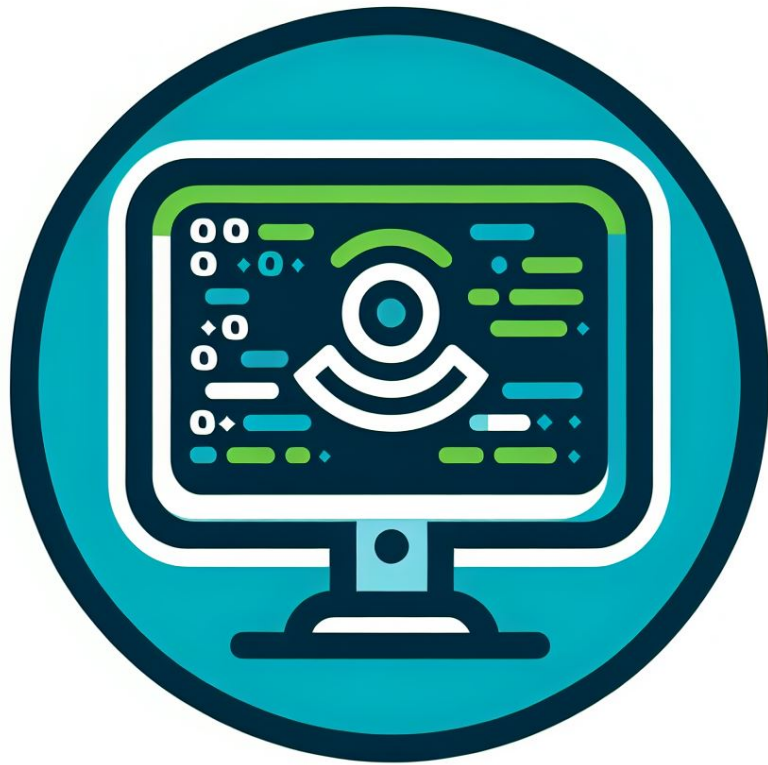




**PSC 2024/25** (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni

<http://www.di.unipi.it/~bruni/>

[http://didawiki.di.unipi.it/doku.php/  
magistraleinformatica/psc/start](http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start)

21 - CCS at work

# CCS syntax

|        |       |                       |                                  |
|--------|-------|-----------------------|----------------------------------|
| $p, q$ | $::=$ | <b>nil</b>            | inactive process                 |
|        |       | $x$                   | process variable (for recursion) |
|        |       | $\mu.p$               | action prefix                    |
|        |       | $p \backslash \alpha$ | restricted channel               |
|        |       | $p[\phi]$             | channel relabelling              |
|        |       | $p + q$               | nondeterministic choice (sum)    |
|        |       | $p   q$               | parallel composition             |
|        |       | <b>rec</b> $x. p$     | recursion                        |

(operators are listed in order of precedence)

# Some notation

write  $\sum_{i=1}^n p_i$  instead of  $p_1 + \cdots + p_n$

write  $\prod_{i=1}^n p_i$  instead of  $p_1 | \cdots | p_n$

write  $p \setminus \{a_1, \dots, a_n\}$  instead of  $p \setminus a_1 \cdots \setminus a_n$

write  $\mu^n.p$  instead of  $\underbrace{\mu.\mu.\dots.\mu}_n.p$

# CCS op. semantics

$$\begin{array}{lll}
 \text{Act)} \frac{}{\mu.p \xrightarrow{\mu} p} & \text{Res)} \frac{p \xrightarrow{\mu} q \quad \mu \notin \{\alpha, \bar{\alpha}\}}{p \setminus \alpha \xrightarrow{\mu} q \setminus \alpha} & \text{Rel)} \frac{p \xrightarrow{\mu} q}{p[\phi] \xrightarrow{\phi(\mu)} q[\phi]} \\
 \\
 \text{SumL)} \frac{p_1 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q} & \text{SumR)} \frac{p_2 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q} & \\
 \\
 \text{ParL)} \frac{p_1 \xrightarrow{\mu} q_1}{p_1 | p_2 \xrightarrow{\mu} q_1 | p_2} & \text{Com)} \frac{p_1 \xrightarrow{\lambda} q_1 \quad p_2 \xrightarrow{\bar{\lambda}} q_2}{p_1 | p_2 \xrightarrow{\tau} q_1 | q_2} & \text{ParR)} \frac{p_2 \xrightarrow{\mu} q_2}{p_1 | p_2 \xrightarrow{\mu} p_1 | q_2} \\
 \\
 \text{Rec)} \frac{p[\mathbf{rec} \ x. \ p / x] \xrightarrow{\mu} q}{\mathbf{rec} \ x. \ p \xrightarrow{\mu} q} & & 
 \end{array}$$

# CCS

## Encoding imperative languages

# Preliminaries: termination

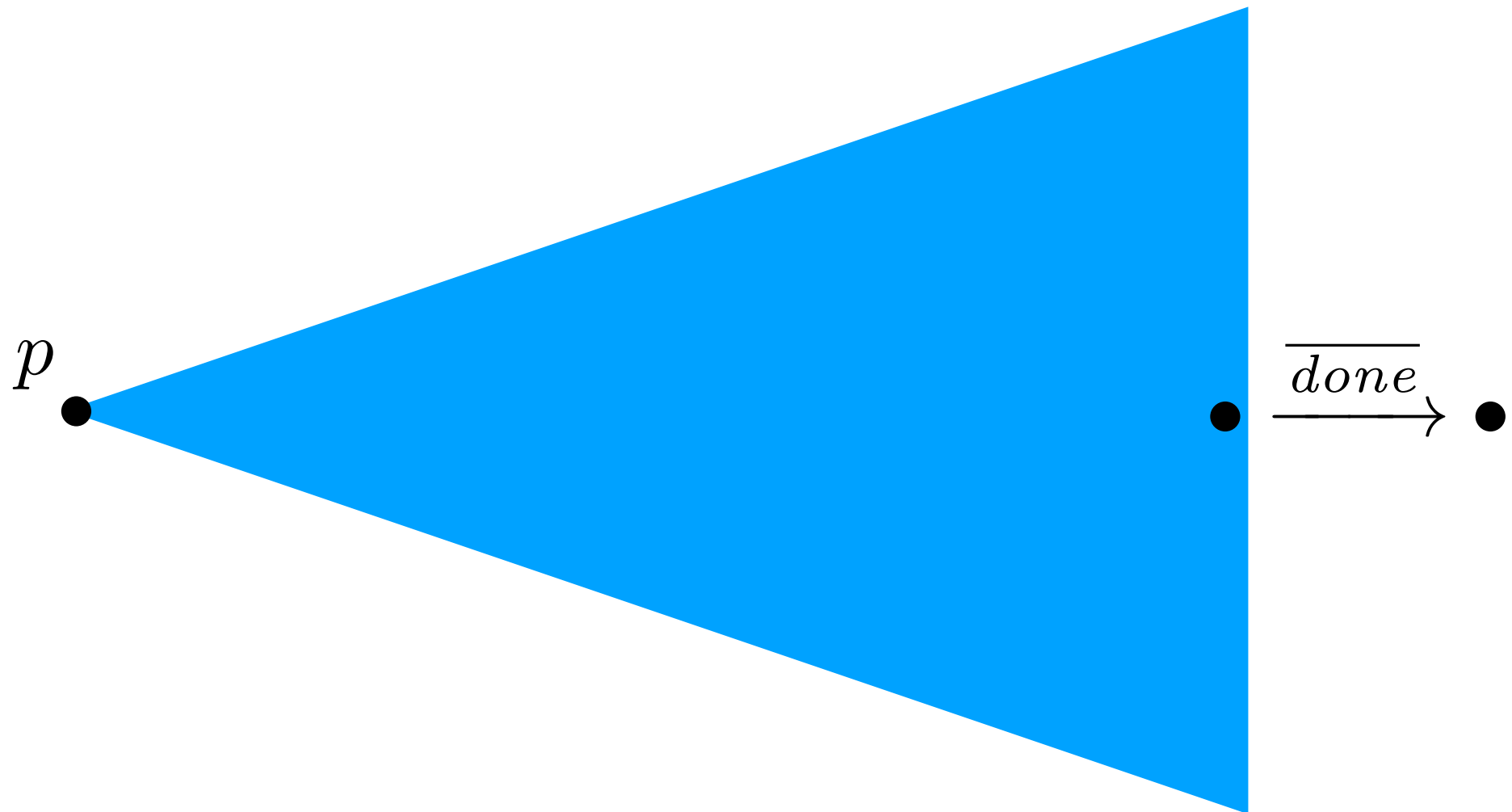
A dedicated channel *done*:

a message is sent when the current command terminates

$$\text{Done} \triangleq \overline{done}$$

$$\text{Done} \xrightarrow{\overline{done}} \mathbf{nil}$$

# Termination



# Skip

skip

does nothing and sends *done*

$\tau$ .Done

•  $\xrightarrow{\tau}$  Done  $\xrightarrow{\overline{done}}$  **nil**

# Variables

$x$  ranging over  $V = \{v_1, \dots, v_n\}$

a dedicated process for managing each variable

we can read its current value (channel  $xr_i$  )

we can write any value (channel  $xw_i$  )

$$\begin{aligned} XW &\triangleq \sum_{i=1}^n xw_i.X_i \\ &= xw_1.X_1 + \dots + xw_n.X_n \end{aligned}$$

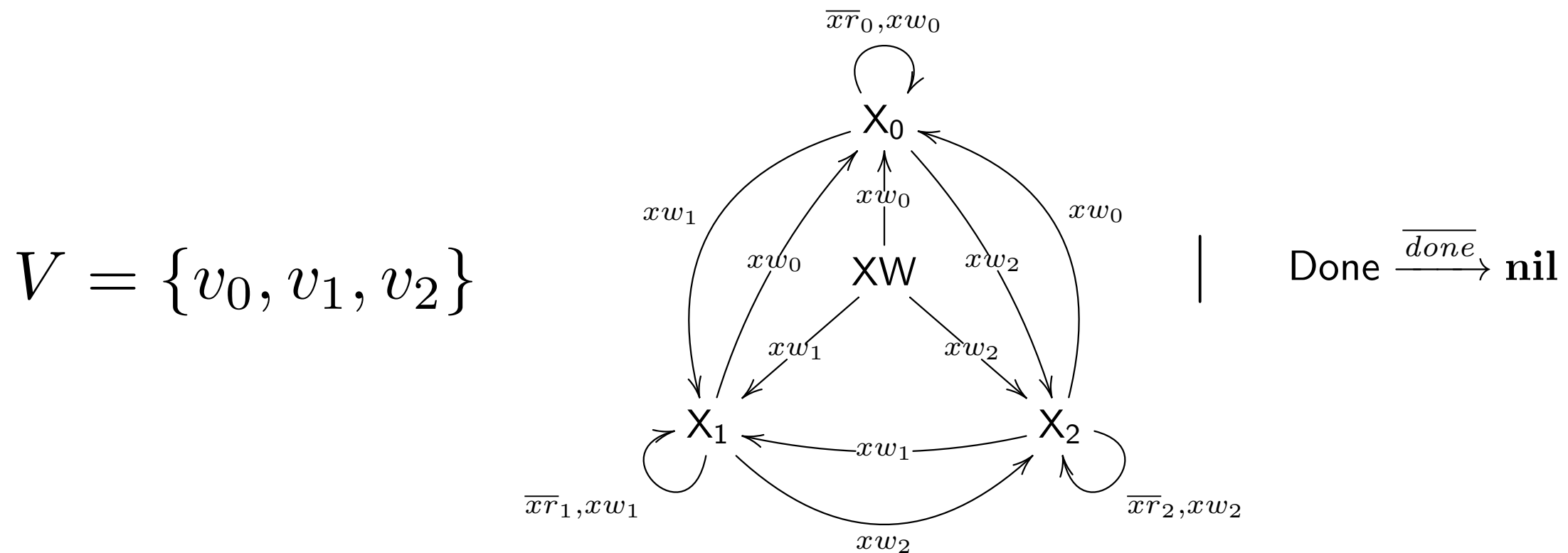
$$X_i \triangleq \overline{xr_i}.X_i + XW$$

# Variable declaration

$\text{var } x$

releases an uninitialised variable and terminates

$XW \mid \text{Done}$



# Assignment

$$x := v_i$$

sends a message to change the state of the variable  
and then terminates

$$\overline{xw_i}.Done$$

$$\bullet \xrightarrow{\overline{xw_i}} Done \xrightarrow{\overline{done}} \mathbf{nil}$$

# Sequential composition

$$c_1 ; c_2$$

suppose  $p_1$  models  $c_1$   
 $p_2$  models  $c_2$

$$p_1 | done.p_2$$

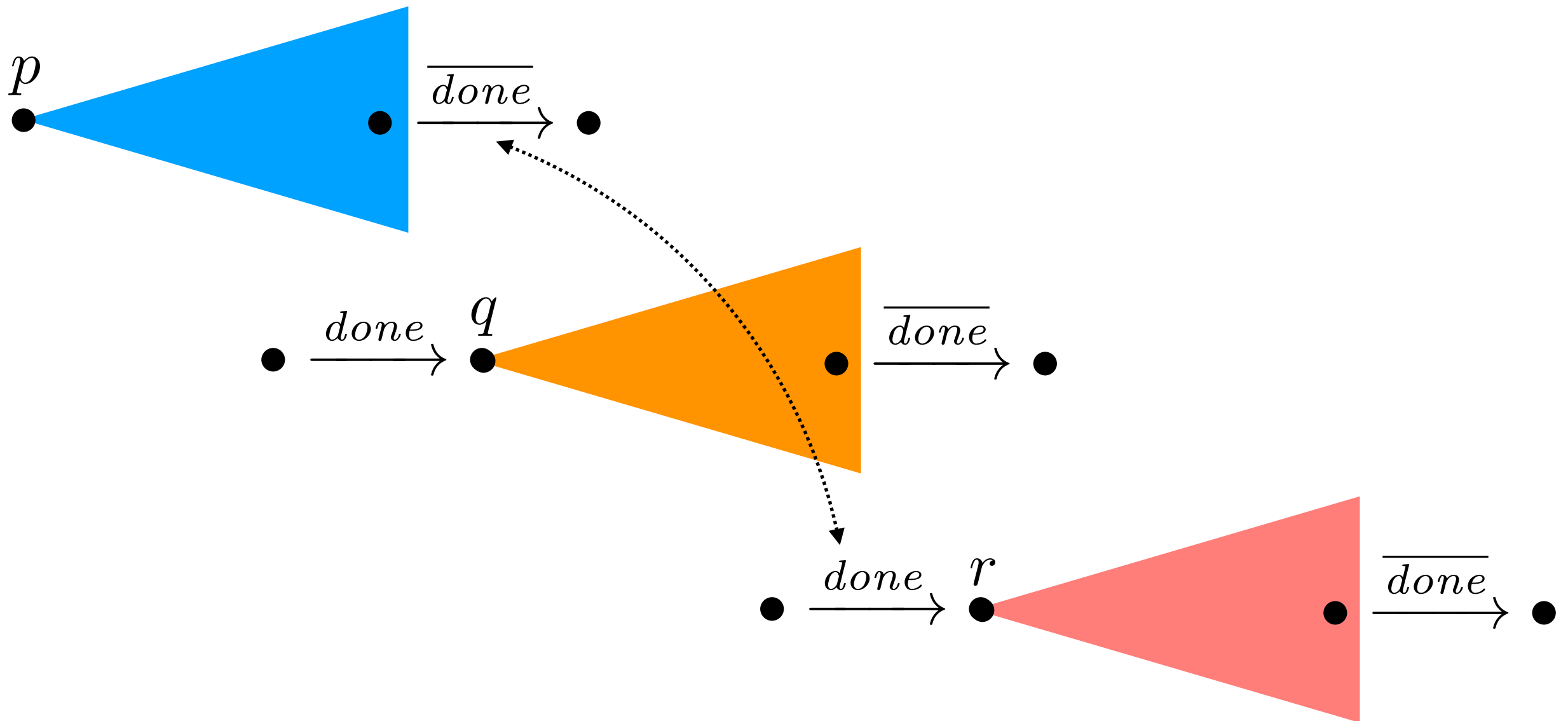
unfortunate choice: does not scale

$$c_1 ; c_2 ; c_3$$

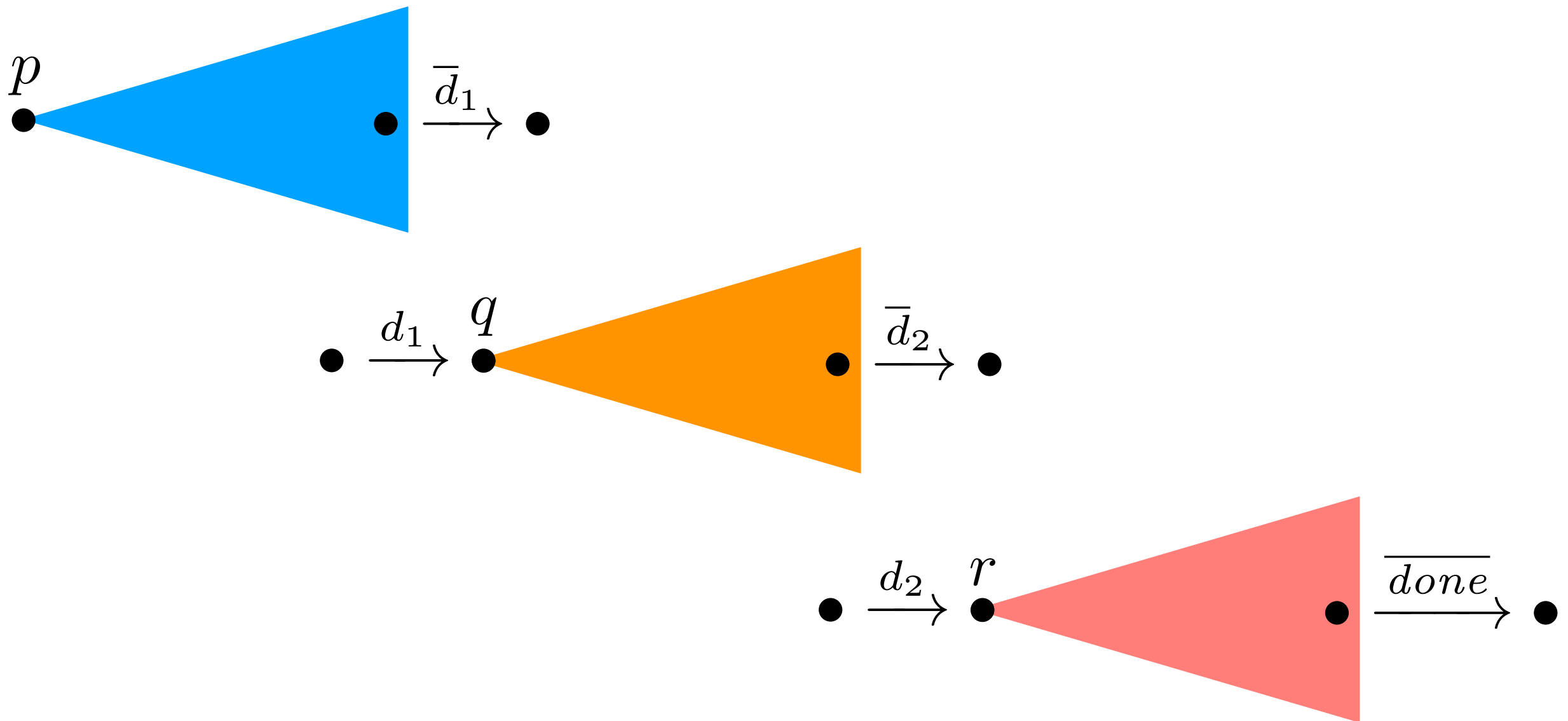
$$p_1 | done.p_2 | done.p_3$$

$p_3$  can start after  $p_1$

# Sequential?



# Sequential!



# Sequential composition

$$c_1 ; c_2$$

suppose  $p_1$  models  $c_1$   $\phi_d(done) = d$   
 $p_2$  models  $c_2$

$$p_1 \frown p_2 \triangleq (p_1[\phi_d] \mid d.p_2) \setminus d$$

now  $d$  is local to  $p_1$  and  $p_2$

$$( ( (p_1[\phi_{d_1}] \mid d_1.p_2) \setminus d_1) [\phi_{d_2}] \mid d_2.p_3) \setminus d_2$$

$$( ( (p_1[\phi_d] \mid d.p_2) \setminus d) [\phi_d] \mid d.p_3) \setminus d$$

$$(p_1 \frown p_2) \frown p_3$$

# Conditional

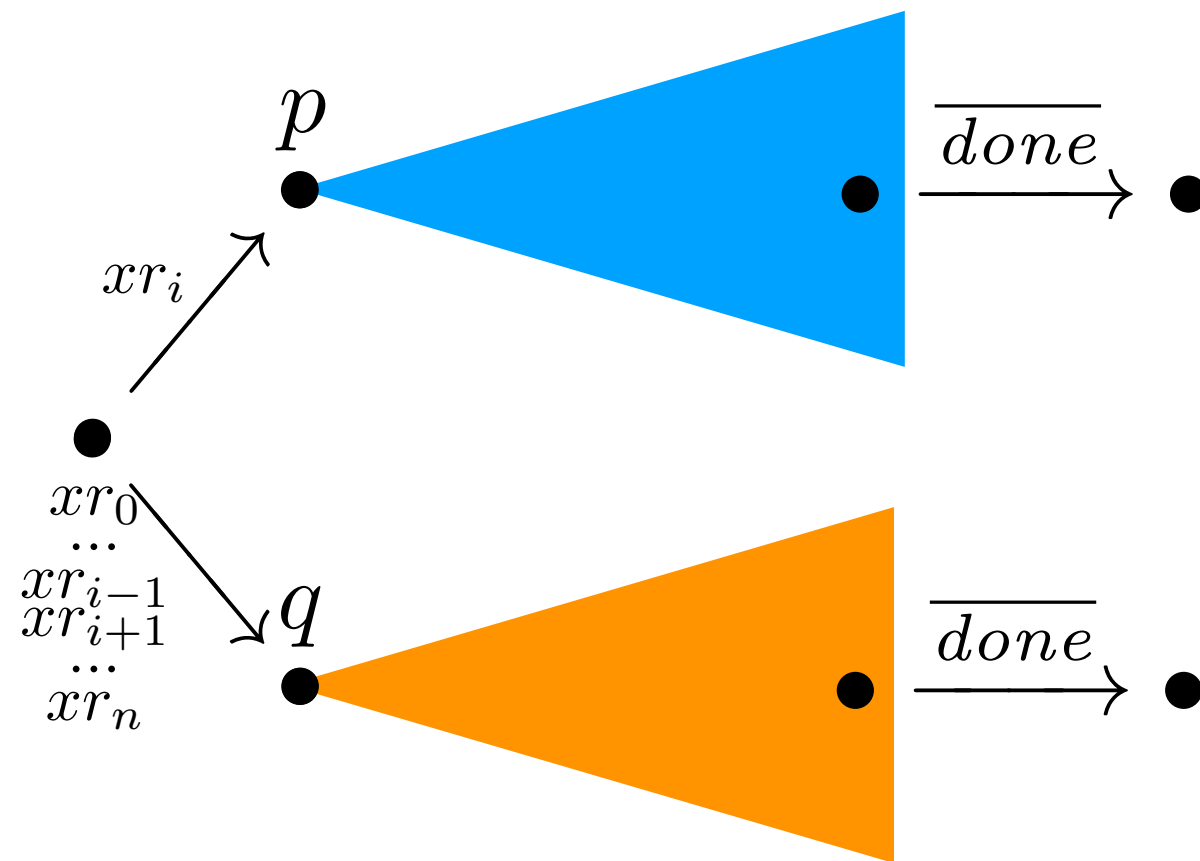
if  $x = v_i$  then  $c_1$  else  $c_2$

suppose  $p_1$  models  $c_1$   
 $p_2$  models  $c_2$

receives the state of the variable  
and then chooses accordingly

$$xr_i \cdot p_1 + \sum_{j \neq i} xr_j \cdot p_2$$

# Conditional



# Iteration

while  $x = v_i$  do  $c$

suppose  $p$  models  $c$

receives the state of the variable  
and then chooses accordingly, possibly recurring

$$\mathbf{rec} \ y. \ x r_i. (p[\phi_d] | d.y) \setminus d + \sum_{j \neq i} x r_j. \text{Done}$$

$$Y \triangleq x r_i. (p[\phi_d] | d.Y) \setminus d + \sum_{j \neq i} x r_j. \text{Done}$$

$$Y \triangleq x r_i. (p \frown Y) + \sum_{j \neq i} x r_j. \text{Done}$$

# Concurrency

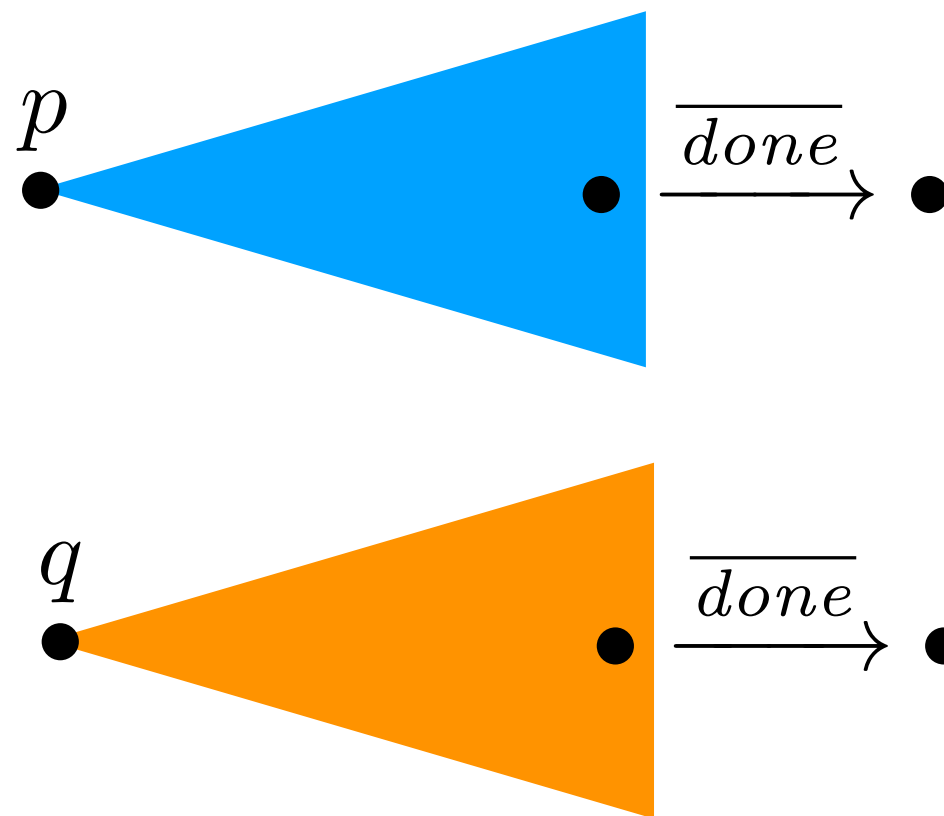
$$c_1 \mid c_2$$

suppose  $p_1$  models  $c_1$   
 $p_2$  models  $c_2$

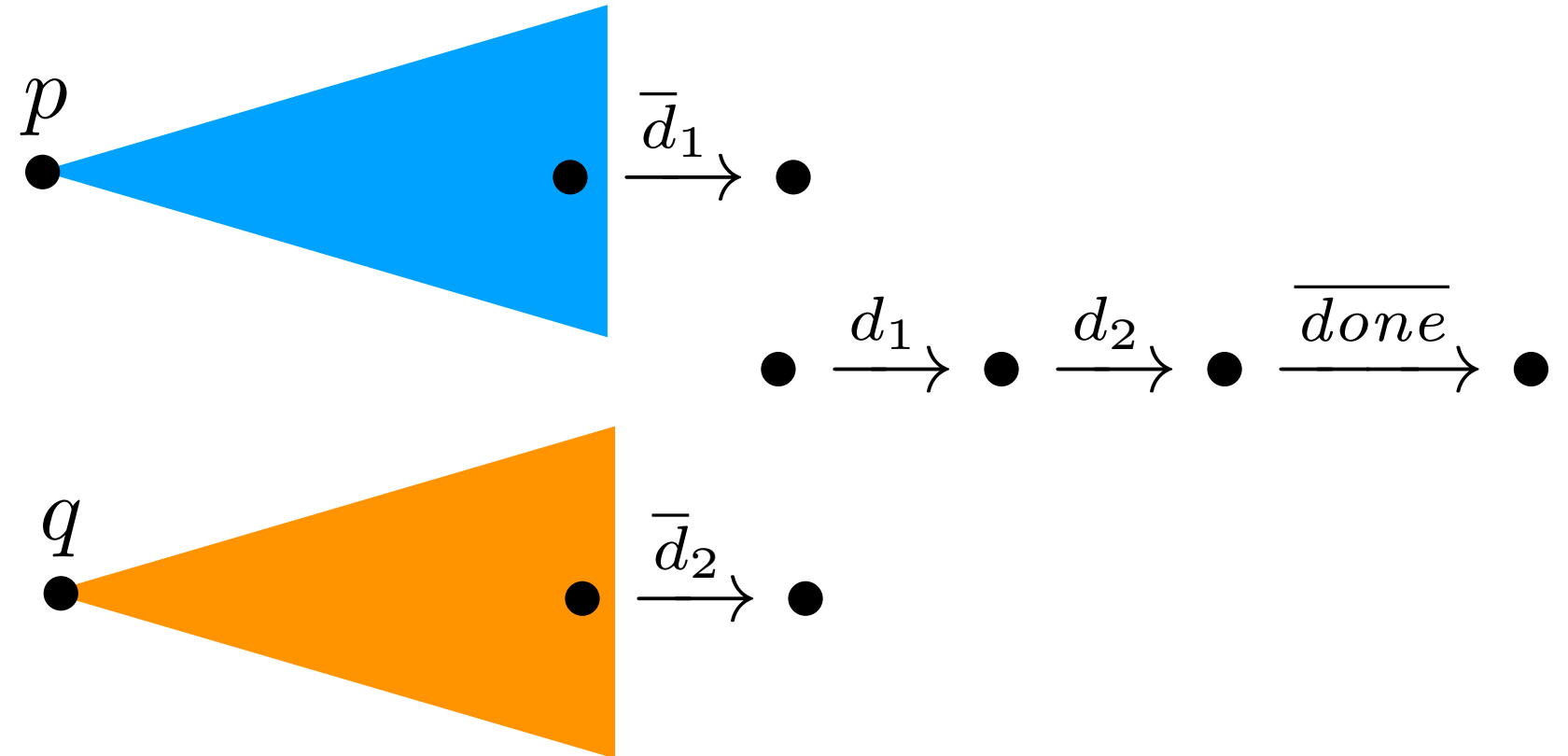
$$p_1 \mid p_2$$

unfortunate choice: *done* is possibly issued twice

# Concurrent termination



# Joint termination



# Concurrency

$$c_1 \mid c_2$$

suppose  $p_1$  models  $c_1$   
 $p_2$  models  $c_2$

$$(p_1[\phi_{d_1}] \mid p_2[\phi_{d_2}] \mid d_1.d_2.\text{Done}) \setminus d_1 \setminus d_2$$

the order is not important: both must be done to terminate

$$(p_1[\phi_d] \mid p_2[\phi_d] \mid d.d.\text{Done}) \setminus d$$

$$(p_1[\phi_d] \mid p_2[\phi_d] \mid d^2.\text{Done}) \setminus d$$

# Finally

all channels for communicating with variables  
must be restricted at the top  
to guarantee read/write requests are synchronised

$$p \setminus \{xw_1, xr_1, \dots\}$$

several optimisations are possible:  
action prefix instead of linking for sequential composition  
allows more expressive guards  
remove silent transitions  
introduce constants for loops  
implement expressions  
...

# Example: optimisation

$$x := 1; y := 2$$

$$\overline{xw_1.done} \quad \frown \quad \overline{yw_2.done}$$

$$( (\overline{xw_1.done})[\phi_d] \mid d.\overline{yw_2.done} ) \backslash d$$

$$\overline{xw_1.yw_2.done}$$

# Example: optimisation

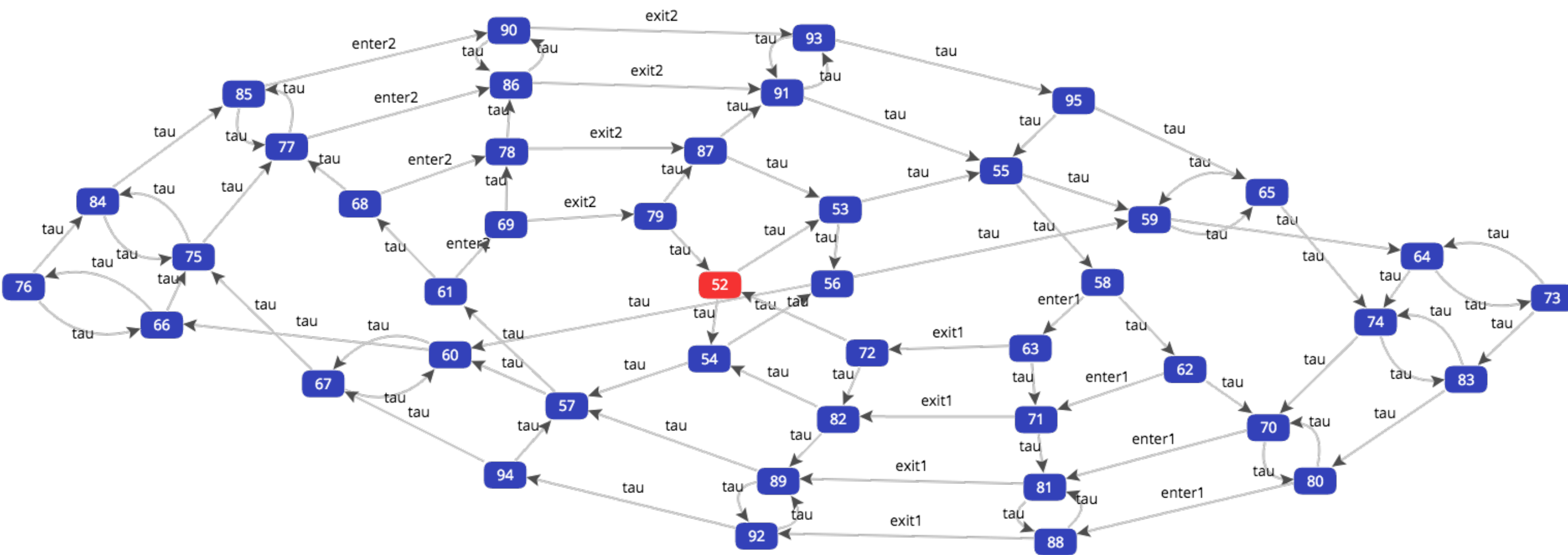
if  $b(x)$  then  $c_1$  else  $c_2$

$$\sum_{b(v_i)} x r_i \cdot p_1 + \sum_{\neg b(v_j)} x r_j \cdot p_2$$

# CCS

## Playing with CAAL

# Concurrency Workbench, Aalborg Edition



<http://caal.cs.aau.dk/>

# CAAL

CAAL is a web-based tool for  
modeling, visualization and verification  
of concurrent processes in CCS (and its timed extension)

developed for educational purposes

**Edit** (recursively defined) processes

**Explore** their LTS (collapse states up to equivalences)

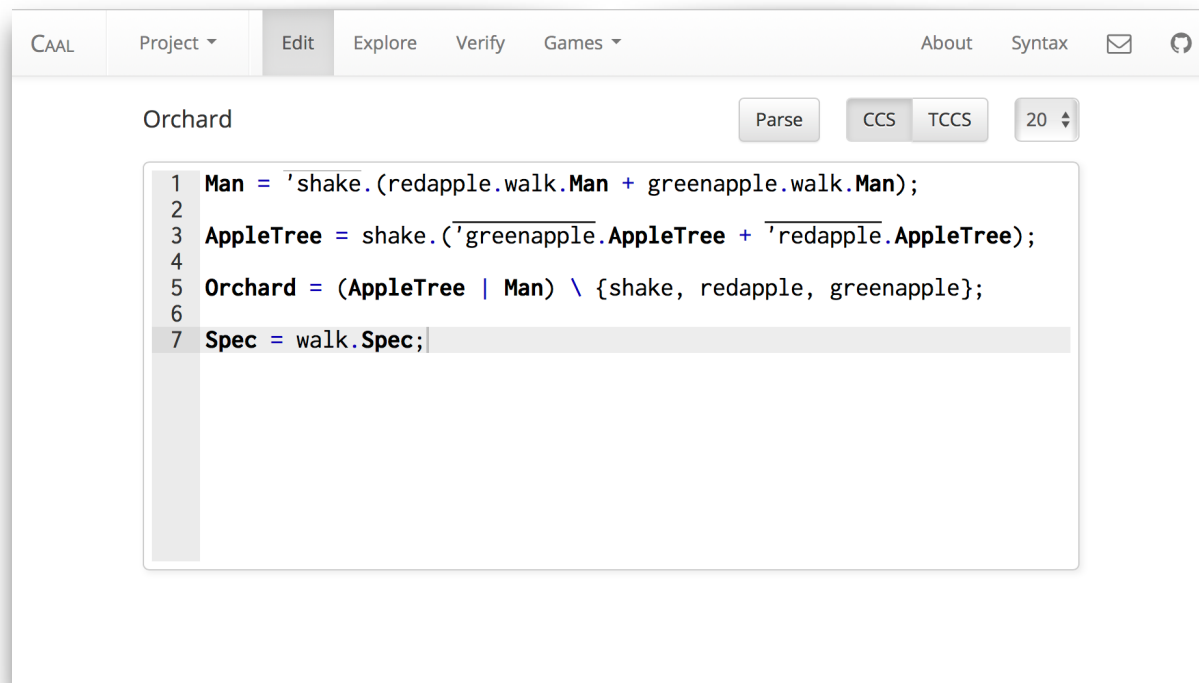
**Verify** HML formulas satisfaction (model checking)

**Verify** equivalences between processes  
(generate distinguishing formulas)

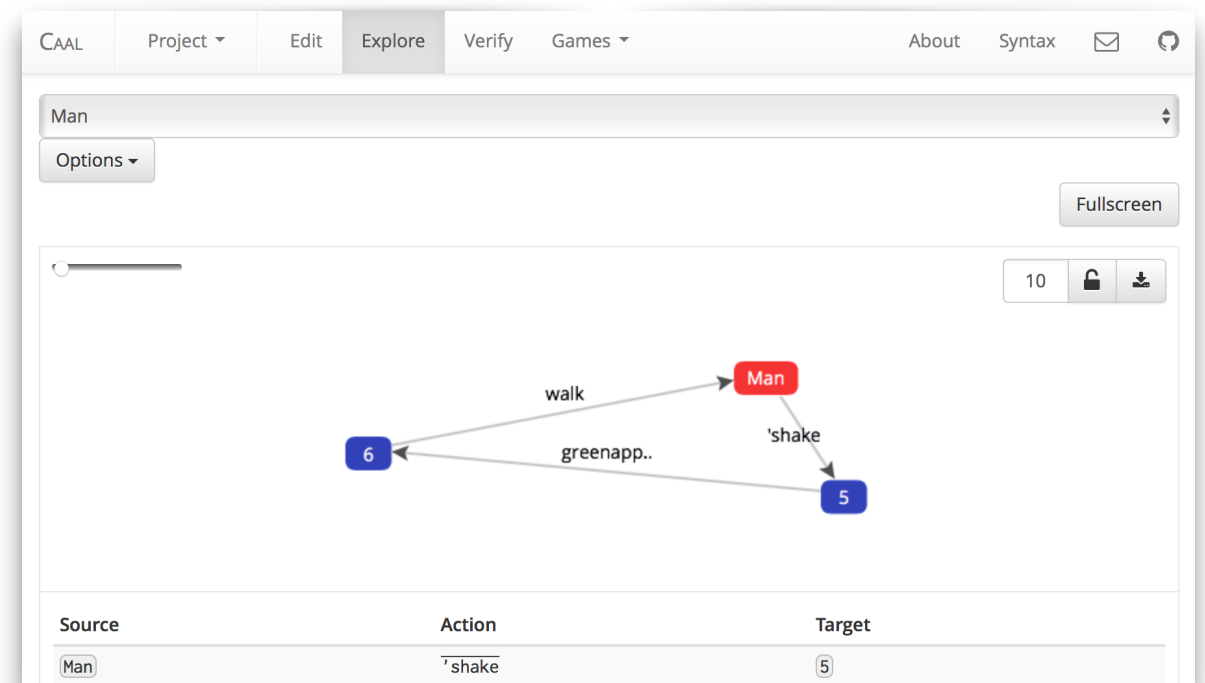
play bisimulation **Game**

# CAAL

## Edit processes



## Explore LTS

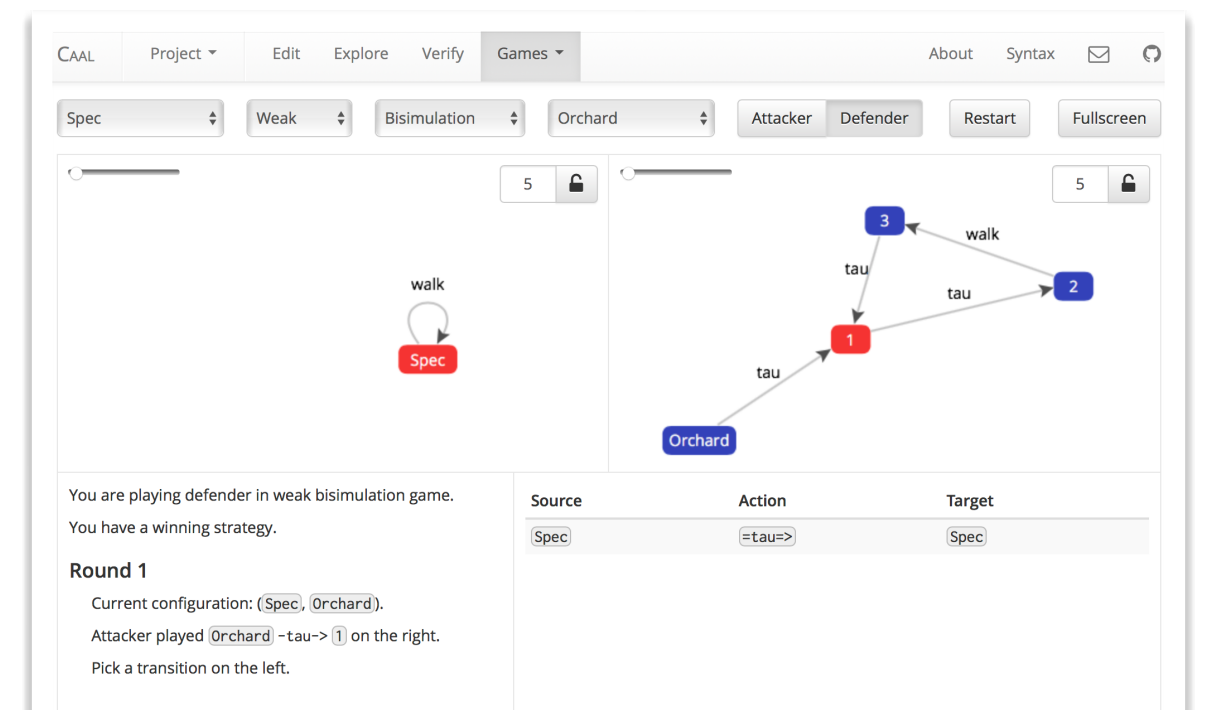


## Verify HML & Equivalences

The screenshot shows the CAAL Verify view with the 'Verify' tab selected. The table displays the results of verification for various properties. The table has columns for Status, Time, Property, Verify, Edit, Delete, and Options.

| Status | Time  | Property                                    | Verify | Edit | Delete | Options |
|--------|-------|---------------------------------------------|--------|------|--------|---------|
| ✗      | 26 ms | Orchard ~ Spec                              | ▶      | ✎    | 🗑      | ☰       |
| ✓      | 25 ms | Orchard $\models$ $\langle \tau \rangle tt$ | ▶      | ✎    | 🗑      | ☰       |
| ✗      | 25 ms | Spec $\models$ $\langle \tau \rangle tt$    | ▶      | ✎    | 🗑      | ☰       |
| ✓      | 25 ms | Orchard $\approx$ Spec                      | ▶      | ✎    | 🗑      | ☰       |

## play bisimulation Game



# CCS syntax in CAAL

$\text{nil}$       0

$a.P$      $\bar{a}.P$      $\tau.P$        $a.P$     'a.P    tau.P

$P + Q$        $P + Q$

$P \mid Q$        $P \mid Q$

$P \setminus \{a_1, \dots, a_n\}$        $P \setminus \{a_1, \dots, a_n\}$

$P[b_1/a_1, \dots, b_n/a_n]$        $P[b_1/a_1, \dots, b_n/a_n]$

$A \triangleq P$        $A = P ;$

# HML syntax in CAAL

tt tt

ff ff

$F \wedge G$  F and G

$F \vee G$  F or G

$\Diamond_{\{a1, \dots, an\}} F$   $\langle a1, \dots, an \rangle F$

$\Box_{\{a1, \dots, an\}} F$   $[a1, \dots, an] F$

# CAAL

## mutual exclusion protocols analysis

# Peterson's mex algorithm

```
% Two processes P1, P2
% Two boolean variables b1, b2 (both initially false)
% when Pi wants to enter the critical section, then it sets bi to true
% An integer variable k, taking values in {1,2}
% (initial value is arbitrary)
% the process Pk has priority over the other process
%
% Process P1 in pseudocode
while (true) {
    ...                                % non critical section
    b1 = true ;                        % P1 wants to enter the critical section
    k = 2 ;                            % P1 gives priority to the other process
    while (b2 && k==2) skip ;          % P1 waits its turn
    ...                                % P1 enters the critical section
    b1 = false                         % P1 leaves the critical section
}

% Process P2 is analogous to P1
```

# Peterson's mex in CCS

$$B1W \triangleq b1wf.B1f + b1wt.B1t$$

$$B1f \triangleq \overline{b1rf}.B1f + B1W$$

$$B1t \triangleq \overline{b1rt}.B1t + B1W$$

$$KW \triangleq kw1.K1 + kw2.K2$$

$$K1 \triangleq \overline{kr1}.K1 + KW$$

$$K2 \triangleq \overline{kr2}.K2 + KW$$

% Process P1 in pseudocode

```
while (true) {
  ...
  b1 = true ; % wants to enter
  k = 2 ; % gives priority to P2
  while (b2 && k==2) skip ; % waits
  ... % enters critical section
  b1 = false % leaves
}
```

% Process P2 is analogous to P1

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b \quad \begin{array}{l} \text{leave cycle} \\ \text{loop} \end{array} \quad + b2rt.(kr1.P1b + kr2.P1a)$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

just something to observe

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

# Peterson's mex in CCS

$$B1W \triangleq b1wf.B1f + b1wt.B1t$$

$$B1f \triangleq \overline{b1rf}.B1f + B1W$$

$$B1t \triangleq \overline{b1rt}.B1t + B1W$$

$$KW \triangleq kw1.K1 + kw2.K2$$

$$K1 \triangleq \overline{kr1}.K1 + KW$$

$$K2 \triangleq \overline{kr2}.K2 + KW$$

% Process P1 in pseudocode

while (true) {

...

b1 = true ; % wants to enter

k = 2 ; % gives priority to P2

while (b2 && k==2) skip ; % waits

... % enters critical section

b1 = false % leaves

}

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + \overline{kr1}.P1b$$

no busy wait

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

% Process P2 is analogous to P1

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

# Peterson's mex in CCS

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + kr1.P1b$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

a formula for mutual exclusion?

any label

$$F \triangleq [exit_1]\mathbf{ff} \vee [exit_2]\mathbf{ff}$$

$$F \wedge [-](\dots)$$

$$F \wedge [-](F \wedge [-](\dots))$$

$$MEX_{(max)} \triangleq F \wedge [-]MEX$$

$$F \wedge [-](F \wedge [-](F \wedge [-](\dots)))$$

a recursively defined formula!

# Peterson's mex in CCS

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + kr1.P1b$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

has P1 the possibility to enter?

$$G \triangleq \langle enter_1 \rangle \mathbf{tt}$$

$$G \vee \langle - \rangle (\dots)$$

$$G \vee \langle - \rangle (G \vee \langle - \rangle (\dots))$$

$$EN_{(\min)} \triangleq G \vee \langle - \rangle EN$$

$$G \vee \langle - \rangle (G \vee \langle - \rangle (G \vee \langle - \rangle (\dots)))$$

a recursively defined formula!

# Peterson's mex in CCS

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + kr1.P1b$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

has P1 the possibility to enter from any reachable state ?

$$EN \triangleq_{(min)} G \vee \langle - \rangle EN \quad EN \wedge [-](\dots)$$

$$EN \wedge [-](EN \wedge [-](\dots))$$

$$AEN \triangleq_{(max)} EN \wedge [-]AEN \quad EN \wedge [-](EN \wedge [-](EN \wedge [-](\dots)))$$

a recursively defined formula!

# Peterson's mex in CCS

$$P1 \triangleq \overline{b1wt}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + kr1.P1b$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

deadlock freedom?

$$H \triangleq \langle - \rangle \mathbf{tt}$$

$$H \wedge [-](\dots)$$

$$H \wedge [-](H \wedge [-](\dots))$$

$$DF_{(\max)} \triangleq H \wedge [-]DF$$

$$H \wedge [-](H \wedge [-](H \wedge [-](\dots)))$$

a recursively defined formula!

# Peterson's mex in CCS

$$P1 \triangleq \overline{b1wt}.\boxed{req_1}.\overline{kw2}.P1a$$

$$P1a \triangleq b2rf.P1b + kr1.P1b$$

$$P1b \triangleq enter1.exit1.\overline{b1wf}.P1$$

$$SP \triangleq (B1f|B2f|KW|P1|P2) \setminus \{kw1, kr1, kw2, kr2, \dots\}$$

does P1 access every time it issues a request?

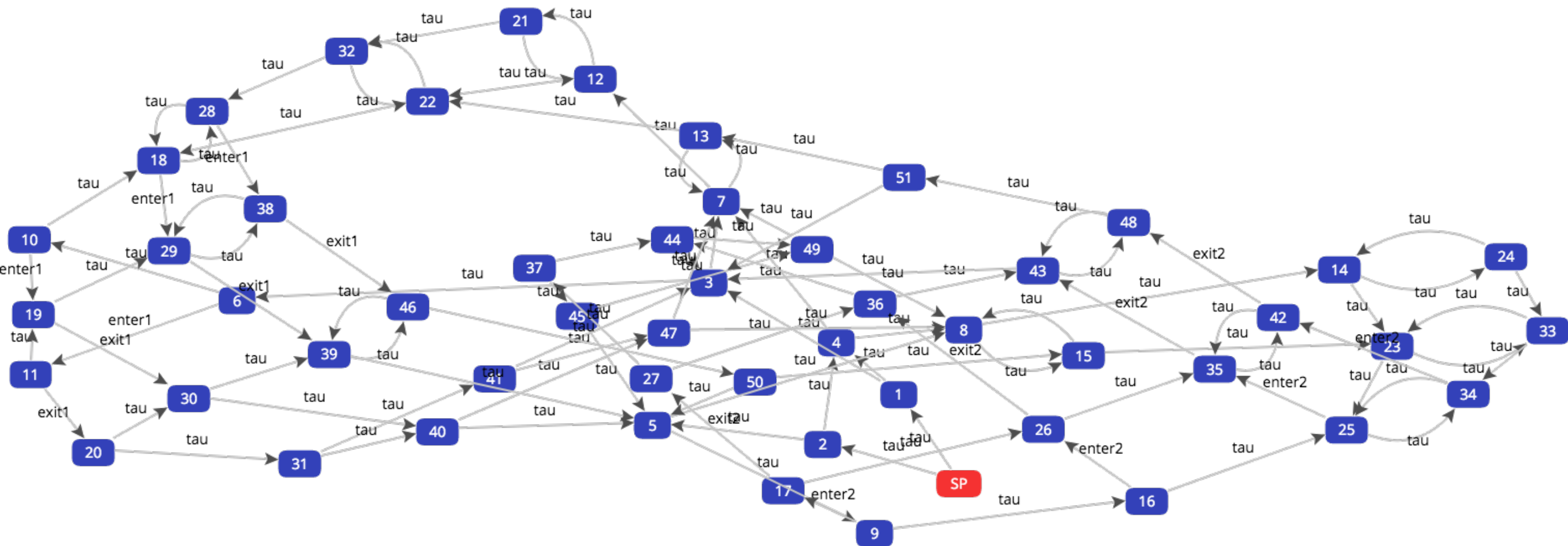
$$A_{(min)} \triangleq \langle exit_1 \rangle \mathbf{tt} \vee [-]A$$

$$REQ_{(max)} \triangleq [req_1]A \wedge [-]REQ$$

a recursively defined formula!

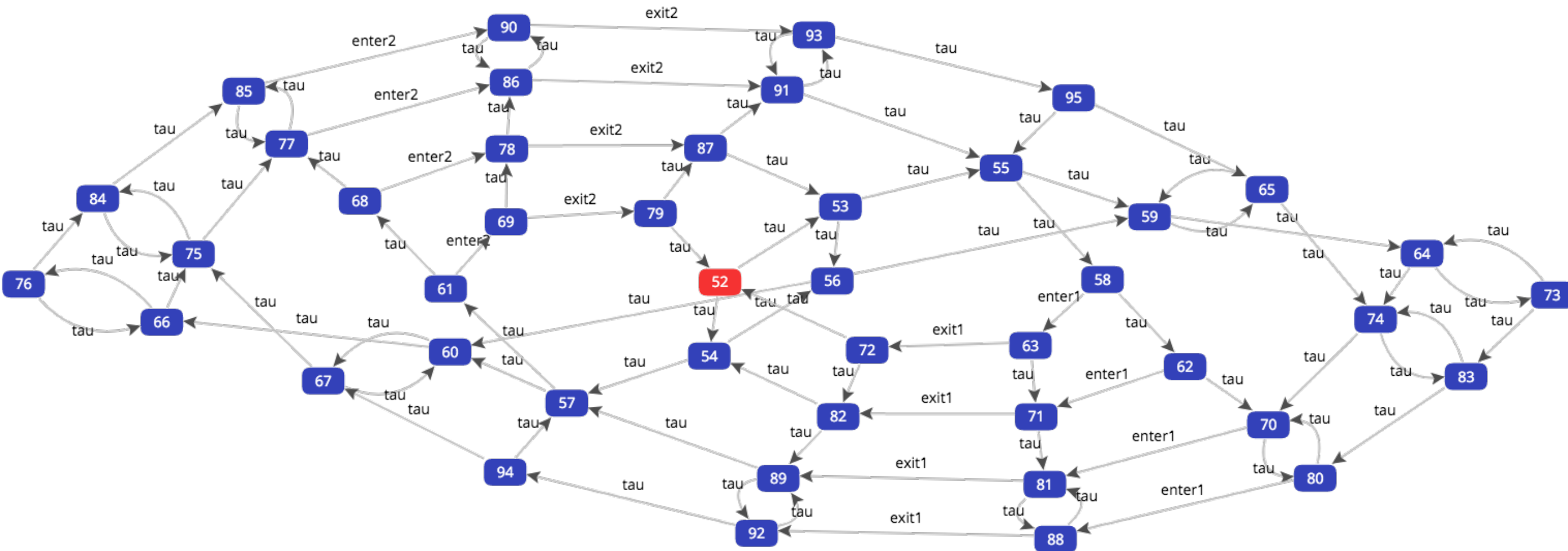
# Peterson's mex in CAAL

# LTS



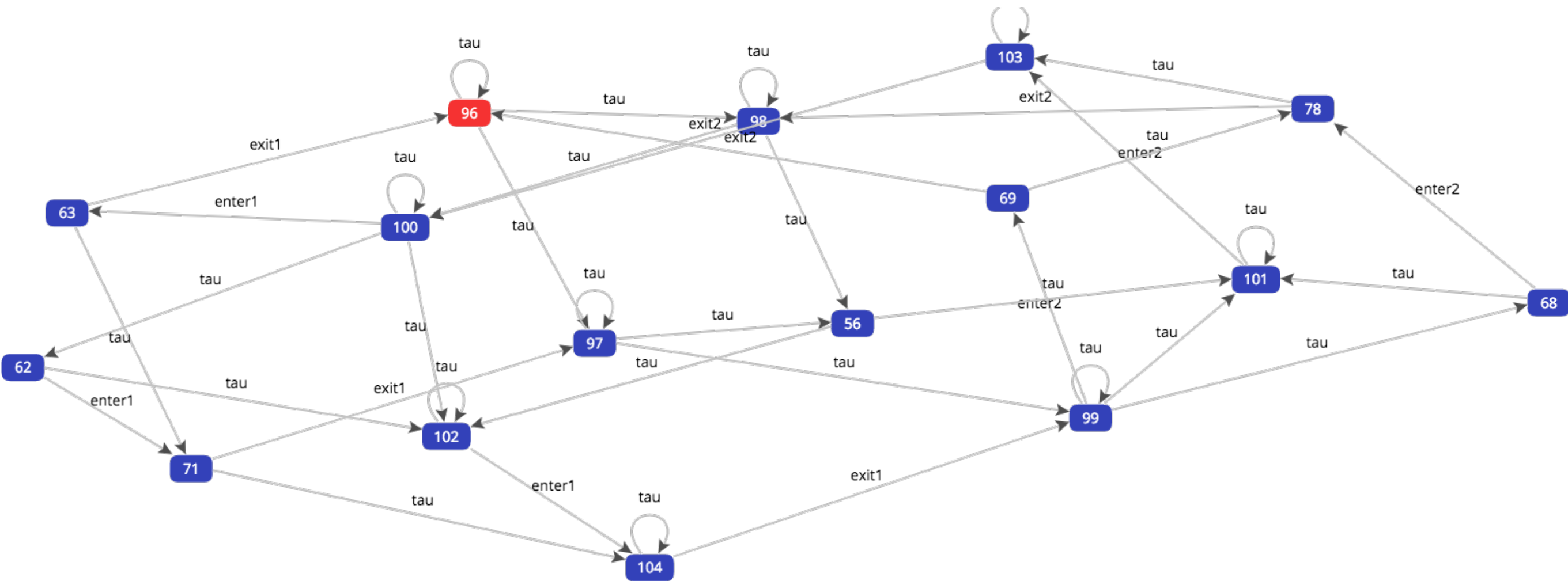
# Peterson's mex in CAAL

LTS up to strong bisimilarity



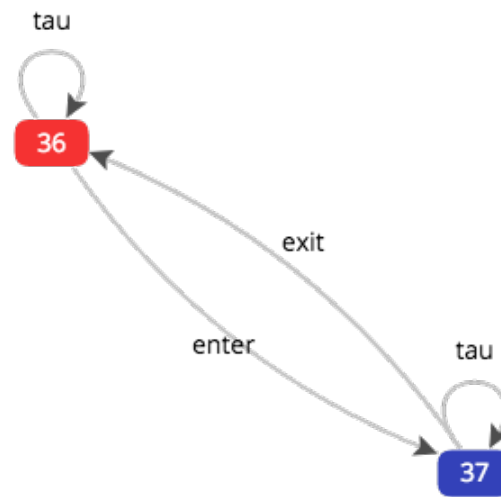
# Peterson's mex in CAAL

LTS up to weak bisimilarity



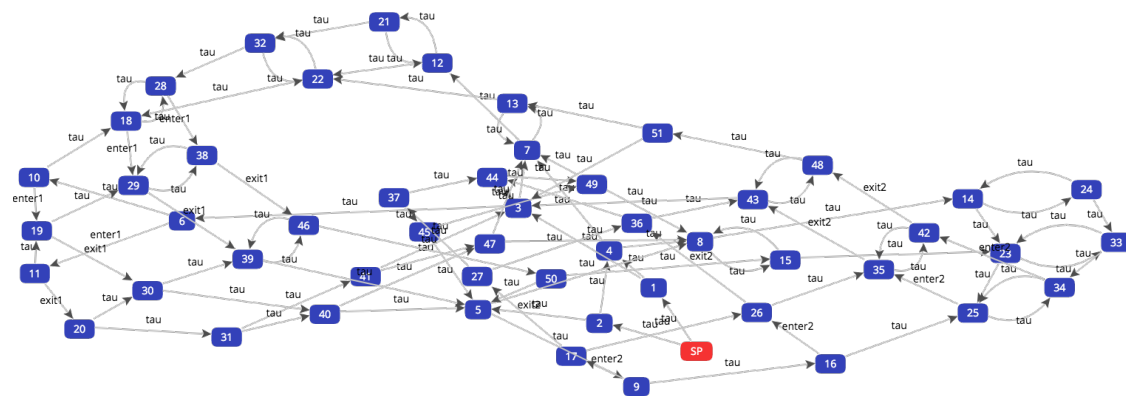
# Peterson's mex in CAAL

LTS up to weak bisimilarity, after renaming  
*enter1/2* to *enter* and *exit1/2* to *exit*

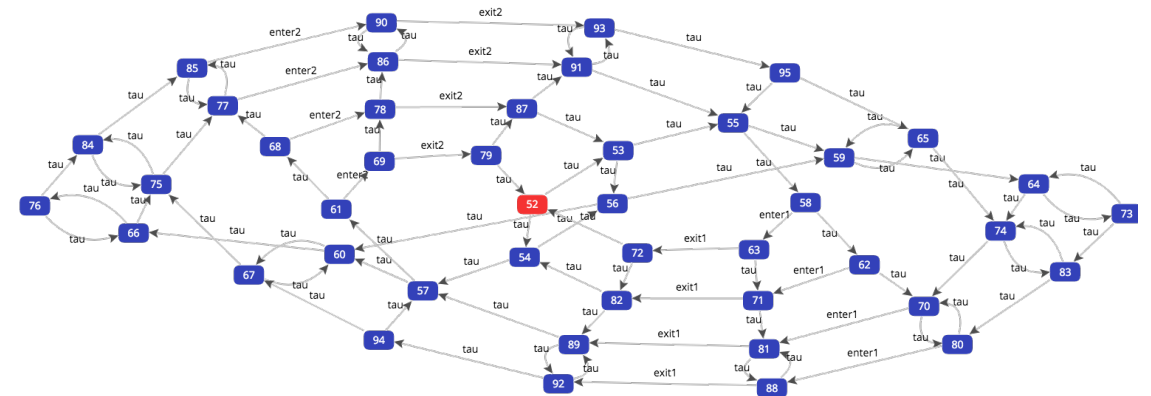


# Peterson's mex in CAAL

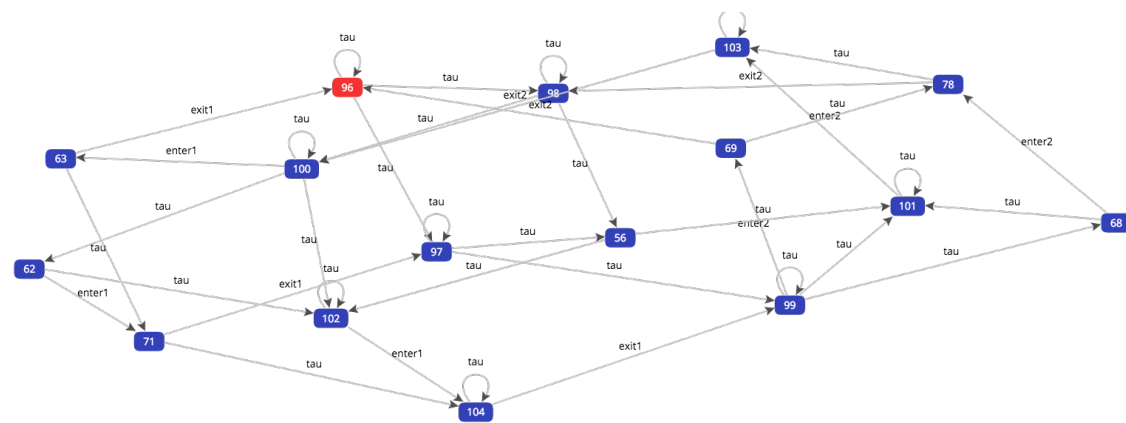
LTS



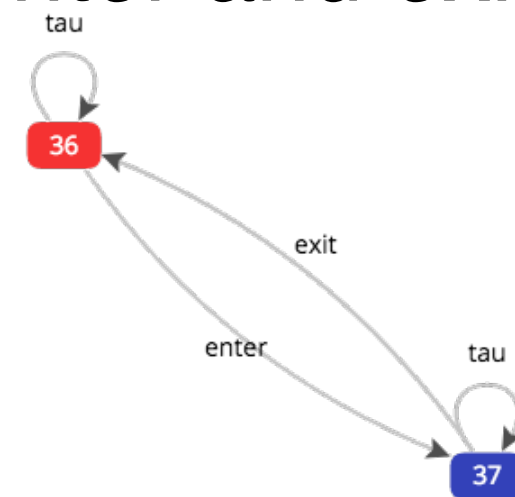
LTS up to strong bisimilarity



LTS up to weak bisimilarity



LTS up to weak bisimilarity,  
*enter* and *exit*



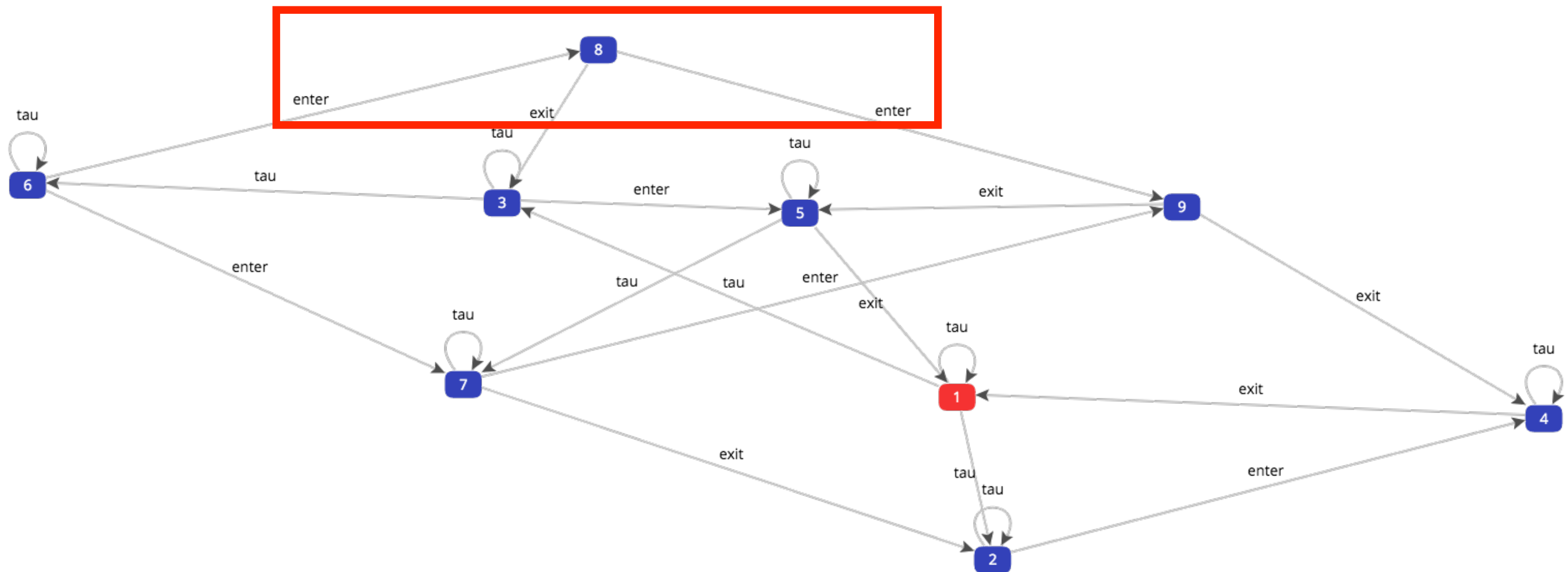
# Hyman's mex algorithm

```
% Two processes H1, H2
% Two boolean variables b1, b2 (both initially false)
% when Hi wants to enter the critical section, then it sets bi to true
% An integer variable k, taking values in {1,2}
% (initial value is arbitrary)
% the process Hk has priority over the other process
%
% Process H1 in pseudocode
while (true) {
    ...                                % non critical section
    b1 = true ;                        % H1 wants to enter the critical section
    while (k==2) {                     % while H2 has priority
        while (b2) skip ;             % H1 waits
        k = 1;                        % H1 sets priority to itself
    }
    ...                                % H1 enters the critical section
    b1 = false                         % H1 leaves the critical section
}

% Process H2 is analogous to H1
```

# Peterson's mex in CAAL

LTS up to weak bisimilarity, after renaming  
*enter1/2* to *enter* and *exit1/2* to *exit*



# 50 prisoners puzzle

50 prisoners kept in separate cells got a chance to be released:  
from time to time one of them will be carried in a special room and then back to cell.  
(in no particular order, possibly multiple times consecutively, but fairly to avoid infinite wait)

The room is completely empty except for a switch that can turn the light on or off  
(the light is not visible from outside).

At any time, if any of them claims rightfully that all the prisoners have already entered the room at least once, then all prisoners will be released  
(but if it proves wrong, then the chance ends and they will never be released).

The prisoners have the possibility to discuss in advance some protocol to follow  
(not all prisoner must behave in the same way).

Can you find a winning strategy for the prisoners?  
Can you formalise it in CCS (for 2 to 4 prisoners)?

- Easy case: it is known that the light in the room is initially off
- Hard case: the initial state of the light in the room is not known.

