PSC 2020/21 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/

Exercises #6

Erlang

[**Ex. 1**] Write a server in erlang to convert temperatures from Celsius degrees to Fahrenheit degrees and vice versa, using the formula $F = 1.8C + 32$. The server receives requests of the form $(Pid, \texttt{cs}, C)$ or $(Pid, \texttt{ft}, F)$ and replies to *Pid* by sending messages in analogous format. The server can be stopped by sending the message `stop`. All the other messages are ignored. Spawn a copy of the server, send it some temperatures to convert, check out the results and stop the server.

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive







    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} ->




    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
      {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                    convert();




    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
      {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                    convert();
      {Pid,ft,F} ->




    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();


    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;

    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;
     _ ->
    end.
```

Ex. 1, temp converter

```
-module(ex1).
-export([convert/0]).

convert() ->
    receive
     {Pid,cs,C} -> Pid ! {self(),ft,(1.8 * C) + 32},
                   convert();
     {Pid,ft,F} -> Pid ! {self(),cs,(F - 32) / 1.8},
                   convert();
     stop -> true;
     _ -> convert()
    end.
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4> receive
4>     {Conv,ft,F} -> io:format("23 celsius = ~p fahrenheit~n",[F])
4> end.
23 celsius = 73.4 fahrenheit
ok
5>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4> receive
4>     {Conv,ft,F} -> io:format("23 celsius = ~p fahrenheit~n",[F])
4> end.
23 celsius = 73.4 fahrenheit
ok
5> Conv ! {self(),ft,74}.
{<0.77.0>,ft,74}
6>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4> receive
4>     {Conv,ft,F} -> io:format("23 celsius = ~p fahrenheit~n",[F])
4> end.
23 celsius = 73.4 fahrenheit
ok
5> Conv ! {self(),ft,74}.
{<0.77.0>,ft,74}
6> receive
6>     {Conv,cs,C} -> io:format("74 fahrenheit = ~p celsius~n",[C])
6> end.
74 fahrenheit = 23.333333333333332 celsius
ok
7>
```

Ex. 1, temp converter

```
Eshell V10.2.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> Conv = spawn(ex1,convert,[]).
<0.84.0>
3> Conv ! {self(),cs,23}.
{<0.77.0>,cs,23}
4> receive
4>     {Conv,ft,F} -> io:format("23 celsius = ~p fahrenheit~n",[F])
4> end.
23 celsius = 73.4 fahrenheit
ok
5> Conv ! {self(),ft,74}.
{<0.77.0>,ft,74}
6> receive
6>     {Conv,cs,C} -> io:format("74 fahrenheit = ~p celsius~n",[C])
6> end.
74 fahrenheit = 23.333333333333332 celsius
ok
7> Conv ! stop.
stop
8>
```

[**Ex. 2**] Write an erlang function `copy` that receives an integer n and if n is positive it prints n copies of n (one per line). Write an erlang function that receives a list of integers and spawn an instance of `copy` for each integer in the list.

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) ->
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) -> [                    || N <- L ].
```

Ex. 2, copy

```
-module(ex2).
-export([copy/1,listCopy/1]).

copy(N) when N > 0 -> copy(N,N);
copy(_) -> true.

copy(N,M) when N > 0 -> io:format("~p~n",[M]),
                        copy(N-1,M);
copy(_,_) -> true.

listCopy(L) -> [ spawn(ex2,copy,[N]) || N <- L ].
```

Ex. 2, copy

```
Eshell V10.2.1  (abort with ^G)
1> c(ex2).
{ok,ex2}
2>
```

Ex. 2, copy

```
Eshell V10.2.1  (abort with ^G)
1> c(ex2).
{ok,ex2}
2> ex2:listCopy(lists:seq(1,5)).
1
2
3
4
5
[<0.84.0>,<0.85.0>,<0.86.0>,<0.87.0>,<0.88.0>]
2
3
4
5
3
4
5
4
5
5
3>
```

[**Ex. 3**] Write an erlang function `view` that displays the content of the mailbox but makes all messages remain available in the mailbox afterwards.

Ex. 3, view

```
-module(ex3).
-export([view/0          ]).






view( ) ->
    receive
        Any -> io:format("view ~p~n",[Any]),
               view(       )
    after 0 -> true
    end.
```

Ex. 3, view

```
-module(ex3).
-export([view/0        ]).




view(L) ->
    receive
        Any -> io:format(“view ~p~n",[Any]),
               view([Any|L])
    after 0 -> true
    end.
```

Ex. 3, view

```
-module(ex3).
-export([view/0        ]).




view() -> view([]);
view(L) ->
    receive
        Any -> io:format("view ~p~n",[Any]),
               view([Any|L])
    after 0 -> true
    end.
```

Ex. 3, view

```
-module(ex3).
-export([view/0        ]).




view() -> view([]);
view(L) ->
    receive
        Any -> io:format("view ~p~n",[Any]),
               view([Any|L])
    after 0 -> send(L)
    end.
```

Ex. 3, view

```
-module(ex3).
-export([view/0,send/1]).

send([]) -> true;
send([A|L]) -> send(L),
               self() ! A.

view() -> view([]);
view(L) ->
    receive
        Any -> io:format("view ~p~n",[Any]),
               view([Any|L])
    after 0 -> send(L)
    end.
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2>
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2> ex3:send([3,2,1]).
3
3>
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2> ex3:send([3,2,1]).
3
3> ex3:view().
view 1
view 2
view 3
3
4>
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2> ex3:send([3,2,1]).
3
3> ex3:view().
view 1
view 2
view 3
3
4> ex3:view().
view 1
view 2
view 3
3
5>
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2> ex3:send([3,2,1]).
3
3> ex3:view().
view 1
view 2
view 3
3
4> ex3:view().
view 1
view 2
view 3
3
5> flush().
Shell got 1
Shell got 2
Shell got 3
ok
6>
```

Ex. 3, view

```
Eshell V10.2.1  (abort with ^G)
1> c(ex3).
{ok,exercise}
2> ex3:send([3,2,1]).
3
3> ex3:view().
view 1
view 2
view 3
3
4> ex3:view().
view 1
view 2
view 3
3
5> flush().
Shell got 1
Shell got 2
Shell got 3
ok
6> ex3:view().
true
```

CCS

[**Ex. 4**] Define a CCS process B^n_k that represents an in/out buffer with capacity n of which k positions are taken. Show that B^n_0 is strongly bisimilar to n copies of B^1_0 that run in parallel.

Ex. 4, buffers

$$B^n_0 \triangleq in.B^n_1$$

$$B^n_k \triangleq in.B^n_{k+1} + \overline{out}.B^n_{k-1} \text{ with } 0 < k < n$$

$$B^n_n \triangleq \overline{out}.B^n_{n-1}$$

we want to prove $B^n_0 \simeq \underbrace{B^1_0|\cdots|B^1_0}_{n}$

$$\mathbf{R} \triangleq \{(B^n_k, B^1_{k_1}|\cdots|B^1_{k_n}) \mid \forall i.\ k_i \in \{0,1\} \wedge \sum_{i=1}^{n} k_i = k\}$$

we prove that **R** is a strong bisimulation

Ex. 4, buffers

$$\begin{array}{ccc} B_0^n & \mathbf{R} & B_0^1|\cdots|B_0^1 \\ \Big\downarrow in & & \Big\downarrow in \\ B_1^n & \mathbf{R} & B_1^1|\cdots|B_0^1 \end{array}$$

$$\begin{array}{ccc} B_0^n & \mathbf{R} & B_0^1|\cdots|B_0^1 \\ \Big\downarrow in & & \Big\downarrow in \\ B_1^n & \mathbf{R} & B_0^1|\cdots|B_1^1|\cdots|B_0^1 \end{array}$$

Ex. 4, buffers

$$0 < k < n \qquad B^n_k \quad \mathbf{R} \quad B^1_{k_1} | \cdots | B^1_{k_n} \qquad \sum_{i=1}^{n} k_i = k$$

$$B^n_k \xrightarrow{in} B^n_{k+1} \qquad B^1_{k_1} | \cdots | B^1_{k_n} \xrightarrow{in} B^1_{k_1} | \cdots | B^n_{k_i+1} | \cdots | B^1_{k_n} \qquad \exists k_i = 0$$

$$B^n_{k+1} \;\; \mathbf{R} \;\; B^1_{k_1} | \cdots | B^n_{k_i+1} | \cdots | B^1_{k_n}$$

$$B^n_k \quad \mathbf{R} \quad B^1_{k_1} | \cdots | B^1_{k_n} \qquad \exists k_i = 0$$

$$B^n_k \xrightarrow{in} B^n_{k+1} \qquad B^1_{k_1} | \cdots | B^1_{k_n} \xrightarrow{in} B^1_{k_1} | \cdots | B^n_{k_i+1} | \cdots | B^1_{k_n}$$

$$B^n_{k+1} \;\; \mathbf{R} \;\; B^1_{k_1} | \cdots | B^n_{k_i+1} | \cdots | B^1_{k_n}$$

Ex. 4, buffers

$$0 < k < n \qquad B^n_k \quad \mathbf{R} \quad B^1_{k_1} | \cdots | B^1_{k_n} \qquad \sum_{i=1}^{n} k_i = k$$

$$\downarrow \overline{out} \qquad \downarrow \overline{out} \qquad \exists k_i = 1$$

$$B^n_{k-1} \;\; \mathbf{R} \;\; B^1_{k_1} | \cdots | B^n_{k_i - 1} | \cdots | B^1_{k_n}$$

$$B^n_k \quad \mathbf{R} \quad B^1_{k_1} | \cdots | B^1_{k_n} \qquad \exists k_i = 1$$

$$\downarrow \overline{out} \qquad \downarrow \overline{out}$$

$$B^n_{k-1} \;\; \mathbf{R} \;\; B^1_{k_1} | \cdots | B^n_{k_i - 1} | \cdots | B^1_{k_n}$$

Ex. 4, buffers

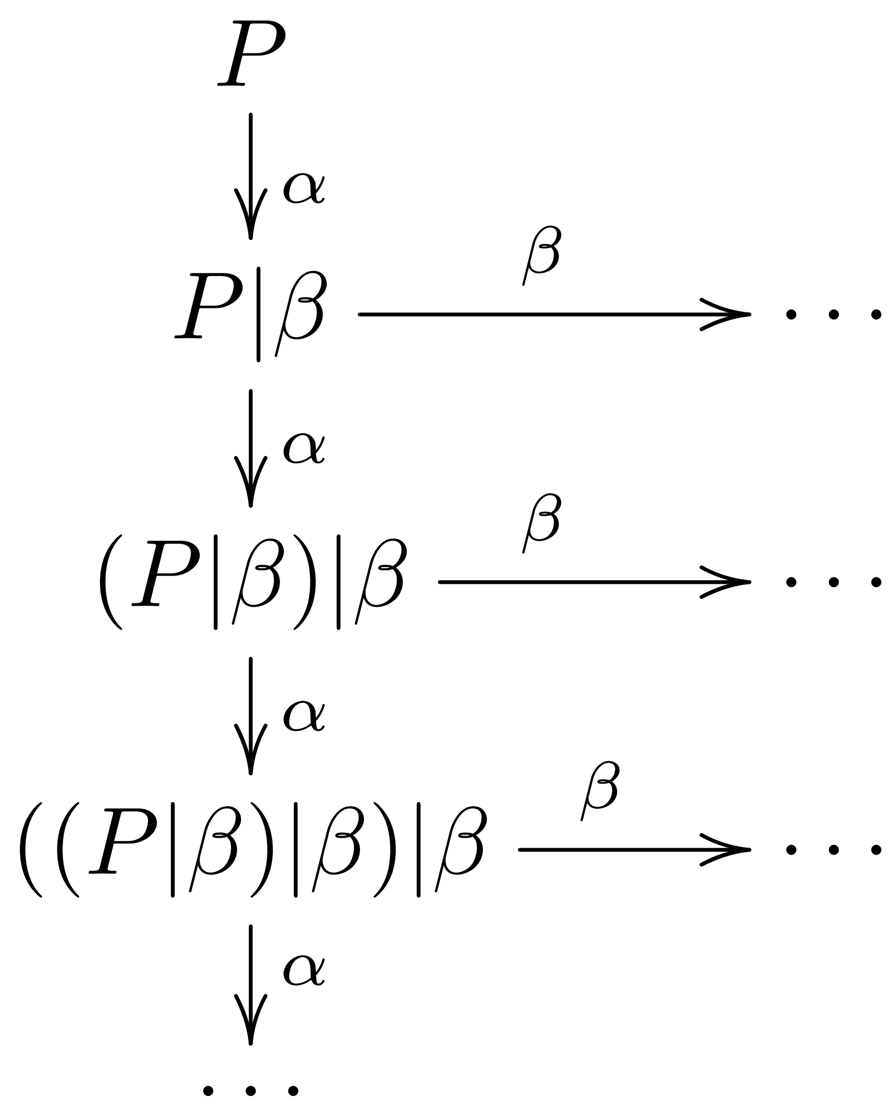
$$\begin{array}{ccc} B^n_n & \mathbf{R} & B^1_1|\cdots|B^1_1 \\ \downarrow \overline{out} & & \downarrow \overline{out} \\ B^n_{n-1} & \mathbf{R} & B^1_0|\cdots|B^1_1 \end{array}$$

$$\begin{array}{ccc} B^n_n & \mathbf{R} & B^1_1|\cdots|B^1_1 \\ \downarrow \overline{out} & & \downarrow \overline{out} \\ B^n_{n-1} & \mathbf{R} & B^1_1|\cdots|B^1_0|\cdots|B^1_1 \end{array}$$

[**Ex. 5**] Write a guarded CCS process whose LTS has infinitely many states without using parallel composition.

Ex. 5, infinite LTS

using par $P \triangleq \mathbf{rec}\ x.\ \alpha.(x|\beta.\mathbf{nil})$ $P \triangleq \alpha.(P|\beta)$



Ex. 5, infinite LTS

$P \triangleq \mathbf{rec}\ x.\ (\alpha.x)\backslash\beta$ $\qquad$ $P \triangleq (\alpha.P)\backslash\beta$

$$P \xrightarrow{\alpha} P\backslash\beta \xrightarrow{\alpha} P\backslash\beta\backslash\beta \xrightarrow{\alpha} P\backslash\beta\backslash\beta\backslash\beta \xrightarrow{\alpha} \cdots$$

all states are bisimilar

Ex. 5, infinite LTS

$$P \triangleq \mathbf{rec}\ x.\ \alpha_1.(x[\phi]) \qquad \phi(\alpha_i) = \alpha_{i+1}$$

$$P \xrightarrow{\alpha_1} P[\phi] \xrightarrow{\alpha_2} P[\phi][\phi] \xrightarrow{\alpha_3} P[\phi][\phi][\phi] \xrightarrow{\alpha_4} \cdots$$

Bisimulation

[**Ex. 6**] Prove that CCS strong bisimilarity is a congruence w.r.t. restriction, i.e., that for all p, q, α:

$$p \simeq q \;\Rightarrow\; p\backslash\alpha \simeq\; q\backslash\alpha$$

Ex. 6, congruence \a

$\mathbf{R} \triangleq \{(p\backslash\alpha, q\backslash\alpha) \mid p \simeq q\}$ we prove **R** is a strong bisimulation

take $(p\backslash\alpha, q\backslash\alpha) \in \mathbf{R}$ (with $p \simeq q$)

take $p\backslash\alpha \xrightarrow{\mu} p'$ we want to find $q\backslash\alpha \xrightarrow{\mu} q'$ with $p' \, \mathbf{R} \, q'$

by rule res) it must be $p \xrightarrow{\mu} p''$ with $\mu \notin \{\alpha, \overline{\alpha}\}$ and $p' = p''\backslash\alpha$

since $p \simeq q$ and $p \xrightarrow{\mu} p''$ we have $q \xrightarrow{\mu} q''$ with $p'' \simeq q''$

take $q' = q''\backslash\alpha$: by rule res) we have $q\backslash\alpha \xrightarrow{\mu} q'$ and $(p', q') \in \mathbf{R}$

take $q\backslash\alpha \xrightarrow{\mu} q'$ we want to find $p\backslash\alpha \xrightarrow{\mu} p'$ with $p' \, \mathbf{R} \, q'$

analogous to the previous case

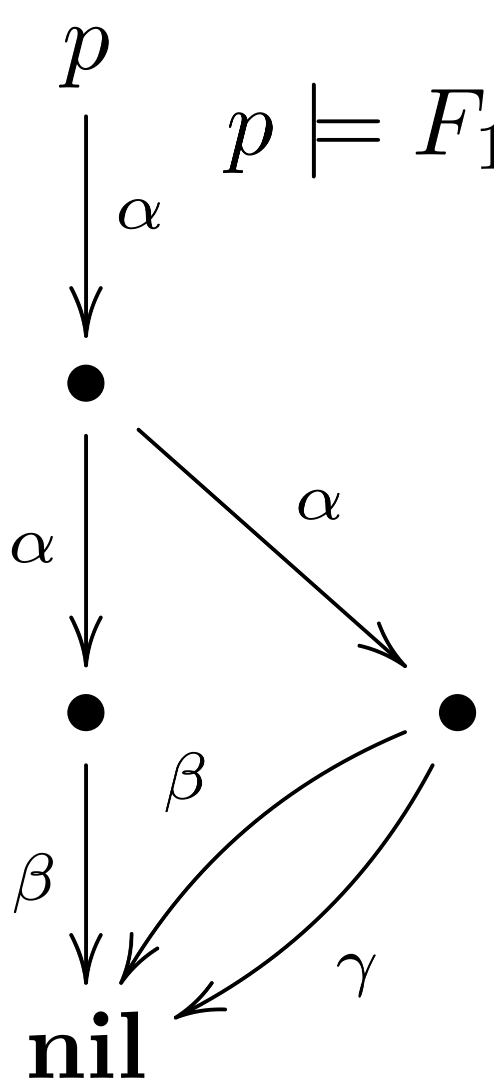
[Ex. 7] Prove that the CCS agents

$$p \stackrel{\text{def}}{=} \alpha.(\alpha.\beta.\mathbf{nil} + \alpha.(\beta.\mathbf{nil} + \gamma.\mathbf{nil})) \qquad \text{and} \qquad q \stackrel{\text{def}}{=} \alpha.(\alpha.\beta.\mathbf{nil} + \alpha.\gamma.\mathbf{nil})$$

are not strong bisimilar.

Ex. 7, non bisimilar

$p \triangleq \alpha.(\alpha.\beta + \alpha.(\beta + \gamma))$ $q \triangleq \alpha.(\alpha.\beta + \alpha.\gamma)$



$F_1 \triangleq \Diamond_\alpha \Diamond_\alpha (\Diamond_\beta \mathbf{tt} \wedge \Diamond_\gamma \mathbf{tt})$

$p \models F_1$ $q \not\models F_1$

$F_2 \triangleq \Box_\alpha \Box_\alpha \Diamond_\beta \mathbf{tt}$

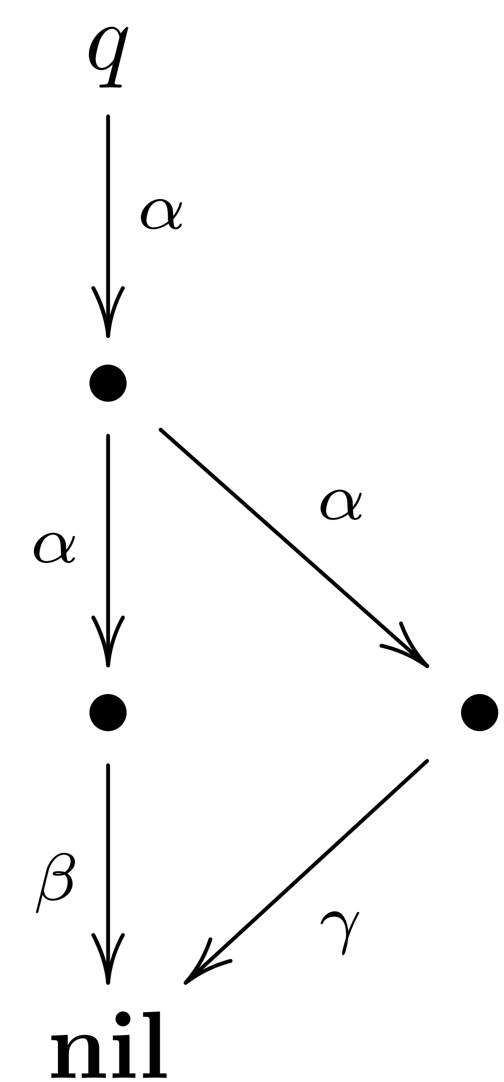
$p \models F_2$ $q \not\models F_2$

$F_2^c \triangleq \Diamond_\alpha \Diamond_\alpha \Box_\beta \mathbf{ff}$

$p \not\models F_2^c$ $q \models F_2^c$

$F_1^c \triangleq \Box_\alpha \Box_\alpha (\Box_\beta \mathbf{ff} \vee \Box_\gamma \mathbf{ff})$

$p \not\models F_1^c$ $q \models F_1^c$



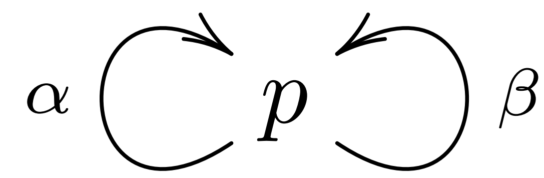
[**Ex. 8**] Let us consider the guarded CCS processes

$$p \stackrel{\text{def}}{=} \textbf{rec } x.(\alpha.x + \beta.x) \quad q \stackrel{\text{def}}{=} \textbf{rec } y.(\overline{\alpha}.\textbf{nil} + \gamma.y) \quad r \stackrel{\text{def}}{=} \textbf{rec } z.(\overline{\beta}.\textbf{nil} + \overline{\gamma}.z)$$

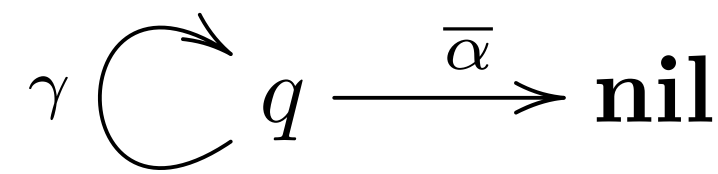
1. Draw the LTSs of the processes p, q, r and $s \stackrel{\text{def}}{=} (p|q|r)\backslash\alpha\backslash\beta\backslash\gamma$.
2. Show that s is strong bisimilar to the process $t \stackrel{\text{def}}{=} \textbf{rec } w.(\tau.w + \tau.\tau.\textbf{nil})$.

Ex. 8, bisimilar

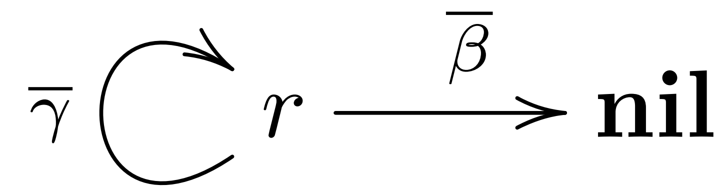
$$p \triangleq \mathbf{rec}\ x.(\alpha.x + \beta.x)$$



$$q \triangleq \mathbf{rec}\ y.(\overline{\alpha} + \gamma.y)$$



$$r \triangleq \mathbf{rec}\ z.(\overline{\beta} + \overline{\gamma}.z)$$

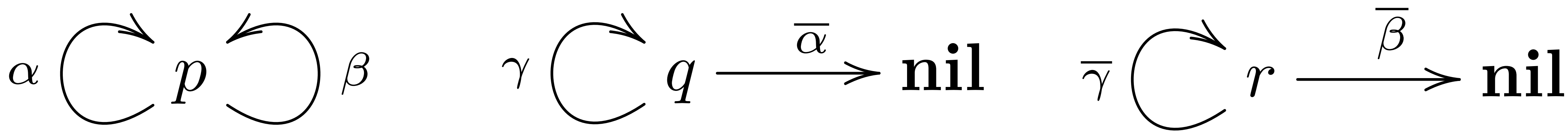


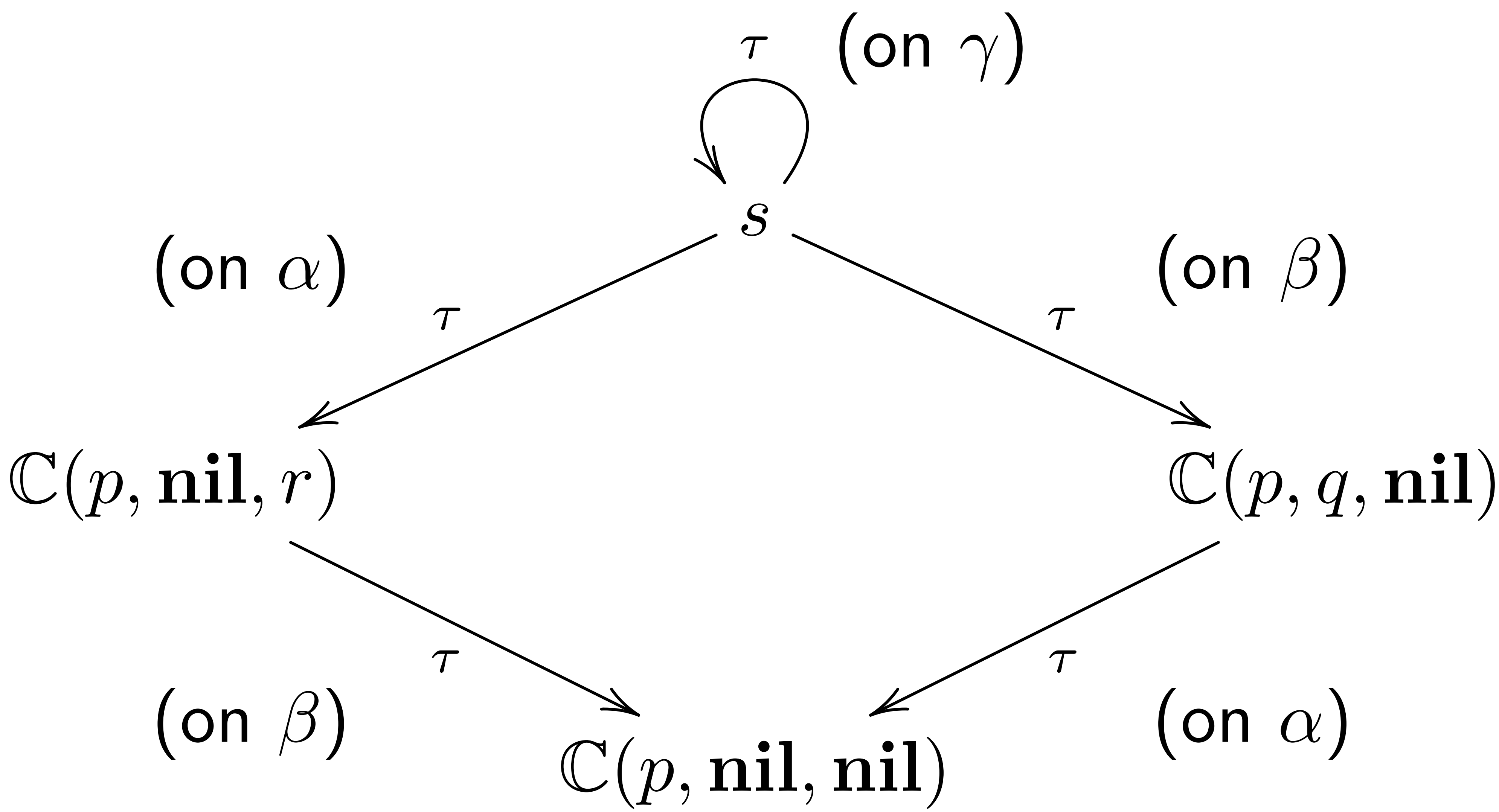
$$s \triangleq (p|q|r)\backslash\alpha\backslash\beta\backslash\gamma$$

$$\mathbb{C}(p,q,r) \triangleq (p|q|r)\backslash\alpha\backslash\beta\backslash\gamma$$

$$s \triangleq \mathbb{C}(p,q,r)$$

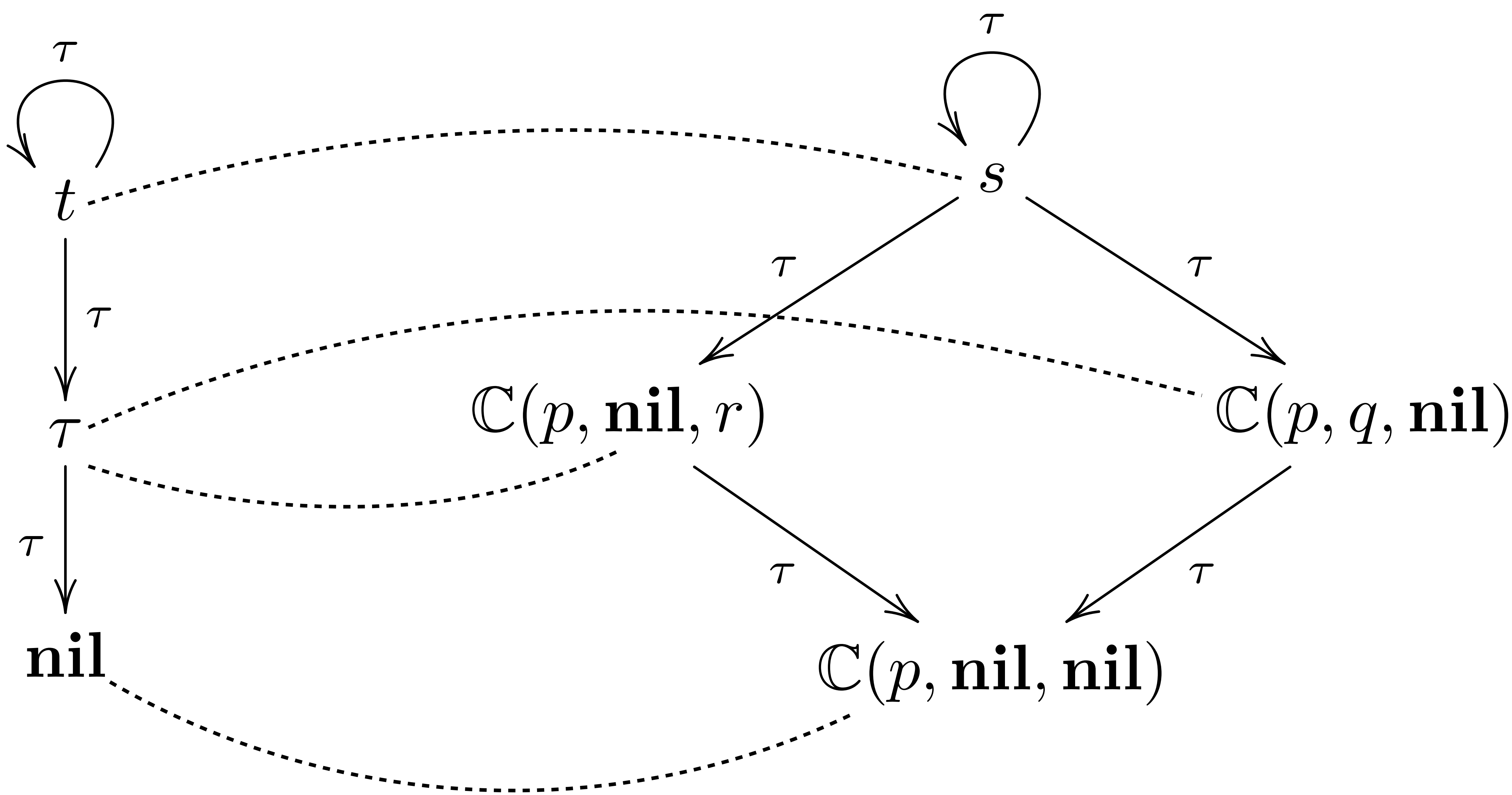
Ex. 8, bisimilar





Ex. 8, bisimilar

$t \triangleq \mathbf{rec}\ w.(\tau.w + \tau.\tau)$



$$\mathbf{R} \triangleq \{\, \{s, t\}\,,\ \{\tau, \mathbb{C}(p, \mathbf{nil}, r), \mathbb{C}(p, q, \mathbf{nil})\}\,,\ \{\mathbf{nil}, \mathbb{C}(p, \mathbf{nil}, \mathbf{nil})\}\,\}$$

R is a strong bisimulation

[**Ex. 9**] Prove that the following property is valid for any agent p, where $\approx$ is the weak bisimilarity:

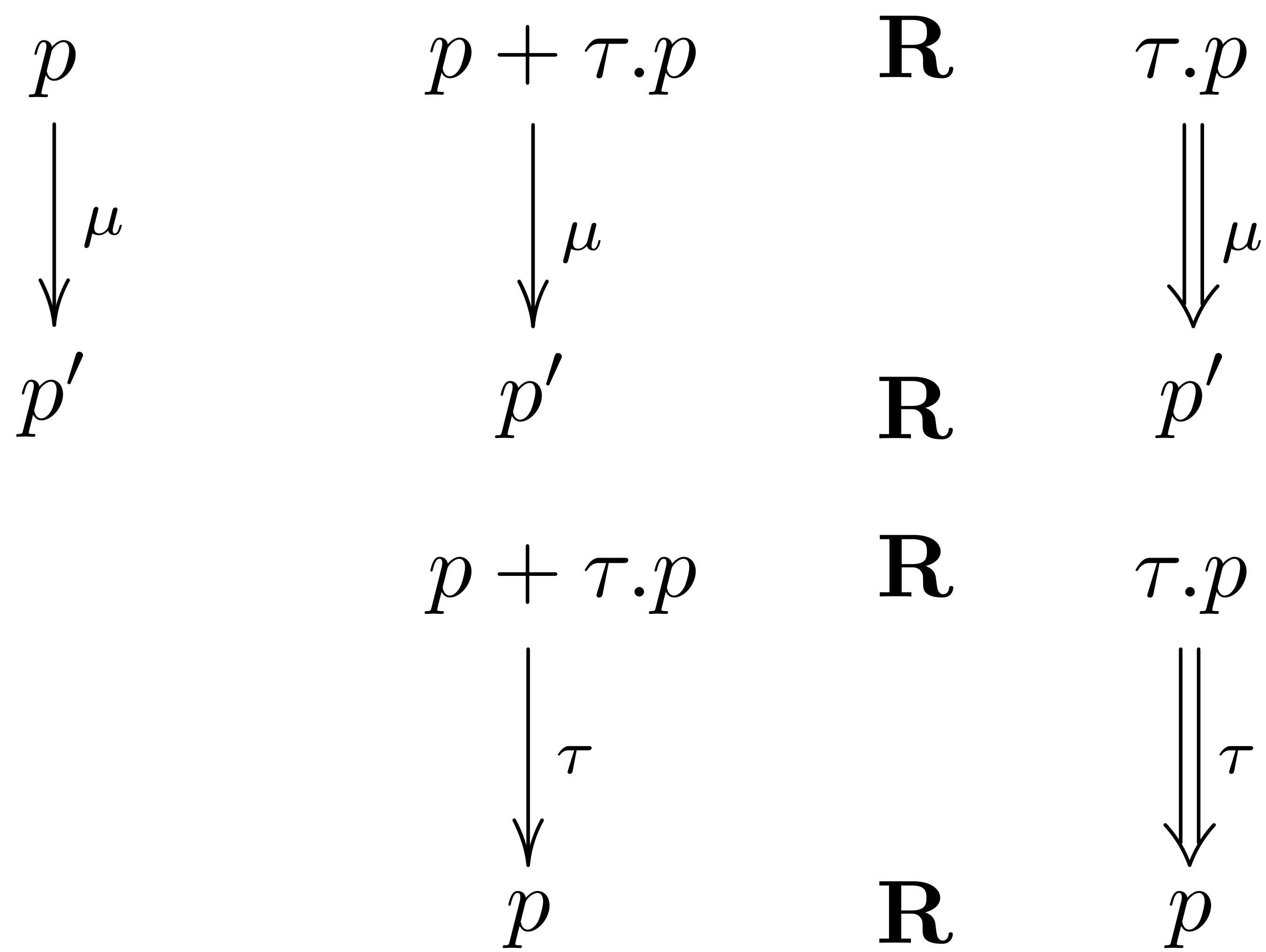
$$p + \tau.p \approx \tau.p$$

Ex. 9, weak bisimilar

$$\mathbf{R} \triangleq \{(p + \tau.p, \tau.p) \mid p \in \mathcal{P}\} \cup Id$$

we check that **R** is a weak bisimulation

no need to check pairs in Id



Ex. 9, weak bisimilar

$$\mathbf{R} \triangleq \{(p + \tau.p, \tau.p) \mid p \in \mathcal{P}\} \cup Id$$

we check that **R** is a weak bisimulation

no need to check pairs in Id

$$\begin{array}{ccc} p + \tau.p & \mathbf{R} & \tau.p \\ \Downarrow \tau & & \downarrow \tau \\ p & \mathbf{R} & p \end{array}$$