1$  or  $c_2$ ,

so trivially fulfil  $\Box 2$ ,  $\&$

Consider transition  $k \rightarrow c_1$ .

The transition relation is as for await (Chap 0 p. 20)

i.e.,

$$\text{move}(k, c_1) \wedge \neg c_2 \wedge \text{pres}(\gamma)$$

so to prove:

$$\text{move}(k, c_1) \wedge \neg c_2 \wedge \text{pres}(\gamma) \wedge \neg(c_1 \wedge c_2) \rightarrow \neg(c_1 \wedge c_2)'$$

where  $\text{move}(k, c_1)$  gives  $\pi_1 = k \wedge \pi_1' = c_1 \wedge \text{pres}(\pi_2)$ .

This evidently holds, as although  $c_1'$ , also  $\neg c_2'$ .  $\&$

Transitions  $l \rightarrow c_1, p \rightarrow c_2$  are treated similarly.  $\&$

Consider transition  $c_2 \xrightarrow{k} c_2$ .

The transition relation now is not guarded by  $\neg c_1$  or  $\neg c_2$ , but the proof even simpler as

$$\text{move}(c_2, c_1) \wedge \text{pres}(\gamma) \wedge \neg(c_1 \wedge c_2) \rightarrow \neg(c_1 \wedge c_2)', \&$$

i.e.,  $\neg k$  is not relevant here.

b)  $k \Rightarrow \Diamond c_1$  holds because the transition

$k \rightarrow c_1$  is enabled infinitely often (when

process 1 is at  $k$ , and process 2 cycles,

process 2 is out of its  $c_2$  infinitely often),

which under compression means: taken.

CHAIN-C is the rule to use

$k \rightarrow c_1$  is the helpful transition

$k \Rightarrow \Diamond \text{En}(k \rightarrow c_1)$  is the crucial clause,

CHAIN- $\exists$  is used to prove this clause.