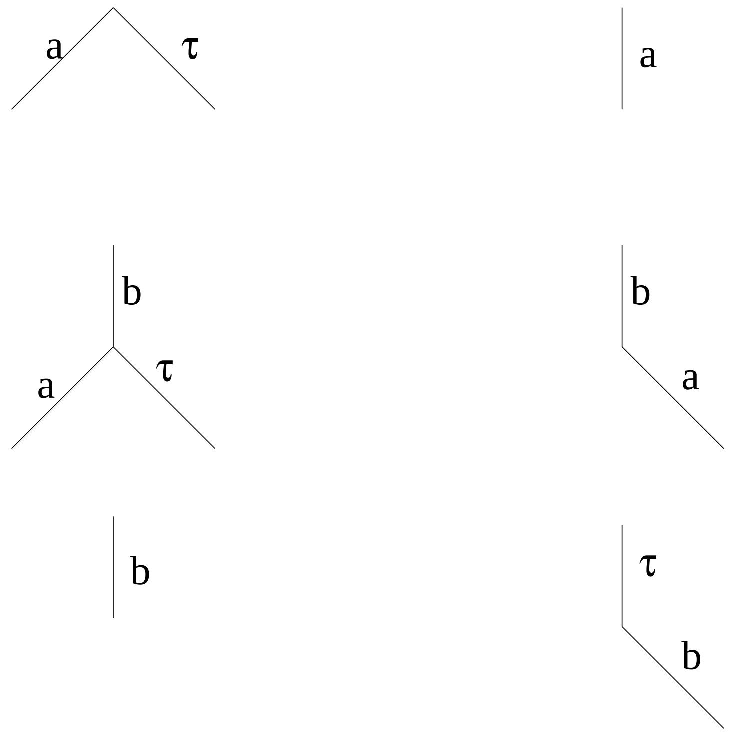
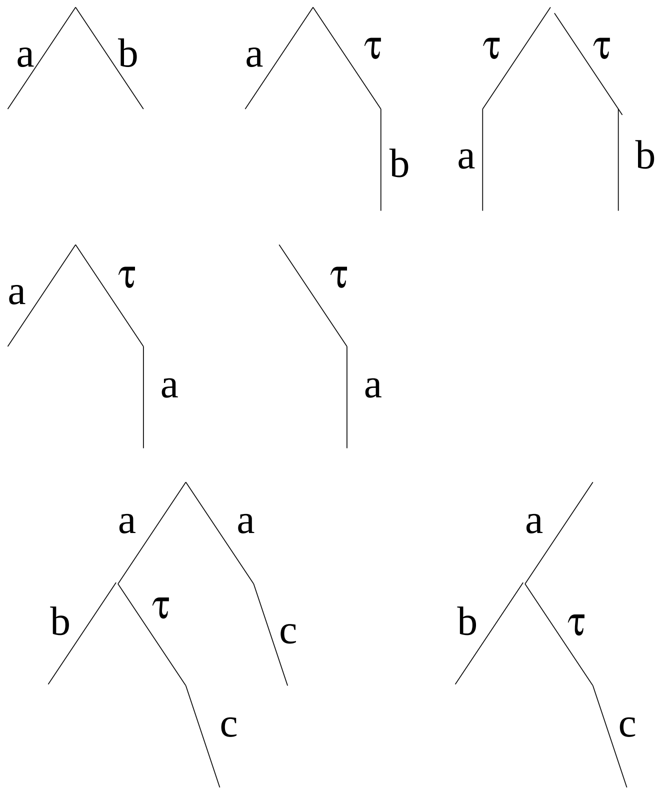
Observations and Weak Equivalence

- Observation = Reaction
  Pressing buttons of a black-box
- τ is unobservable
- Observationally equivalent systems: possible to press the same buttons in the same order
- We don't care about unobservable transitions mixed with the observed actions.

Are these equivalent?





Weak Bisimulation

Actions: “pressed buttons”
Reactions: internal transitions τ

Notation:
$P \Rightarrow Q$: some (maybe no) reactions ($\rightarrow^*$),
$P \Rightarrow^{a1,\ldots,a_n} Q$: actions in this order;
mixed with sequences of reactions.

- S is a **weak simulation** iff
  whenever $P\ S\ Q$:
  if $P \Rightarrow^{a1,\ldots,a_n} P'$, $n \geq 0$
  then there exists Q' such that
  $Q \Rightarrow^{a1,\ldots,an} Q'$ and $P'\ S\ Q'$
- S is a **weak bisimulation** iff:
  S and S^{-1} are weak simulations.
- P, Q are **weakly** or **observation equivalent** ($P \approx Q$).
  iff there exists a w. bisimulation S, s.that
  $P\ S\ Q$.

Checking Weak Simulation

S is a weak simulation iff
whenever $P\ S\ Q$:

- if $P \rightarrow P'$ (internal reaction) then there is Q' s.that $Q \Rightarrow Q'$ and $P'\ S\ Q'$
- if $P \rightarrow^{a} P'$ (observable action) then there is Q' s.that $Q \Rightarrow^{a} Q'$ and $P'\ S\ Q'$

Checking single reactions and actions is sufficient

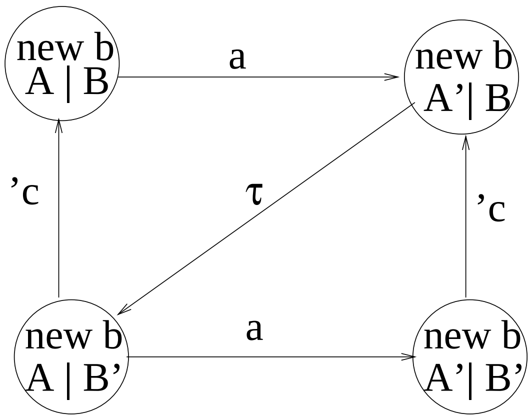
Weak Simulation: An Example

$A = a.A' \quad A' = {}'b.A$
$B = b.B' \quad B' = {}'c.B$

$newb(A|B) \approx E$
where

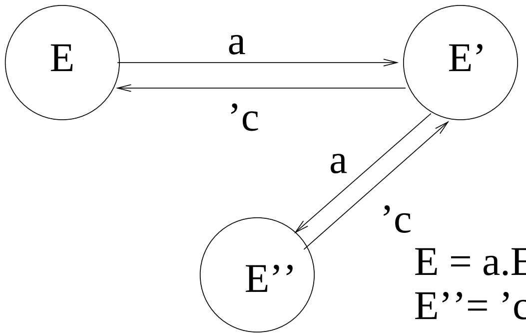
$E = a.E' \quad E' = a.E'' + {}'c.E \quad E'' = {}'c.E'$

new b (A | B)



A = a.A' A' ='b.A

B = b.B' B' = 'c.B



E = a.E' E' = A.E''+ 'c.E
E''= 'c.E'

Properties

- $P \sim Q$ implies $P \approx Q$
- $\approx$ is an equivalence relation
- $\approx$ is a weak bisimulation
- For any process P and any summations M, N:
  1. $P \approx \tau.P$
  2. $M + N + \tau.N \approx M + \tau.N$
  3. $M + \alpha.P + \alpha.(\tau.P + N) \approx M + \alpha.(\tau.P + N)$
- $\approx$ is a process congruence
  That is, if $P \approx Q$ then
  (*replacement of equivalent expressions gives equivalent expressions in the following contexts*)
  1. $\alpha.P + M \approx \alpha.Q + M$ (M any sum)
  2. *new* a $P \approx$ *new* a Q
  3. $P|R \approx Q|R$
  4. $R|P \approx R|Q$

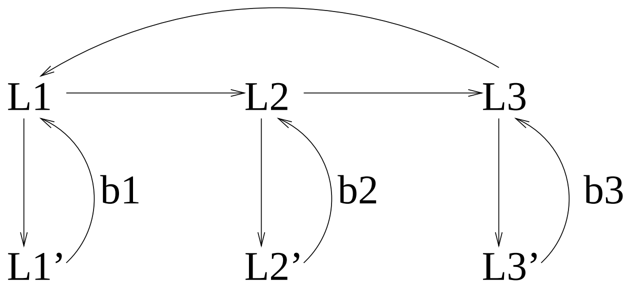
Lottery

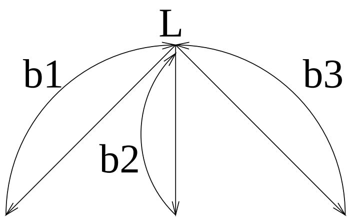
Specification:

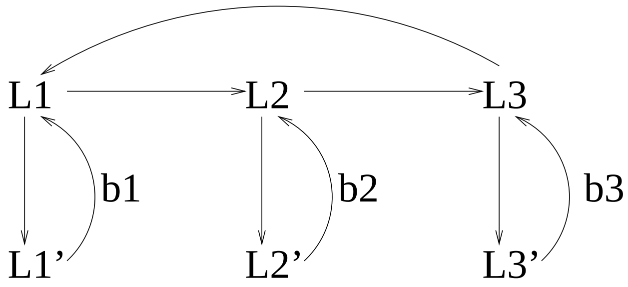
- n balls ($n = 3$)
- choose randomly a ball - repeat the drawing

Implementation idea:

- n concurrent cells for different balls
- A cell :
  - accepts activation signal from the left neigbour, then
  - may give a ball arbitrary number of times
  - sends activation signal to the right neigbour







```
proc Lotspec =
  . t.b1.Lotspec + t.b2.Lotspec + t.b3.Lotspec

proc Cell  = 'a.C
proc C     = t.Ball + c.Cell
proc Ball  = b.C

proc A1 = (Cell)[a1/a,b1/b,a2/c]
proc B1 = (Ball)[a1/a,b1/b,a2/c]
proc C1 = (C)[a1/a,b1/b,a2/c]

proc A2 = ...
proc A3 = ...

proc L = (C1|A2|A3)\{a1,a2,a3}
----
cwb-nc> eq -S obseq "Lotspec" "L"
Building automaton...
States: 15
Transitions: 23
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
```



```
cwb-nc> eq -S bisim "Lotspec" "L"
...
FALSE...
Lotspec satisfies:
[t][t]ff
L does not.
```

Scheduler

Specification (operations modulo n)

- Processes $P_1, ..., P_n$ ($n = 3$),
- activated cyclically $a_1, ..., a_n$
- possibly running concurrently
- An process P_i
  - is active or inactive
    (no multiple activations)
  - if active, will end e_i some time

How to describe this in CCS?

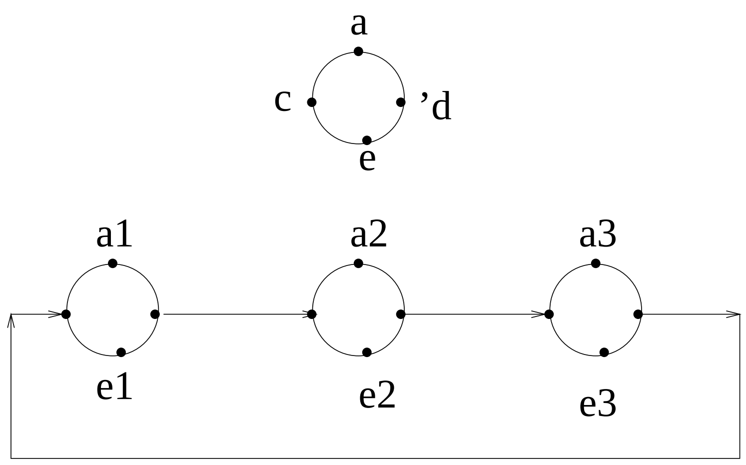

```
proc S1000 = a1.S2100
proc S1111 = e1.S1011 + e2.S1101 + e3.S1110
proc S1110 = e1.S1010 + e2.S1100
proc S1101 = e1.S1001 + e3.S1100
proc S1100 = e1.S1000
proc S1011 = a1.S2111 + e2.S1001 + e3.S1010
proc S1010 = a1.S2110 + e2.S1000
proc S1001 = a1.S2101 + e3.S1000

proc S2000 = a2.S3010
proc S2111 = e1.S2011 + e2.S2101 + e3.S2110
proc S2110 = e1.S2010 + e2.S2100
proc S2101 = a2.S3111 + e1.S2001 + e3.S2100
proc S2100 = a2.S3110 + e1.S2000
proc S2011 = e2.S2001 + e3.S2010
proc S2010 = e2.S2000
proc S2001 = a2.S3011 + e3.S2000

proc S3000 = a3.S1001
proc S3111 = e1.S3011 + e2.S3101 + e3.S3110
proc S3110 = a3.S1111 + e1.S3010 + e2.S3100
proc S3101 = e1.S3001 + e3.S3100
proc S3100 = a3.S1101 + e1.S3000
proc S3011 = e2.S3001 + e3.S3010
proc S3010 = a3.S1011 + e2.S3000
proc S3001 = e3.S3000
```

Scheduler Implementation

- Create a cell for each process
- Activated by its left neighbour
- Activates its right neighbour
- "Neighbours" modulo n



CWB Scheduler Implementation

```
proc A = a.C
proc C = c.E
proc E = e.D
proc D = 'd.A

proc R1 = A[a1/a,c1/c,e1/e,c3/d]
proc R2 = D[a2/a,c2/c,e2/e,c1/d]
proc R3 = D[a3/a,c3/c,e3/e,c2/d]

proc Scf = (R1|R2|R3)\{c1,c2,c3}
proc Scs = S1000
----
cwb-nc> eq Scf Scs
Building automaton...
States: 55
Transitions: 99
...
FALSE...
Scf satisfies:
[[a1]][[a2]][[e2]][[a3]][[e3]]ff
Scs does not.
```

What goes wrong??

```
Scf satisfies:
[[a1]][[a2]][[e2]][[a3]][[e3]]ff
Scs does not.
```

This concerns synchronisation between $R3$ and $R1$

```
*R1:
proc A1 = a1.C1
proc C1 = c2.E2
proc E3 = e1.D2
proc D1 = 'c3.A1

*R3:
proc D3 = 'c2.A3
proc A3 = a3.C3
proc C3 = c3.E3
*blocked until R1 reacts on c3
proc E3 = e3.D3
```

$R1$ did not finish; can't accept $c3$

Correct Scheduler

```
proc A1 = a.C1
proc C1 = c.E1
proc E1 = e.D1  +'d.F1
*can finish before reacting
proc F1 = e.A1
proc D1 = 'd.A1

proc P1 = A1[a1/a,c1/c,e1/e,c3/d]
proc P2 = D1[a2/a,c2/c,e2/e,c1/d]
proc P3 = D1[a3/a,c3/c,e3/e,c2/d]

proc Sc = (P1|P2|P3)\{c1,c2,c3}
----
cwb-nc> eq Scs Sc
Building automaton...
States: 67
Transitions: 129
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
```