Co-Algebraic HD-automata: Implementation of a Partitioning Algorithm

Roberto Raggi & Emilio Tuosto

joint work with

Gianluigi Ferrari, Ugo Montanari and Marco Pistore

Plan of the talk

Plan of the talk

- Principal data structures

Plan of the talk

- Principal data structures
- The main cicle

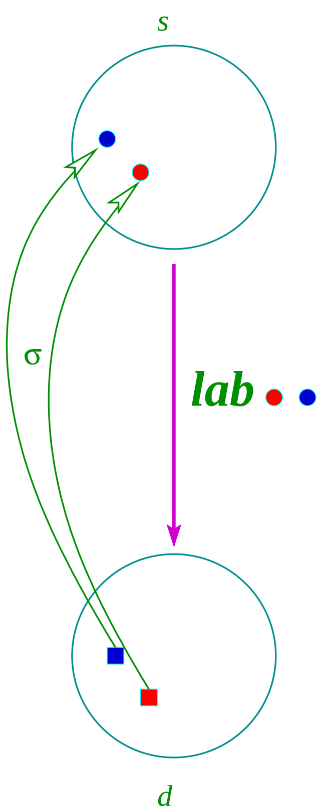
Plan of the talk

- Principal data structures
- The main cicle
- An example

Plan of the talk

- Principal data structures
- The main cicle
- An example
- Final considerations

States, labels & arrows



type α state =

| Star

| State of $\underbrace{\text{string}}_{\text{id}} \;*\; \underbrace{\alpha\ \text{list}}_{\text{names}} \;*\; \underbrace{(\alpha\ \text{list})\ \text{list}}_{\text{group}}$

type α label =

| Tau of α list

| BIn of α list $*$ α list

| BOut of α list $*$ α list

| In of α list $*$ α list

| Out of $\underbrace{\alpha\ \text{list}}_{\pi} \;*\; \underbrace{\alpha\ \text{list}}_{\sigma}$

type α arrow =

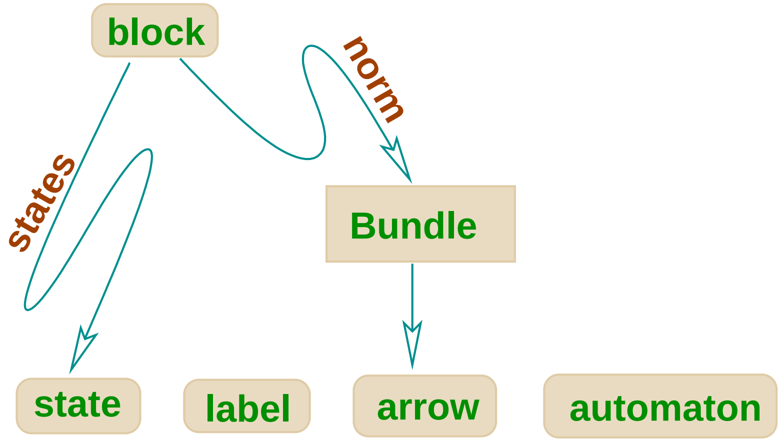
Arrow of $\underbrace{\alpha\ \text{state}}_{\text{source}} \;*\; \underbrace{\alpha\ \text{state}}_{\text{dest}} \;*\; \underbrace{\alpha\ \text{label}}_{\text{lab}}$

Bundle & Automata

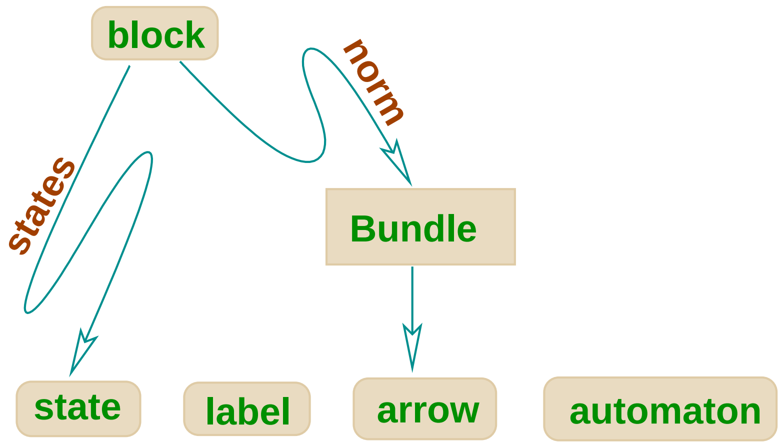
bundle : $\underbrace{(\alpha \text{ arrow}) \quad \text{list}}_{\text{with the same source}}$

type α automaton =
HDAutoma of $\underbrace{\alpha \text{ state}}_{\text{start}} * \underbrace{(\alpha \text{ state}) \text{ list}}_{\text{states}} * \underbrace{(\alpha \text{ arrow}) \text{ list}}_{\text{arrows}}$

Blocks



Blocks



At the end of each iteration,

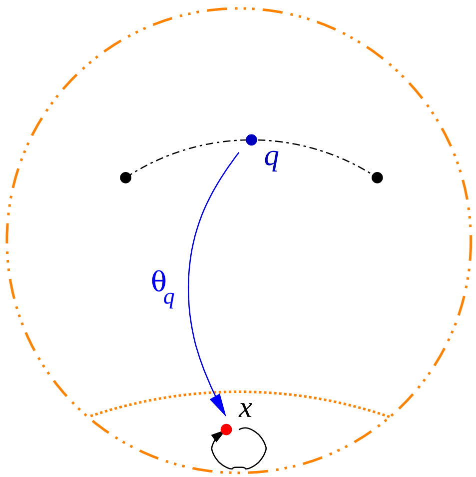
- blocks represent the states of the n-th approximation of the minimal automaton while
- their norm components are the arrows of the approximation

Blocks (2)

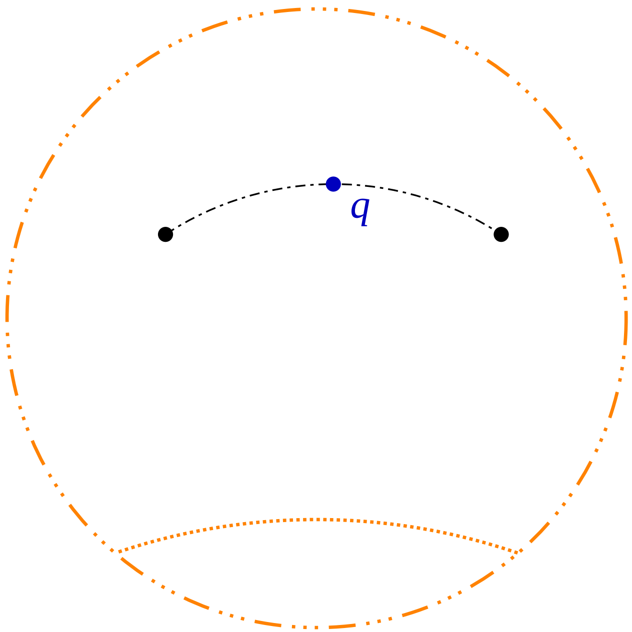
type α blocks =
  Block of
    states : α state list $*$
    norm : α arrow list $*$
    active_names : α list $*$
    group : α list list $*$
    Σ : (α state $\rightarrow$ ($\alpha * \alpha$) list list) $*$
    Θ^{-1} : (α state $\rightarrow$ ($\alpha * \alpha$) list)

Blocks (2)

type α blocks =
  Block of
    states : α state list $*$
    norm : α arrow list $*$
    active_names : α list $*$
    group : α list list $*$
    Σ : (α state $\rightarrow$ (α $*$ α) list list) $*$
    Θ^{-1} : (α state $\rightarrow$ (α $*$ α) list)



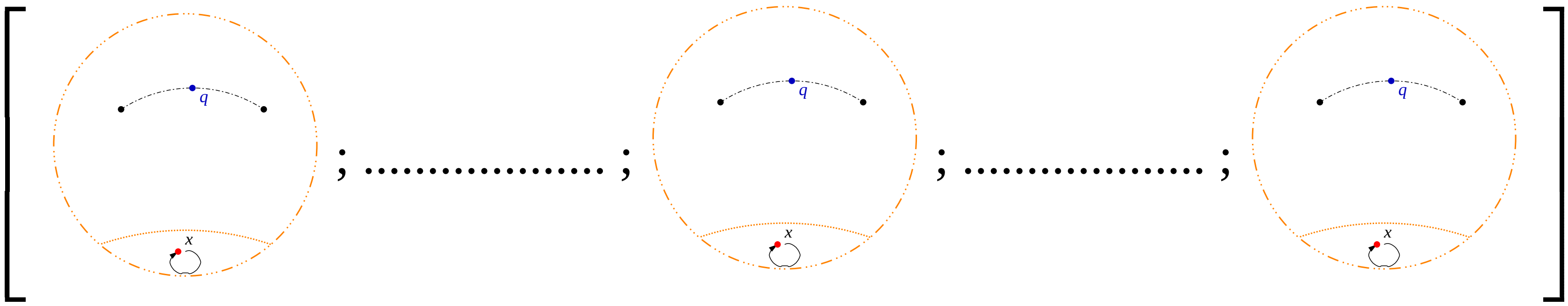
Initially...



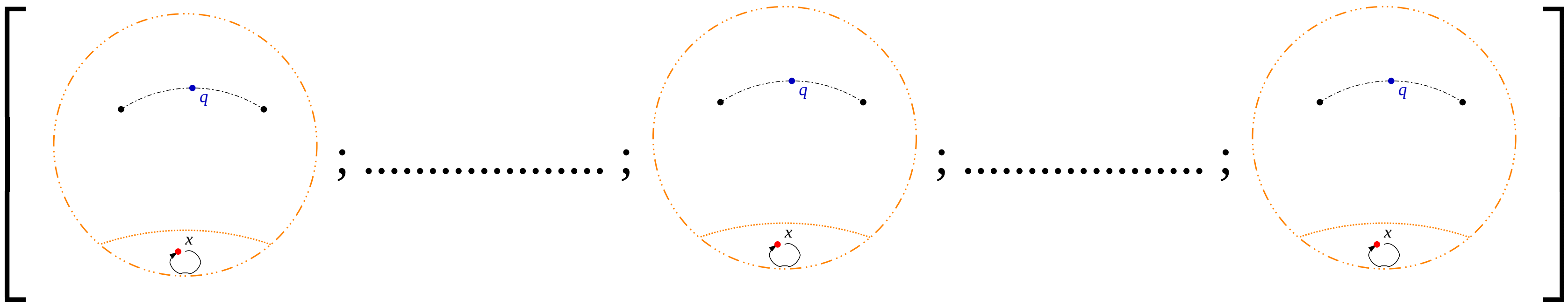
- All the states are (considered) bisimilar
- No norm, group or θ is given

[**Block**(states, [ ], [ ], [ ], (fun q $\rightarrow$ [[ ]]), (fun q $\rightarrow$ [ ]))]

Generic step

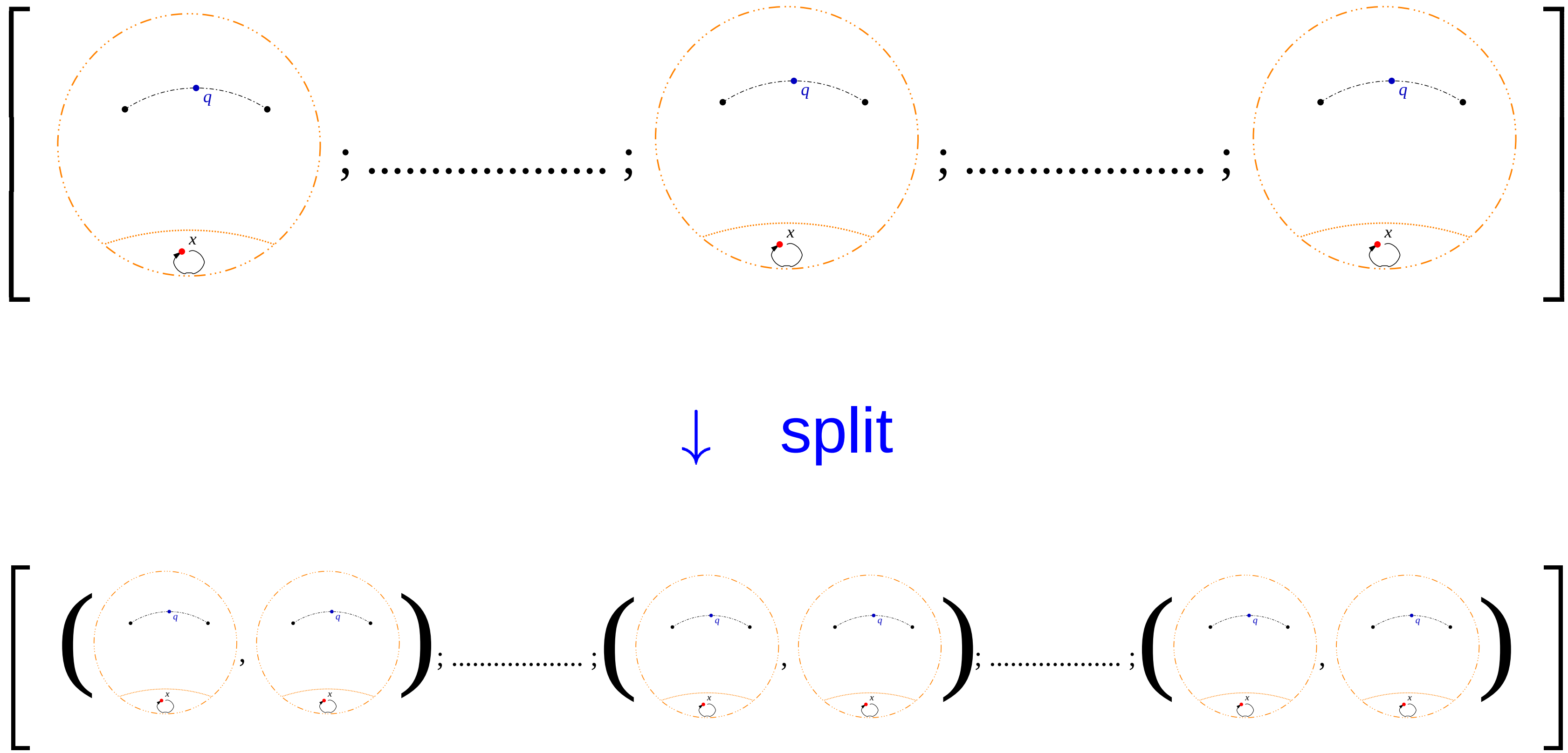


Generic step

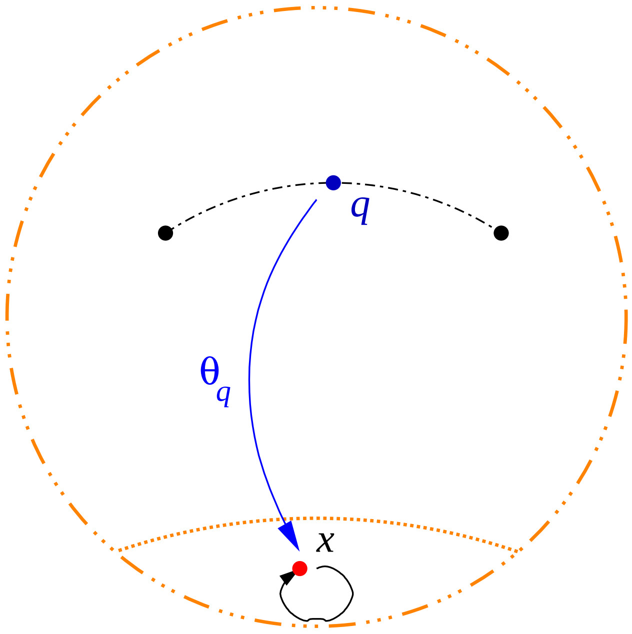


↓ split

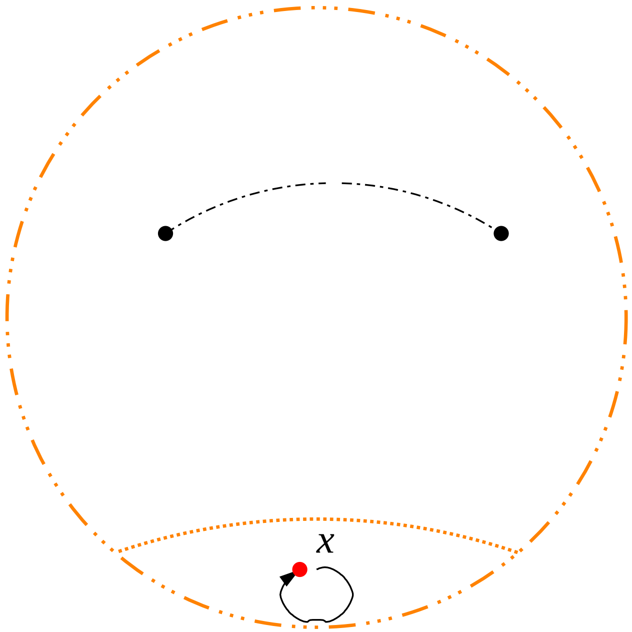
Generic step

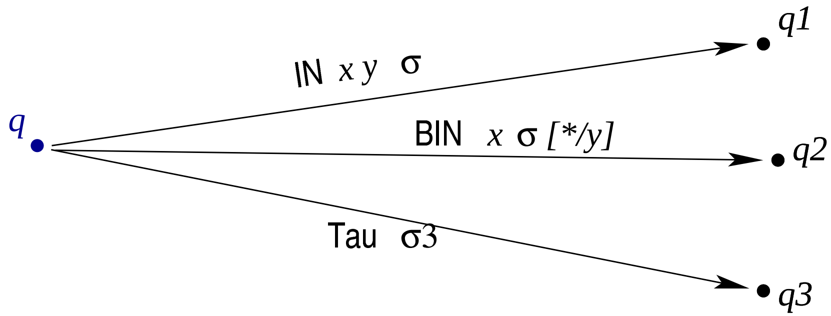


Splitting: a closer look



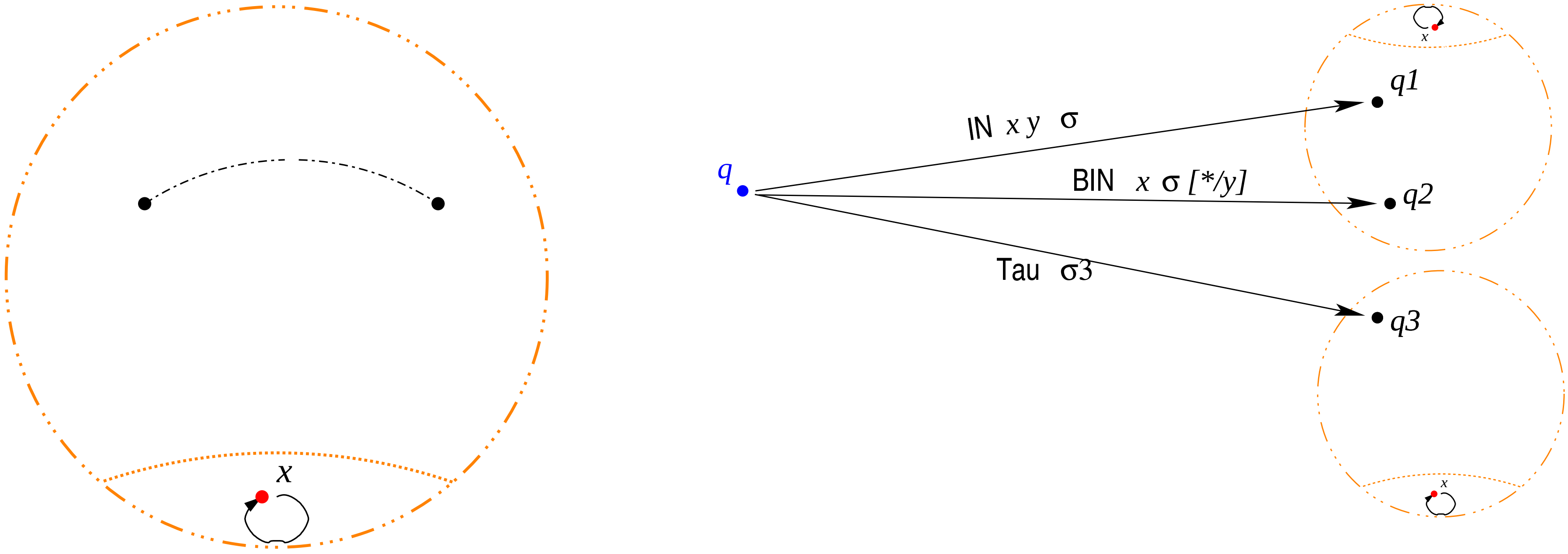
Splitting: a closer look





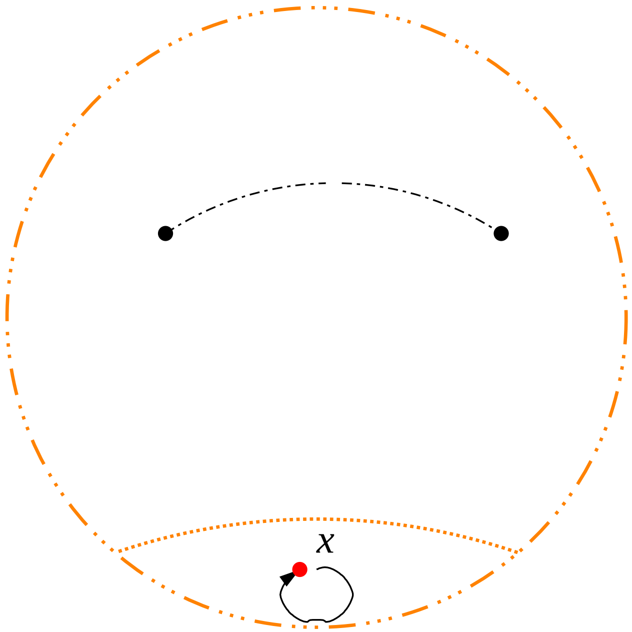
let bundle hd q =
  List.sort compare
    (List.filter (fun h $\rightarrow$ (Arrow.source h) = q) (arrows hd))

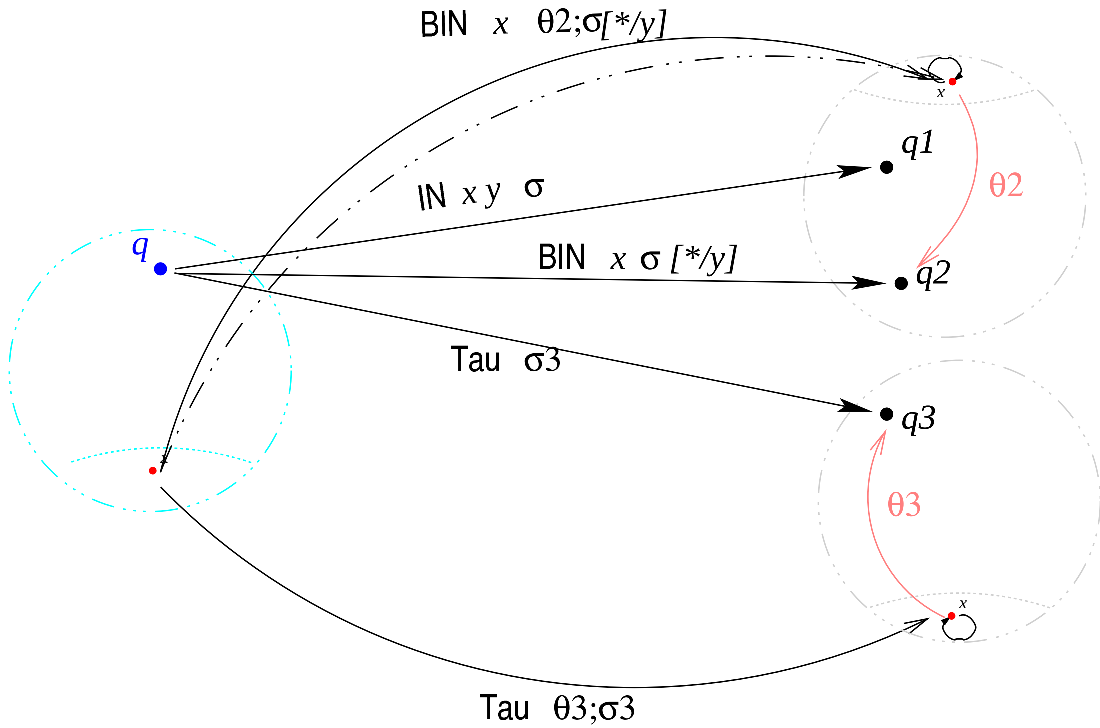
Splitting: a closer look



List.map h_n bundle

Splitting: a closer look

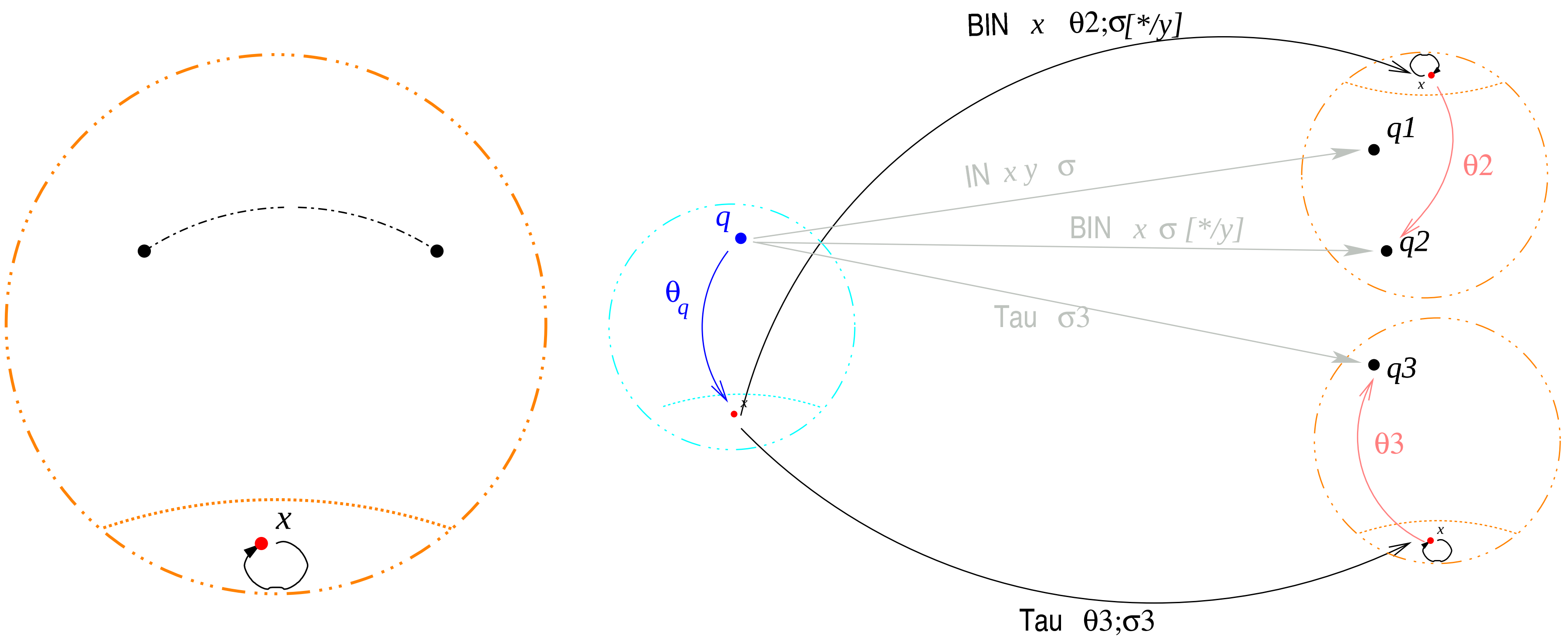




$$h_{n+1} = norm\langle \text{states}, \{\langle \ell, \pi, h_n(q'), \sigma';\sigma\rangle \mid q \xrightarrow{\ell\ \pi\ \sigma} q' \ \wedge\ \sigma' \in \Sigma_n(q')\}\rangle$$

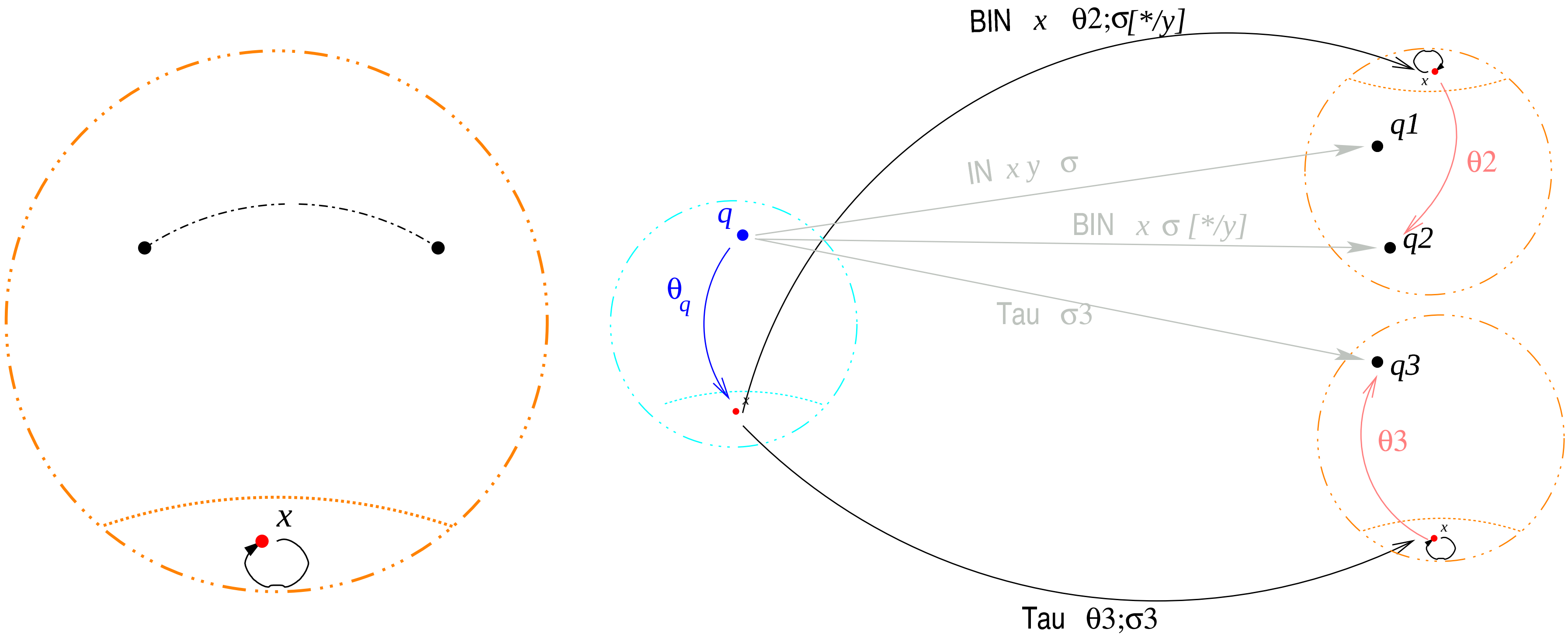
```
let red bl = ......
  let bl_in = List.filter covered_in bl
  in list_diff bl bl_in
```

Splitting: a closer look




```
let an = active_names_bundle (red bundle) in
let remove_in ar = match ar with
  | Arrow(_,_,In(_,_)) → not (List.mem (obj ar) an)
  | _ → false in
    list_diff bundle (List.filter remove_in bundle)
```

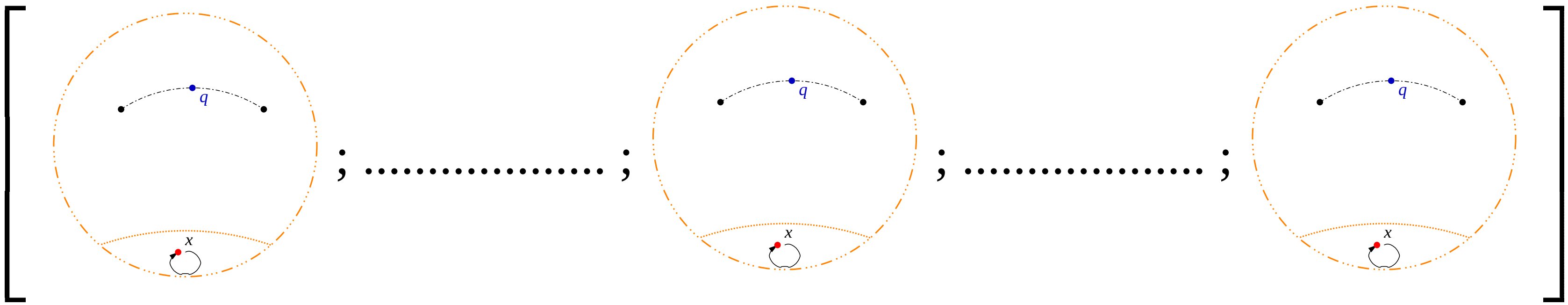
Splitting: a closer look



$$\Sigma_{n+1}(q) = (\text{compute\_group}\ \ (\text{norm}\ \ \textsf{bundle}))\ \ ;\ \ \theta_q^{-1}$$

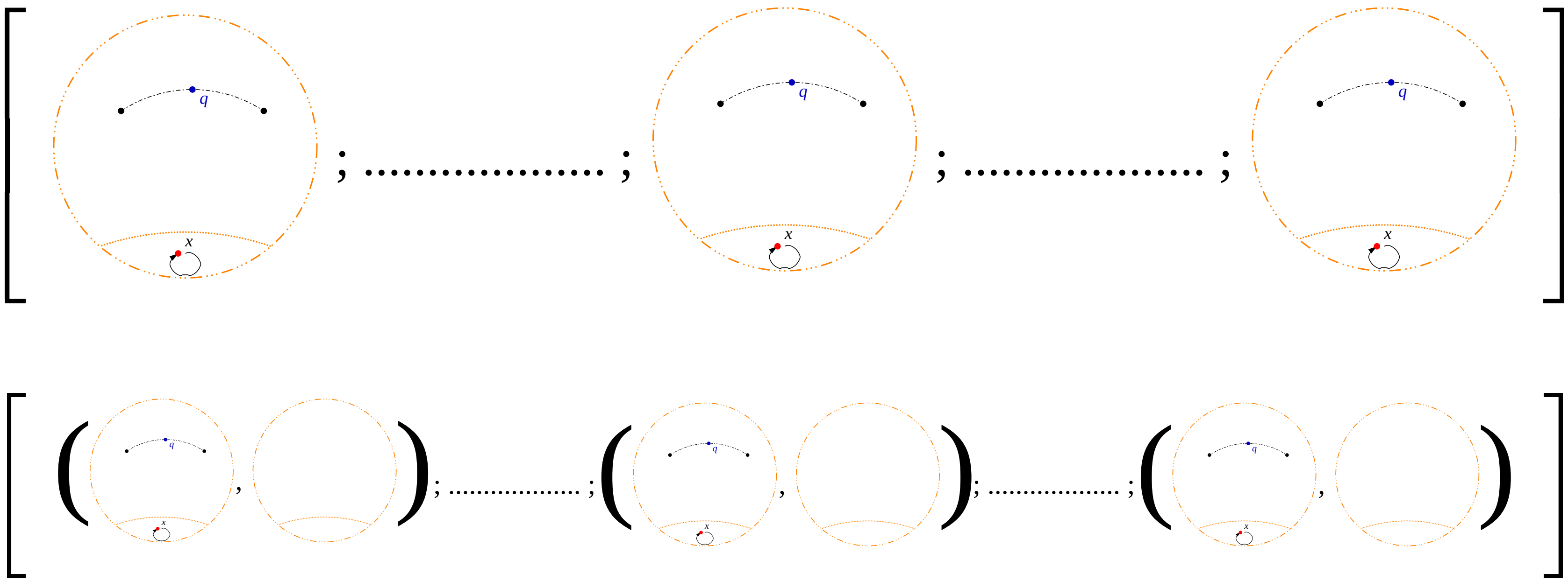
Termination

...informally, when H_{n+1} is isomorph to H_n



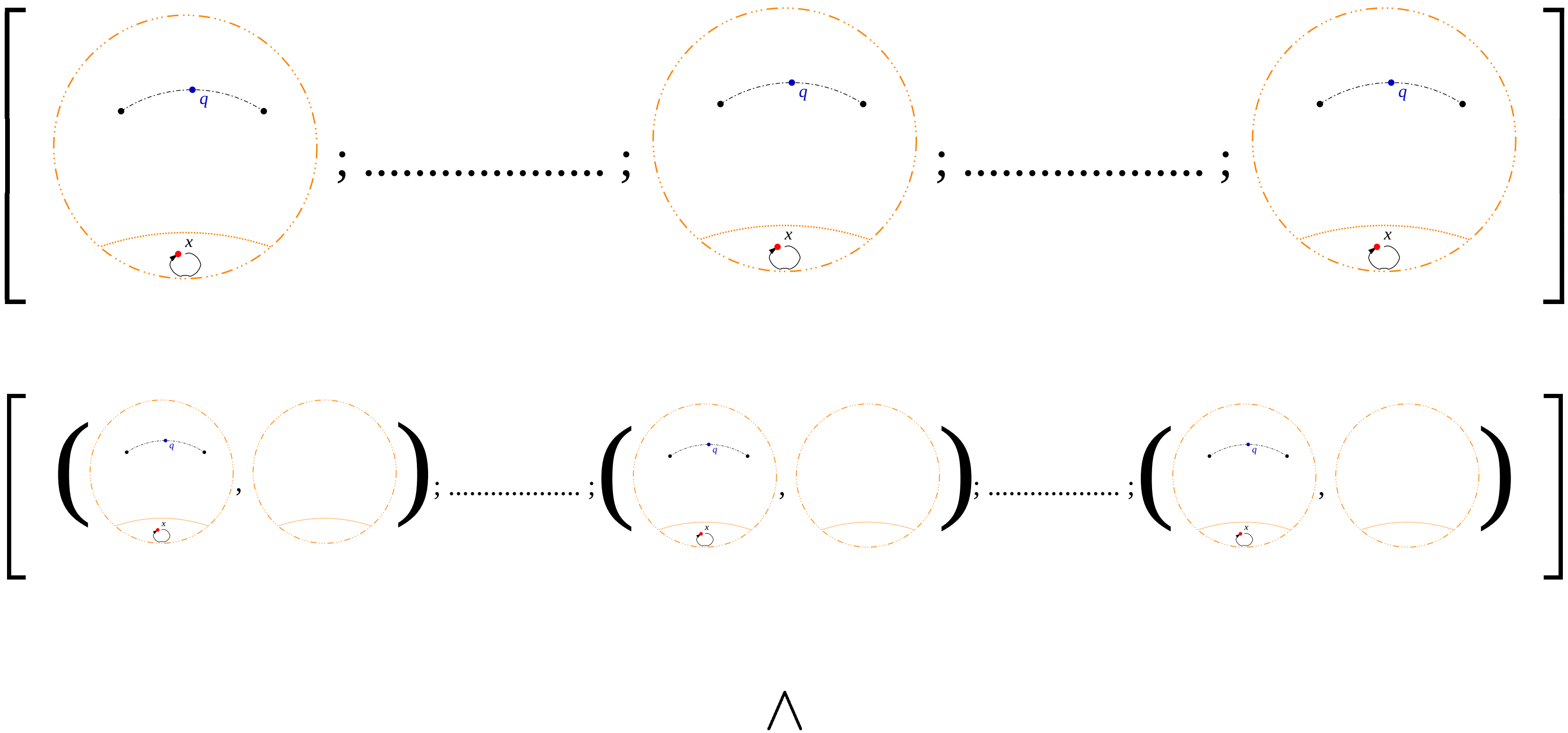
Termination

...informally, when H_{n+1} is isomorph to H_n



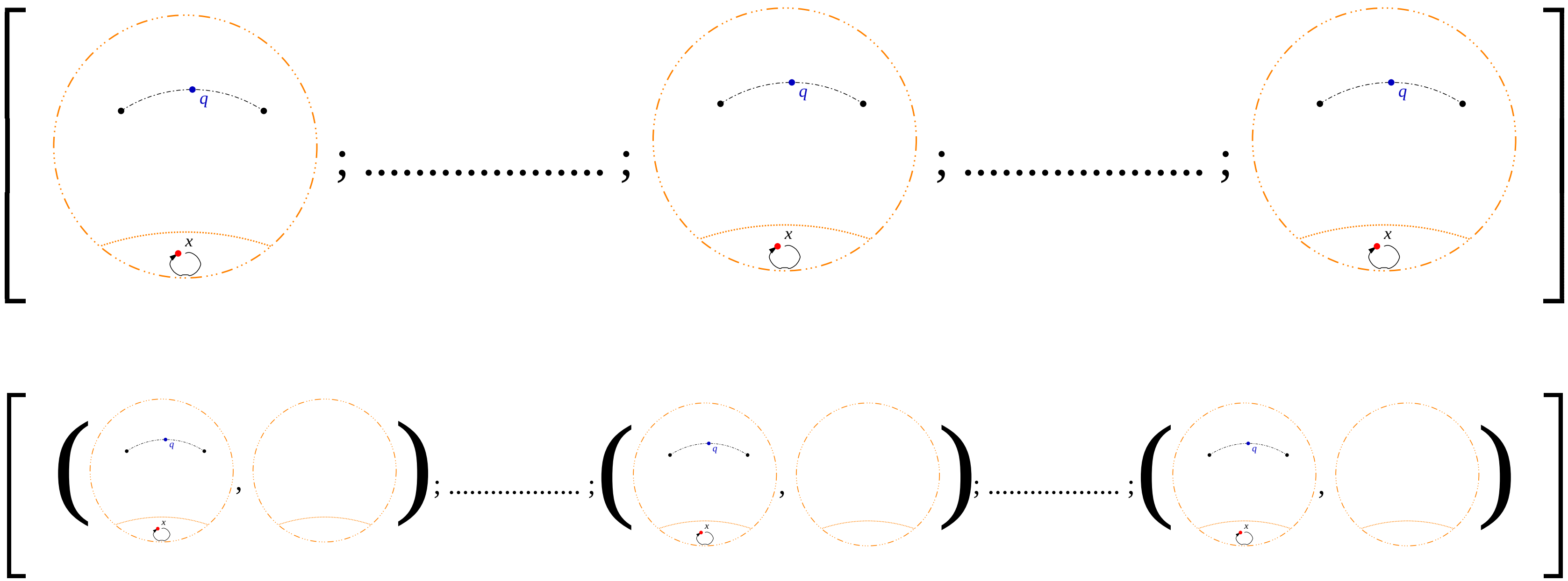
Termination

...informally, when H_{n+1} is isomorph to H_n



Termination

...informally, when H_{n+1} is isomorph to H_n



$$\wedge$$

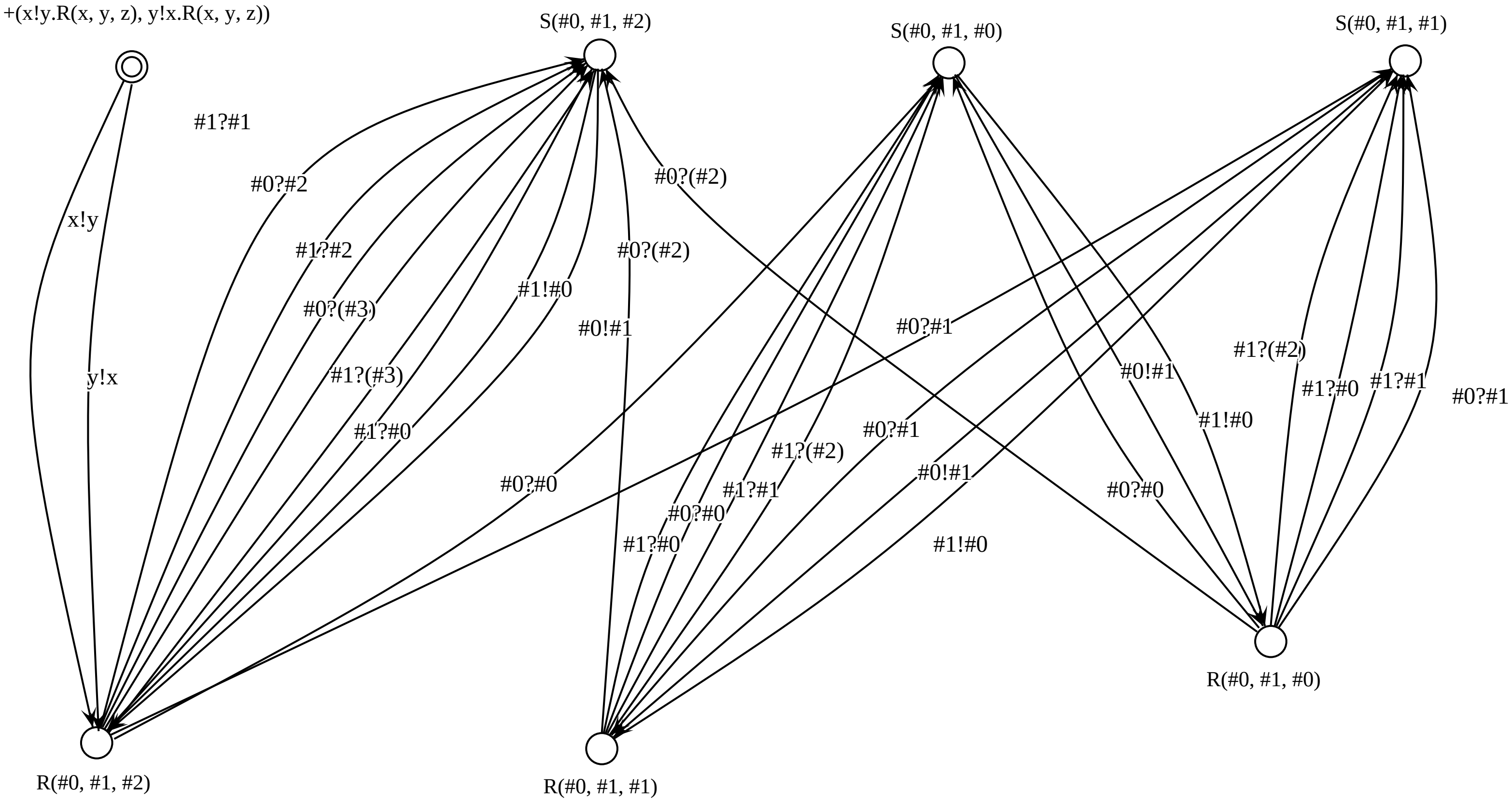
no further names are added

An example

$$S(x, y, z) = x!y.R(x, y, z) + y!x.R(x, y, z)$$
$$R(x, y, z) = x?(w).S(x, y, w) + y?(w).S(y, x, z)$$

An example

$$S(x, y, z) = x!y.R(x, y, z) + y!x.R(x, y, z)$$
$$R(x, y, z) = x?(w).S(x, y, w) + y?(w).S(y, x, z)$$



Minimal representation

state	b0		2	[[1 ; 2] ; [2 ; 1]]
state	b1		2	[[1 ; 2] ; [2 ; 1]]
b0	$\rightarrow$	b1	out[1 ; 2]	[1 ; 2]
b0	$\rightarrow$	b1	out[1 ; 2]	[2 ; 1]
b0	$\rightarrow$	b1	out[2 ; 1]	[1 ; 2]
b0	$\rightarrow$	b1	out[2 ; 1]	[2 ; 1]
b1	$\rightarrow$	b0	bin[1]	[1 ; 2]
b1	$\rightarrow$	b0	bin[1]	[2 ; 1]
b1	$\rightarrow$	b0	bin[2]	[2 ; 1]
b1	$\rightarrow$	b0	bin[2]	[1 ; 2]
b1	$\rightarrow$	b0	in[1 ; 1]	[1 ; 2]
b1	$\rightarrow$	b0	in[1 ; 1]	[2 ; 1]
b1	$\rightarrow$	b0	in[1 ; 2]	[1 ; 2]
b1	$\rightarrow$	b0	in[1 ; 2]	[2 ; 1]
b1	$\rightarrow$	b0	in[2 ; 1]	[1 ; 2]
b1	$\rightarrow$	b0	in[2 ; 1]	[2 ; 1]
b1	$\rightarrow$	b0	in[2 ; 2]	[1 ; 2]
b1	$\rightarrow$	b0	in[2 ; 2]	[2 ; 1]

Final remarks

- Extend to
  - open bisimilarity
  - asynchronous π-calculus
  - complex terms
- Try *big* automata
- Optimization: handling of permutations
- Integrations (HAL and Mobility workbench)
- Ocaml compiler