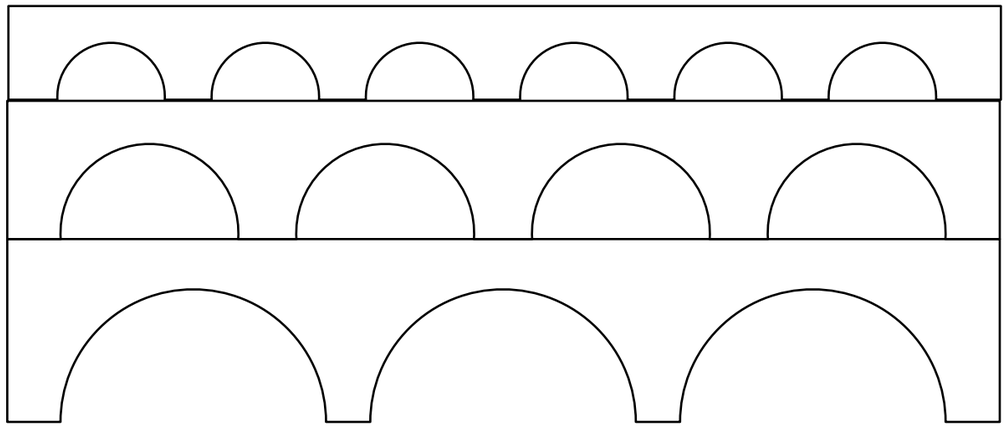
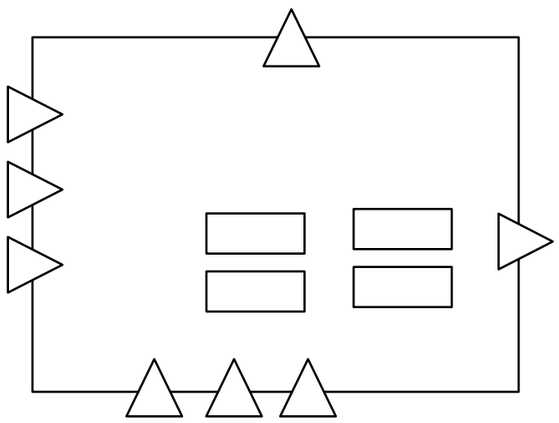
The Roman aqueduct model

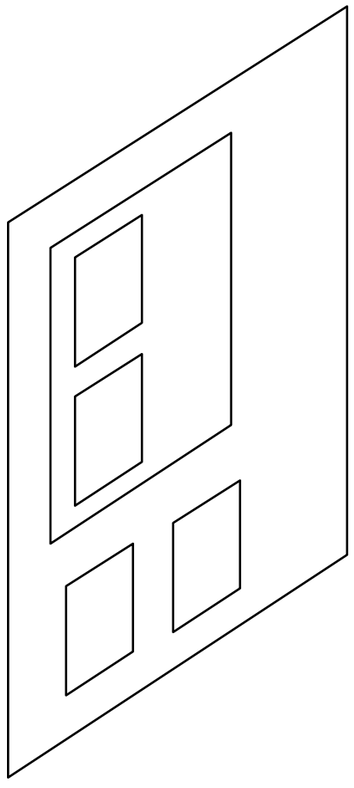


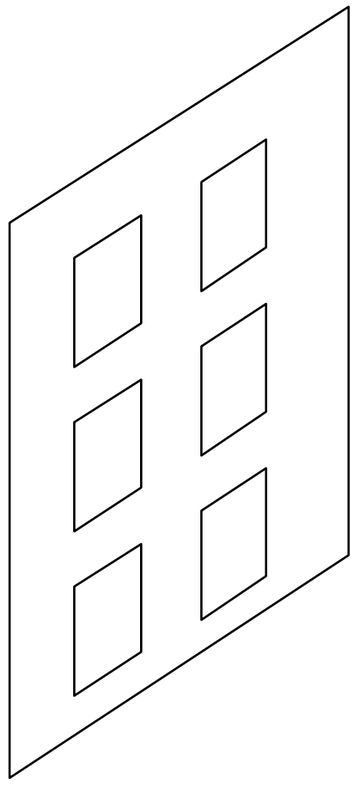
Concepts of toolkits

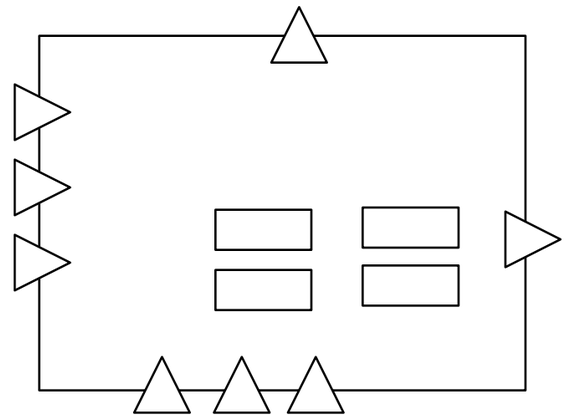
Some common concepts

Undefined Base concepts





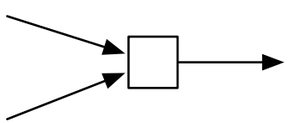




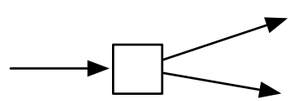
Grandes dualités

Continu / Discret

[abowd22integrating]



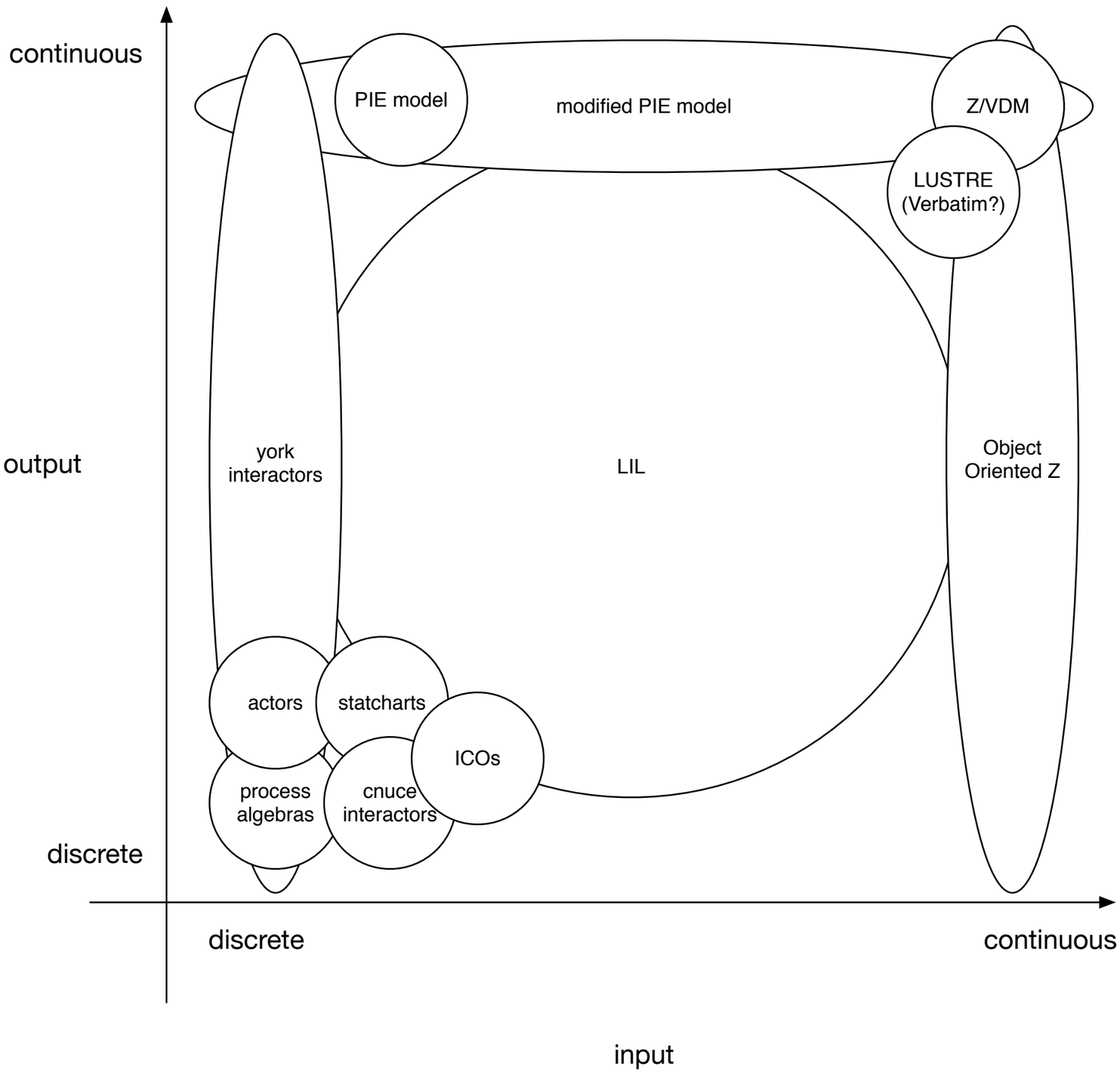
Structuration / Destructuration



Synthetizated / State

React synthesized variables vs state

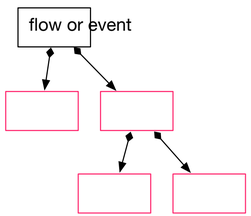
\cite{abowd22integrating}



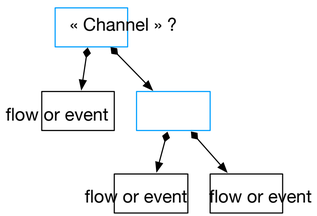
oncept of composition of flows and events

Top - Down

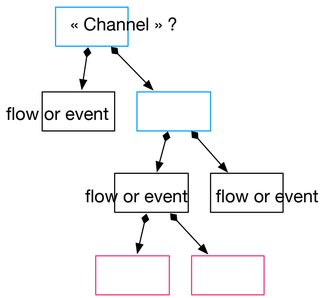
(OLD) LIL



Bottom - Leafs



Mixed ?



data: not specified if flow or event, just data

flow or event

channel : combination of flows and events

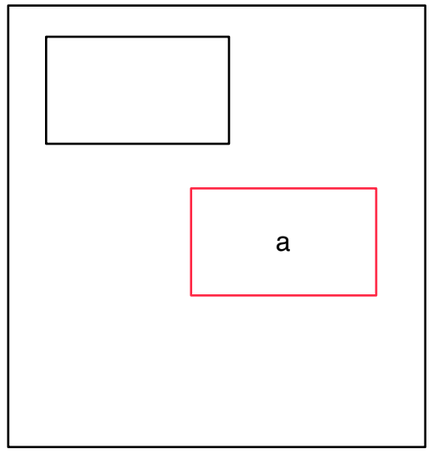
```
parent : channel{
  active: boolean flow in,
  visible : boolean flow in,
  selection : void event out,
  mouse : channel{
    position : position flow in,
    click : button event in
  }
}
```

Concept of multi-composition

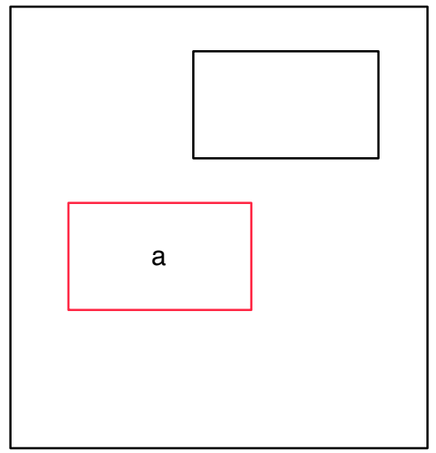
an interaction is part of MULTIPLE interactors
it is SHARED between them

one has access to it and so does the other

an interactor has multiple parents



user hierarchy

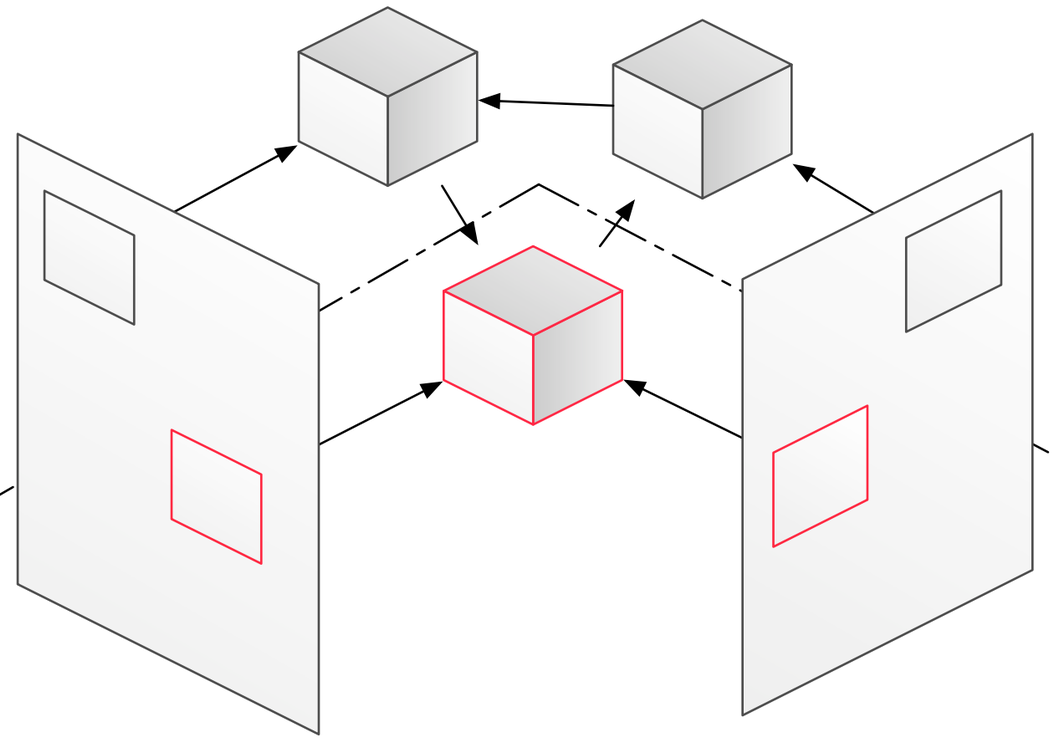


machine hierarchy

an actor is a perspective
an actor is a projection axis
an actor is a direction

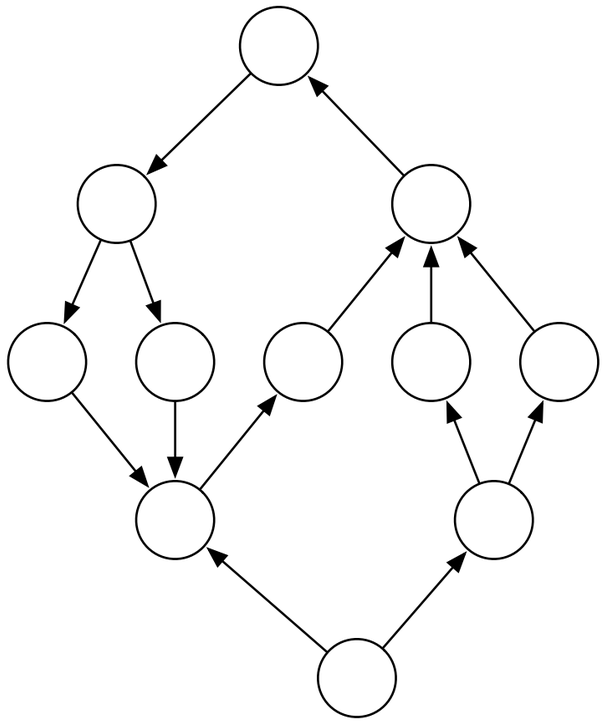
an interactor has got as much parents
as actors he deals with

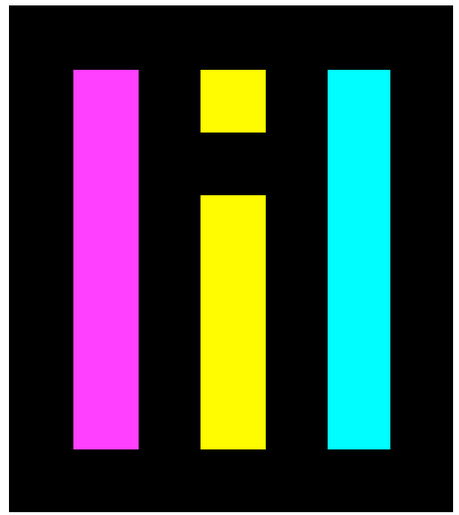
So we have as many interactor hierarchies
as we have actors

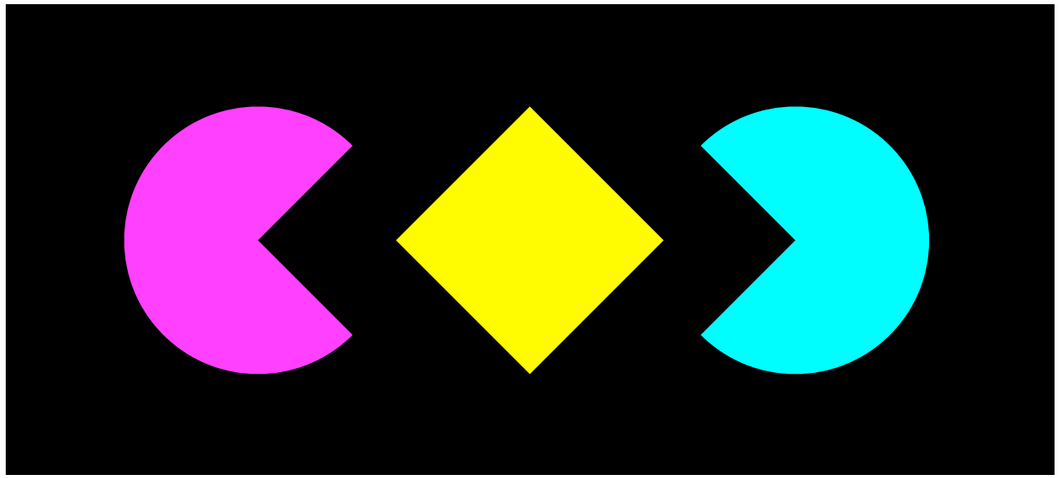


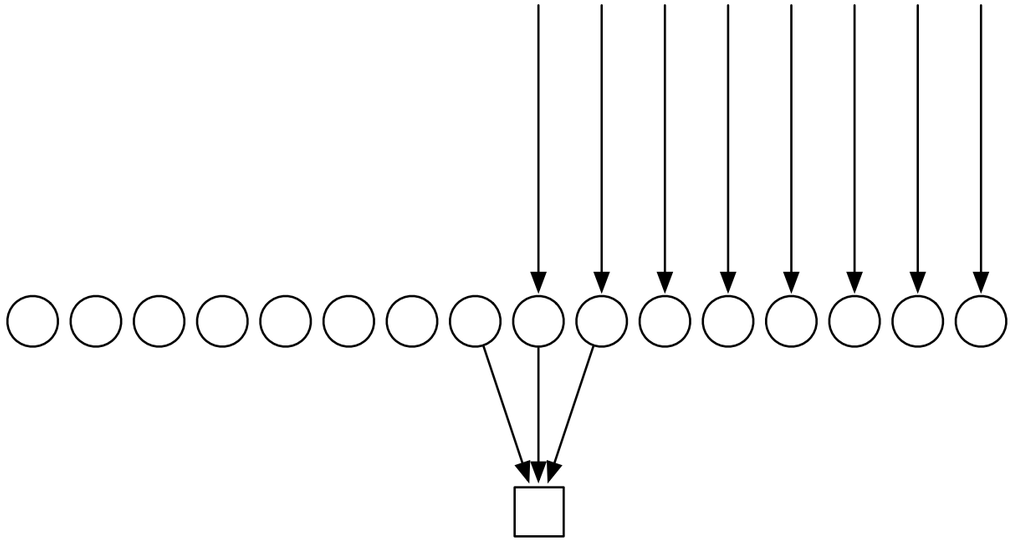
the notion of PARENT
MUST merge with the notion of actor
parent is the next interactor of the hierarchy
in the direction of an actor

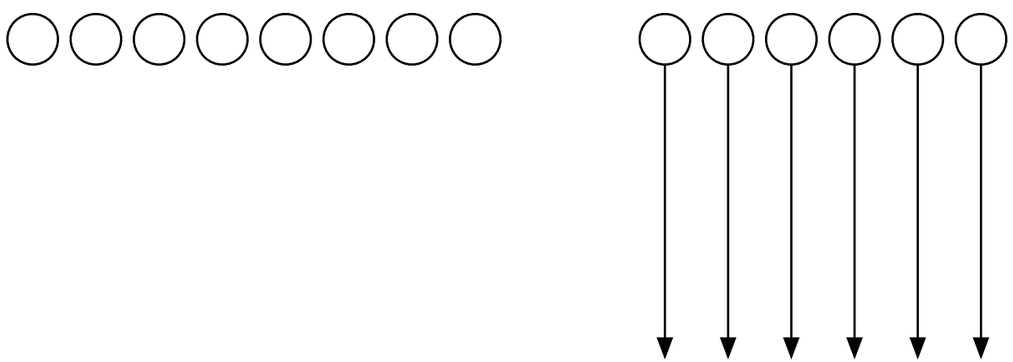
button interactor









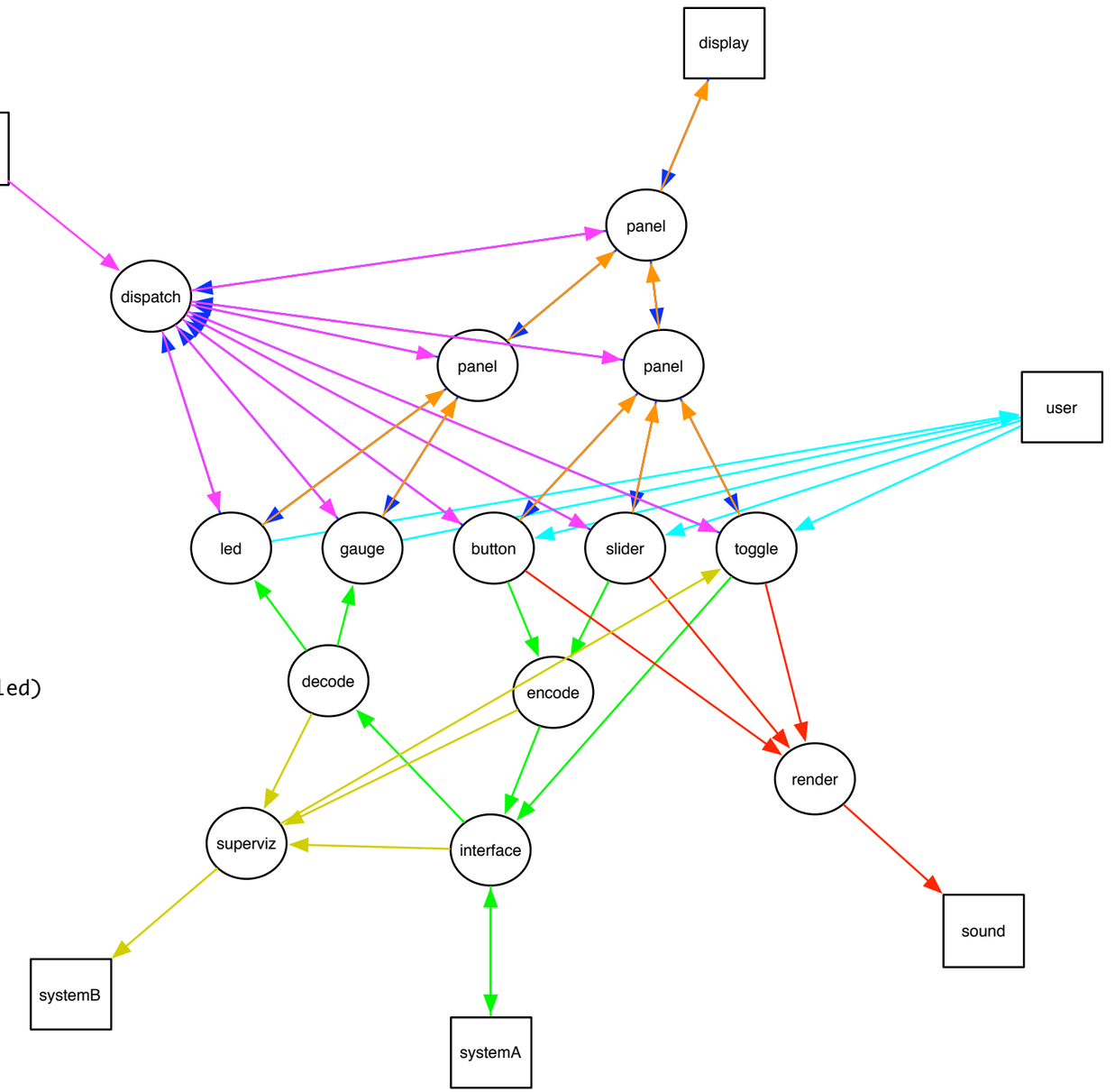


Multi composition example

Example multimodal

modalités en haut de chaque hierarchie

```
Led interactor:
  status : Boolean flow from system to user
  mousePos : MousePosition flow from mouse
  position : Layout flow from screen to mouse
  display : Svg flow to screen
  theme : Theme flow from screen
  focused : Boolean flow internal

  focused = (mousePos inside position)

  display =
    <circle
      fill=(if status then theme.highlight else theme.disabled)
      stroke = (if focused then theme.focus else none)
      x=position.x
      y=position.y
    />
```



layout info

mouse info

display info

systemA info

sound info

systemB info

user info

Check that :

(signal to user) <=> ((signal to sound) U (signal to screen))

signal from user <=> signal from mouse

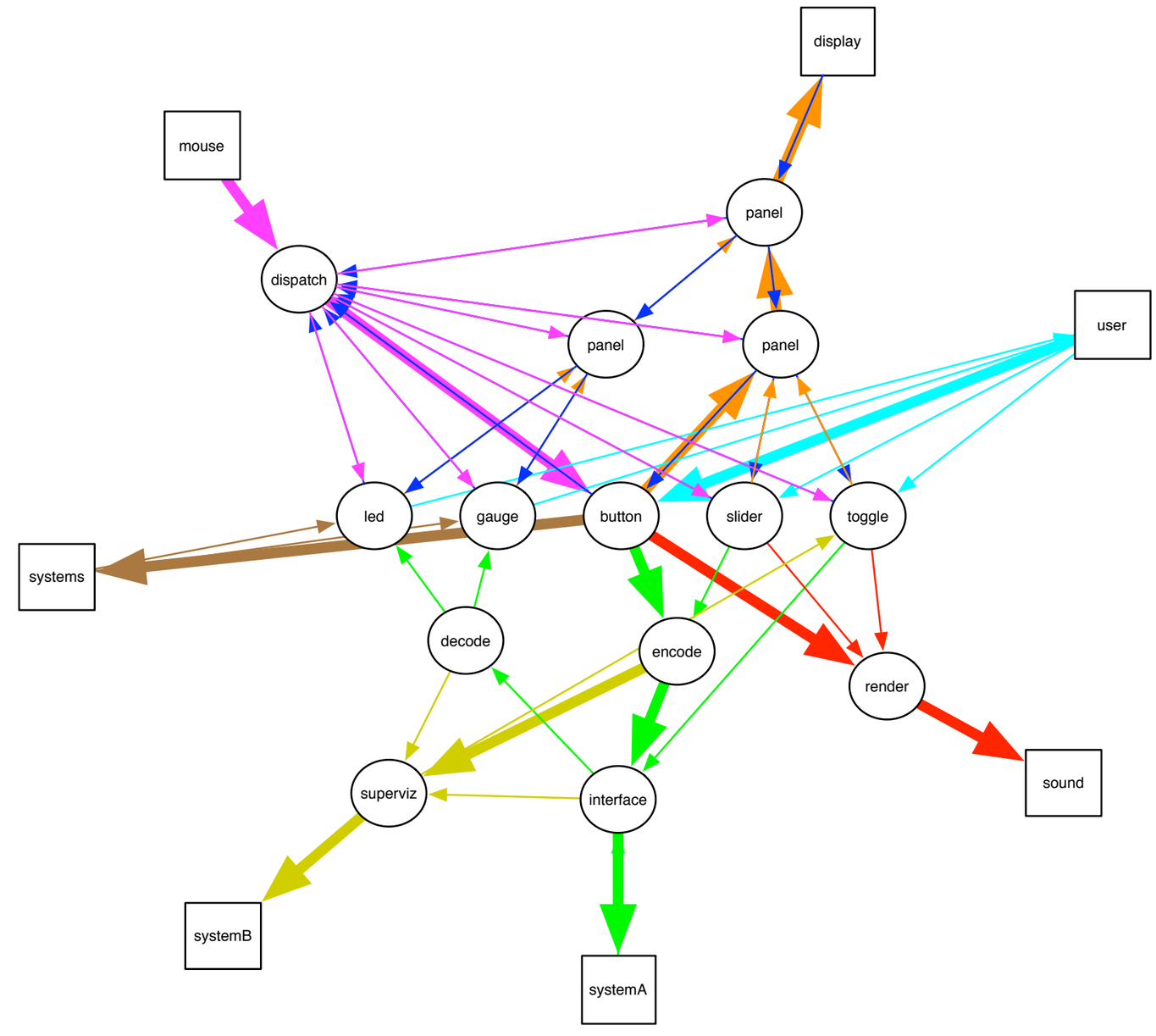
Multi composition example

Example multimodal

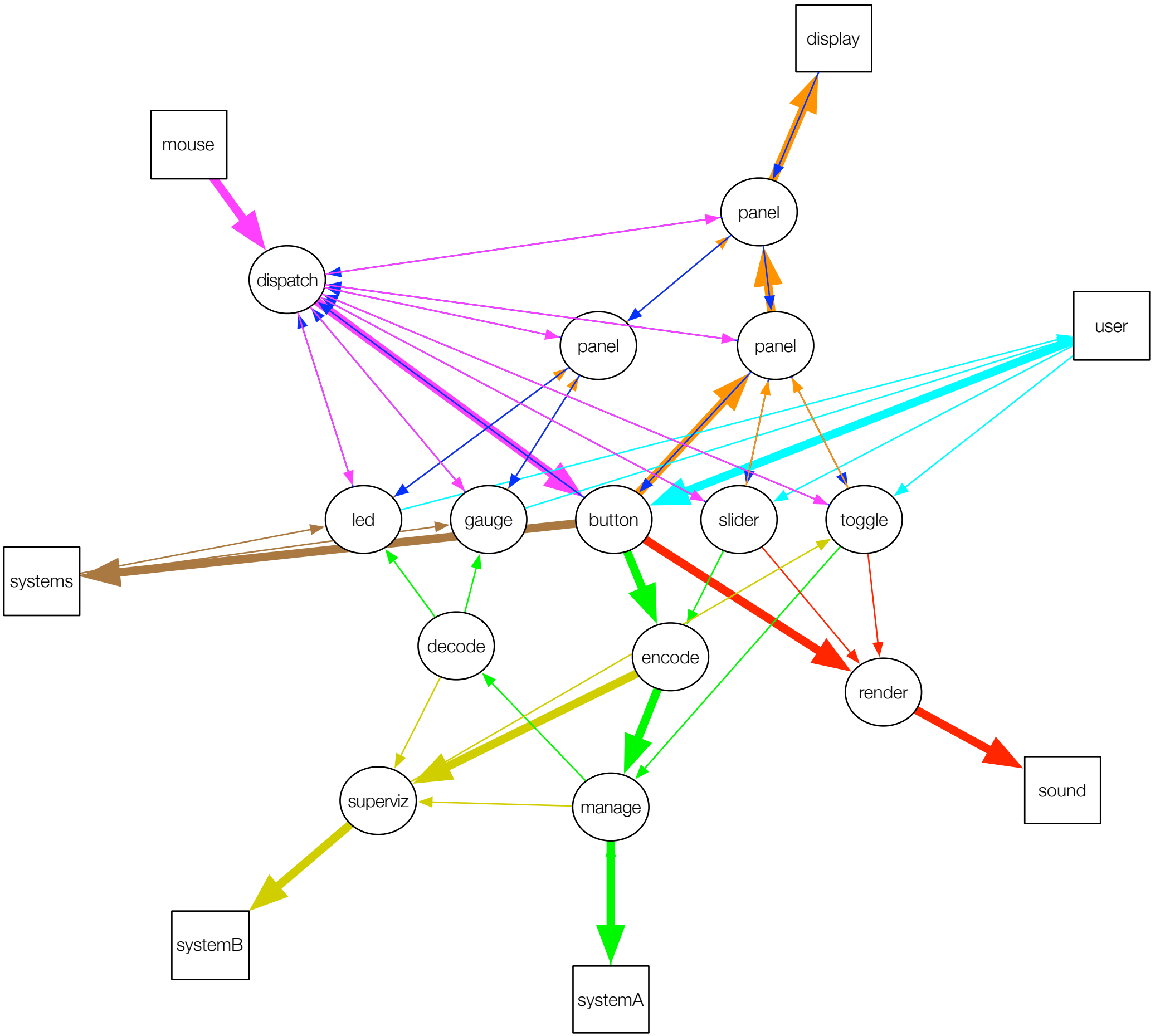
modalités en haut de chaque hierarchie

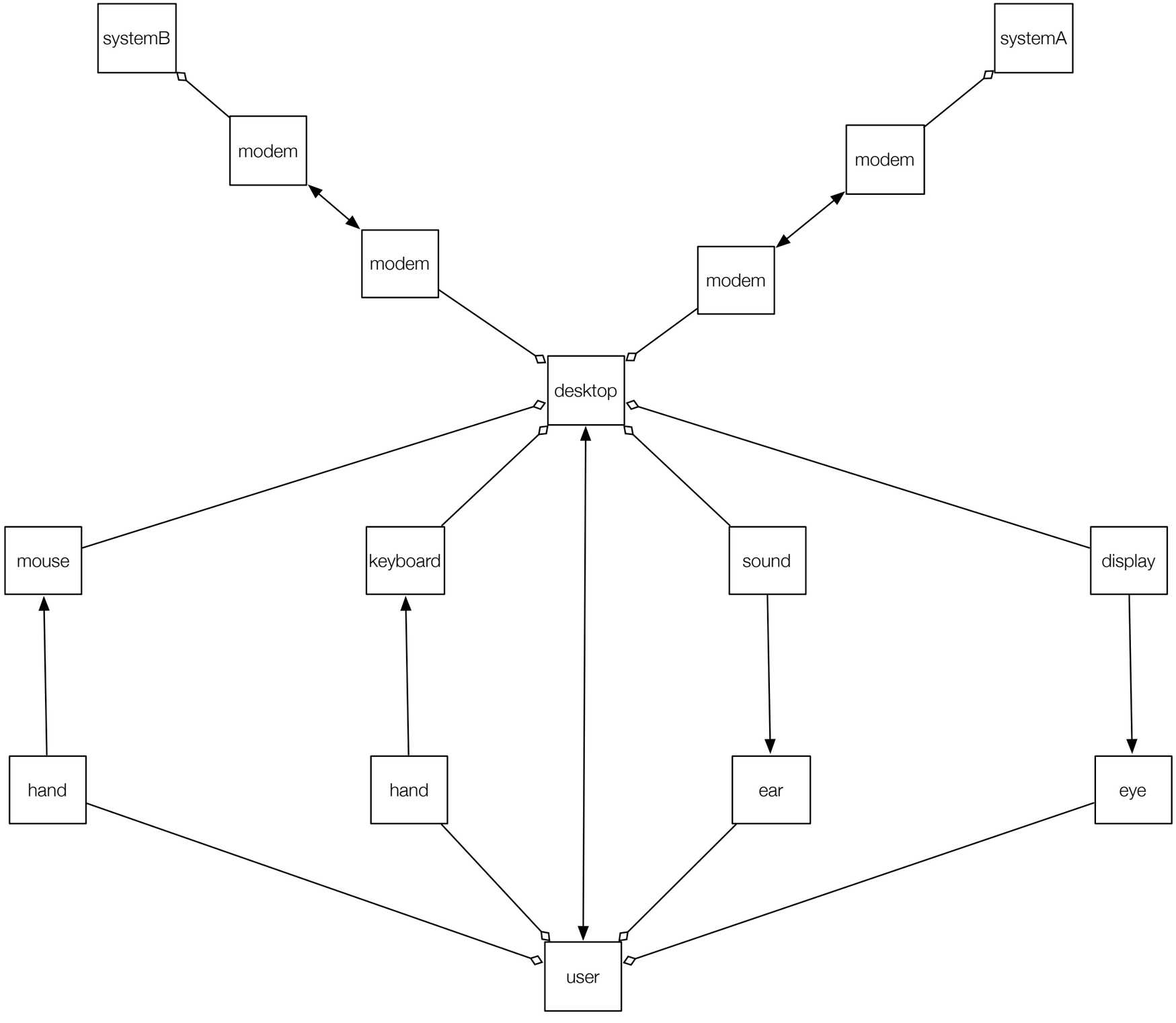
- layout info
- mouse info
- display info
- systemA info
- sound info
- systemB info
- user info

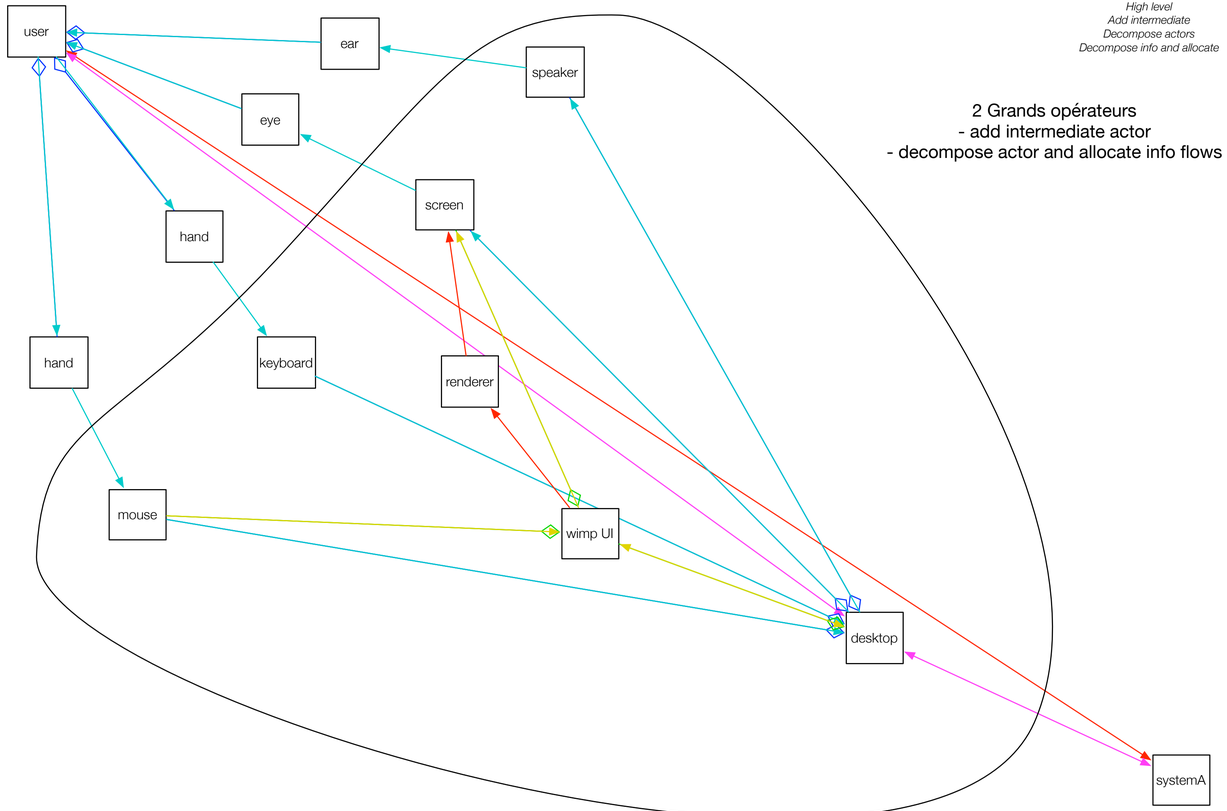
```
Led interactor:
  status : Boolean flow from system to user
  mousePos : MousePosition flow from mouse
  position : Layout flow from screen to mouse
  display : Svg flow to screen
  theme : Theme flow from screen
  focused : Boolean flow internal

  focused = (mousePos inside position)

  display =
    <circle
      fill=(if status then theme.highlight else theme.disabled)
      stroke = (if focused then theme.focus else none)
      x=position.x
      y=position.y
    />
```



Check that :
(signal to user) <=> ((signal to sound) U (signal to screen))
signal from user <=> signal from mouse
(signal to systems) <=> signal to systemA U signal to systemB


display
mouse
panel
dispatch
panel
panel
user
led
gauge
button
slider
toggle
systems
decode
encode
render
superviz
manage
sound
systemB
systemA

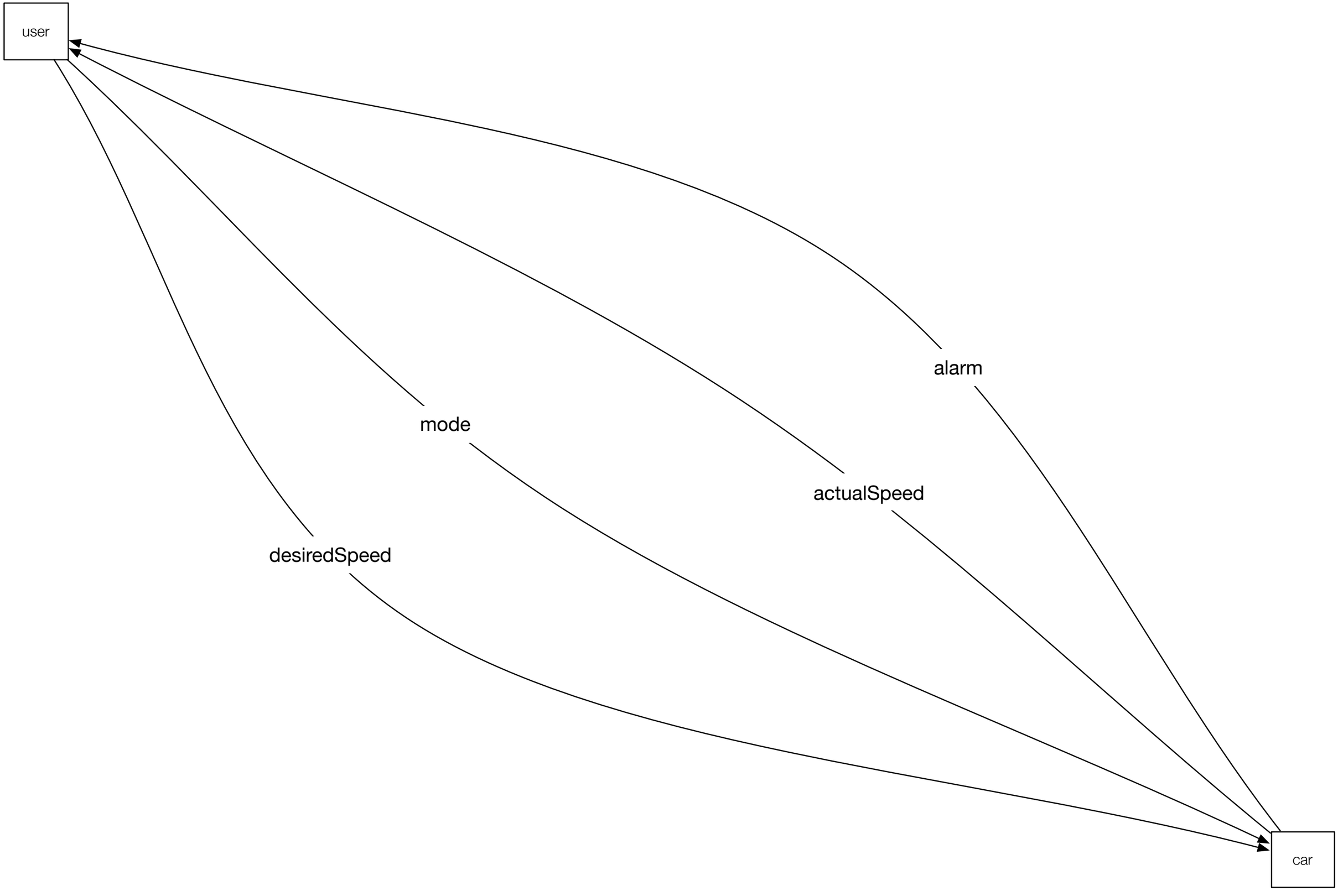

systemB
systemA
modem
modem
modem
modem
desktop
mouse
keyboard
sound
display
hand
hand
ear
eye
user



High level
Add intermediate
Decompose actors
Decompose info and allocate

2 Grands opérateurs
- add intermediate actor
- decompose actor and allocate info flows

desktop INTERMEDIATE BETWEEN (user, systemA)
desktop MADE OFF (screen, keyboard, speaker, mouse)


user
alarm
mode
actualSpeed
desiredSpeed
car

user

```
{
  mode:Mode flow from user to car,
  desiredSpeed:Speed flow from user to car,
  actualSpeed:Speed flow from car to user,
  status:Status flow from car to user
}
```

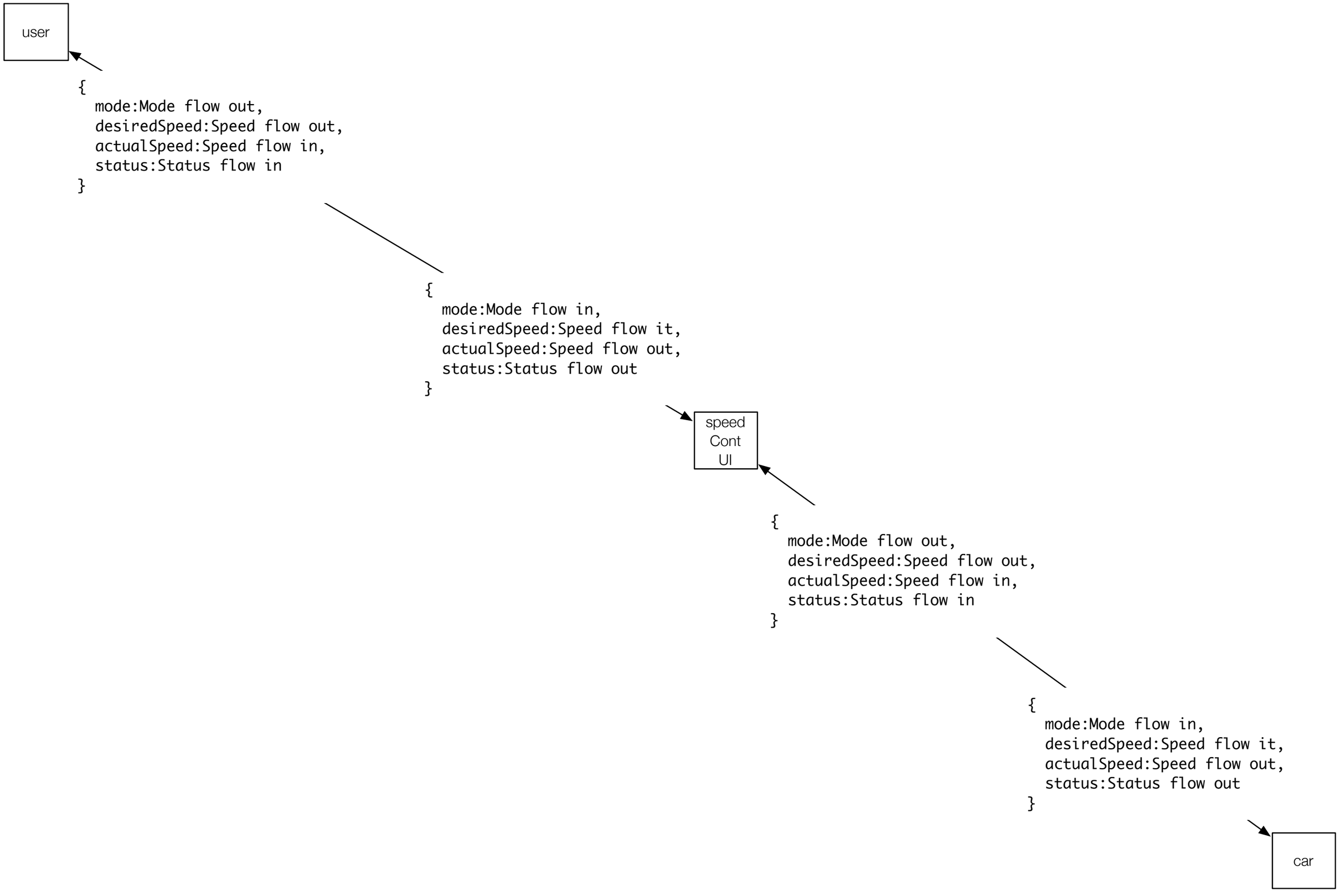
car

user

```
{
  mode:Mode flow out,
  desiredSpeed:Speed flow out,
  actualSpeed:Speed flow in,
  status:Status flow in
}
```

```
{
  mode:Mode flow in,
  desiredSpeed:Speed flow it,
  actualSpeed:Speed flow out,
  status:Status flow out
}
```

car


user
{
mode:Mode flow out,
desiredSpeed:Speed flow out,
actualSpeed:Speed flow in,
status:Status flow in
}
{
mode:Mode flow in,
desiredSpeed:Speed flow it,
actualSpeed:Speed flow out,
status:Status flow out
}
speed
Cont
UI
{
mode:Mode flow out,
desiredSpeed:Speed flow out,
actualSpeed:Speed flow in,
status:Status flow in
}
{
mode:Mode flow in,
desiredSpeed:Speed flow it,
actualSpeed:Speed flow out,
status:Status flow out
}
car

user

```
{
  mode:Mode flow out,
  desiredSpeed:Speed flow out,
  actualSpeed:Speed flow in,
  status:Status flow in
}
```

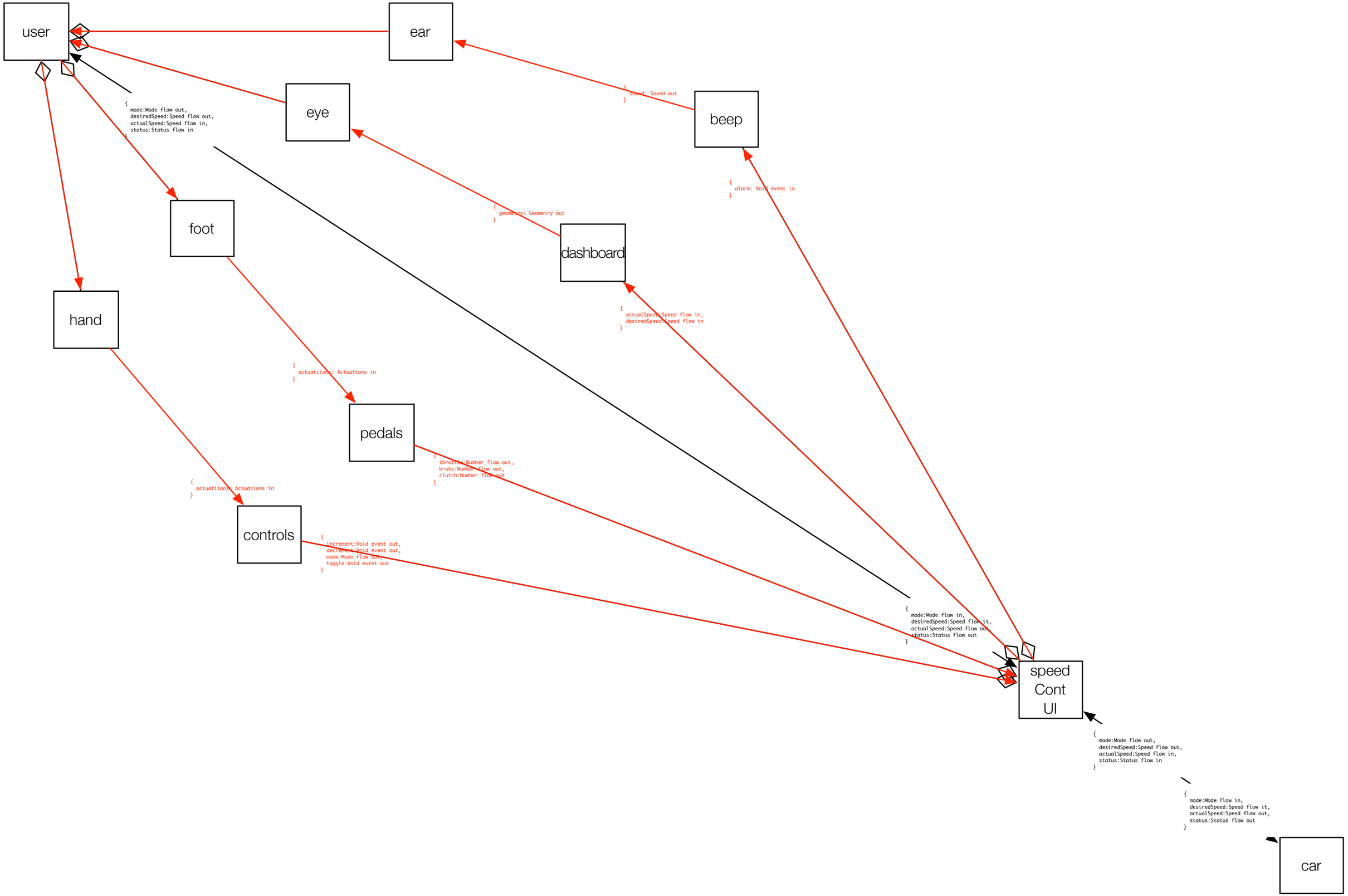
```
{
  mode:Mode flow in,
  desiredSpeed:Speed flow it,
  actualSpeed:Speed flow out,
  status:Status flow out
}
```

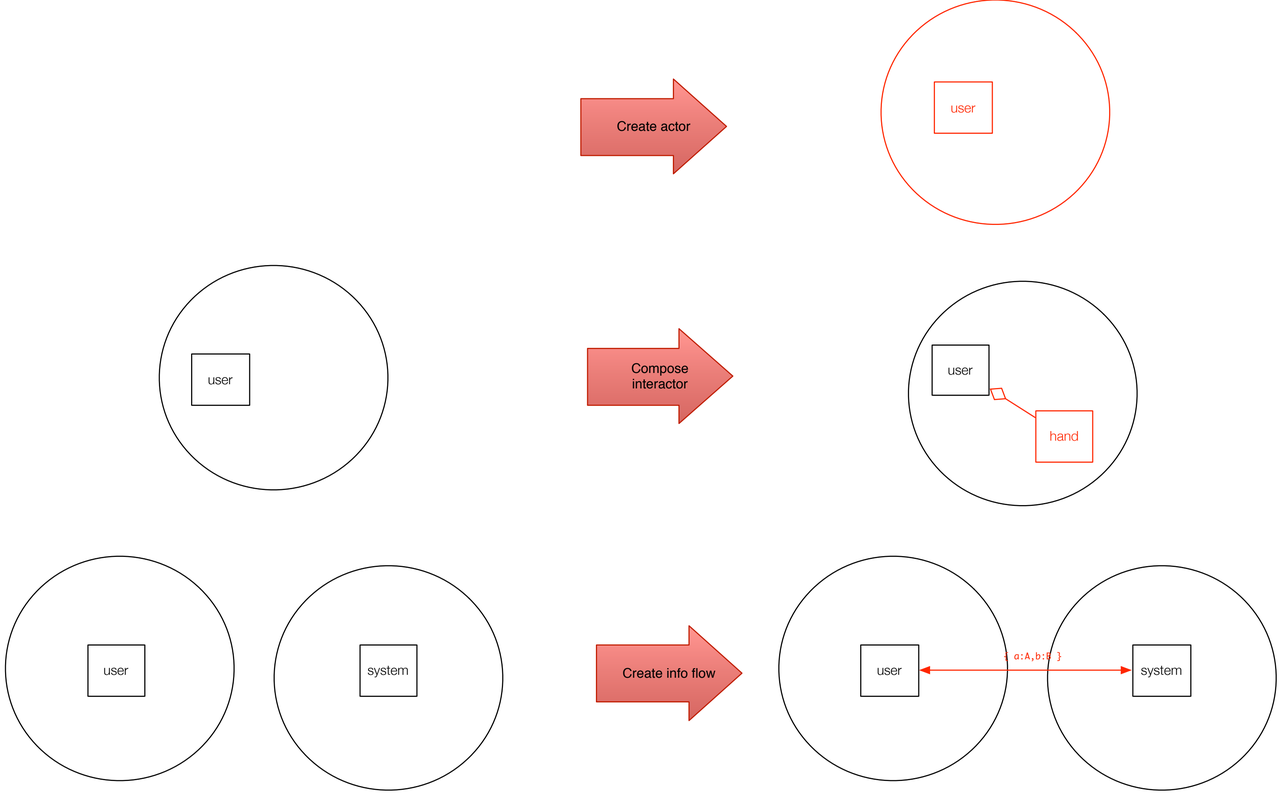
speed
Cont
UI

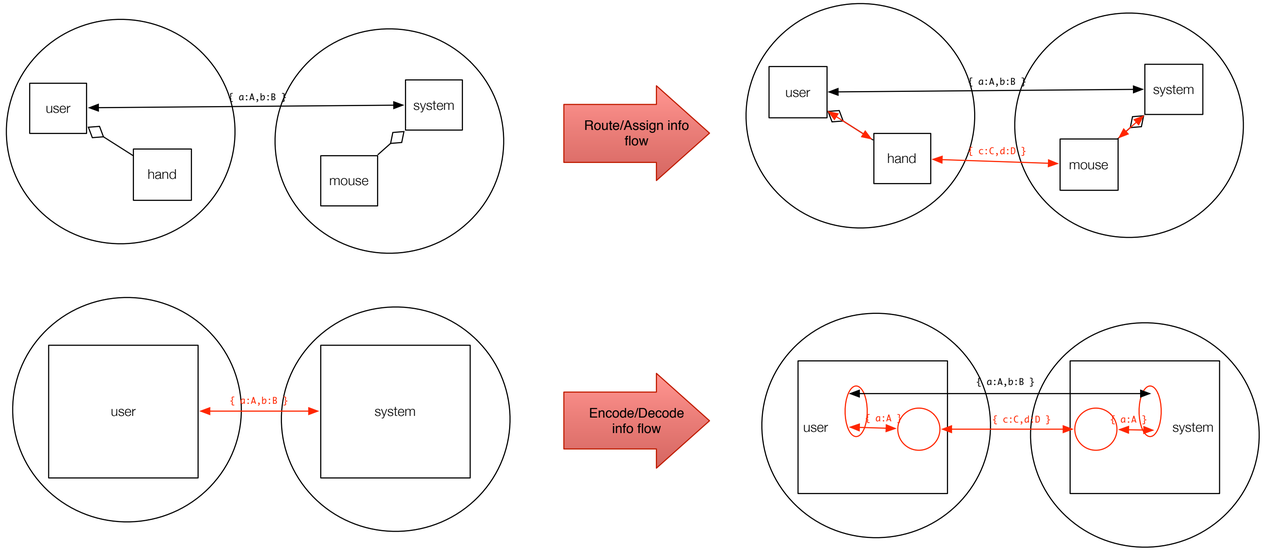
```
{
  mode:Mode flow out,
  desiredSpeed:Speed flow out,
  actualSpeed:Speed flow in,
  status:Status flow in
}
```

```
{
  mode:Mode flow in,
  desiredSpeed:Speed flow it,
  actualSpeed:Speed flow out,
  status:Status flow out
}
```

car


user
ear
eye
beep
foot
dashboard
hand
pedals
controls
speed
Cont
Ul
car
{
mode:Mode flow out,
desiredSpeed:Speed flow out,
actualSpeed:Speed flow in,
status:Status flow in
}
{
sound: Sound out
}
{
alarm: Void event in
}
{
geometry: Geometry out
}
{
actualSpeed:Speed flow in,
desiredSpeed:Speed flow in
}
{
actuations: Actuations in
}
{
throttle:Number flow out,
brake:Number flow out,
clutch:Number flow out
}
{
actuations: Actuations in
}
{
increment:Void event out,
decrement:Void event out,
mode:Mode flow out,
toggle:Void event out
}
{
mode:Mode flow in,
desiredSpeed:Speed flow it,
actualSpeed:Speed flow out,
status:Status flow out
}
{
mode:Mode flow out,
desiredSpeed:Speed flow out,
actualSpeed:Speed flow in,
status:Status flow in
}
{
mode:Mode flow in,
desiredSpeed:Speed flow it,
actualSpeed:Speed flow out,
status:Status flow out
}





An actor IS a (super?)interactor

A behavior IS an (sub)interactor !

NO INTERACTORS, ACTORS, BEHAVIOR : ONLY INTERACTION

WE DESCRIBE THE SET OF POSSIBLE INTERACTIONS FOR AN ENTITY

And their relationship

exemple:
inc/dec to set speed behavior
=
an interactor :
inputs from driver : inc dec
inputs from parent : limits
output to parent : value

Data
Computation
Interaction

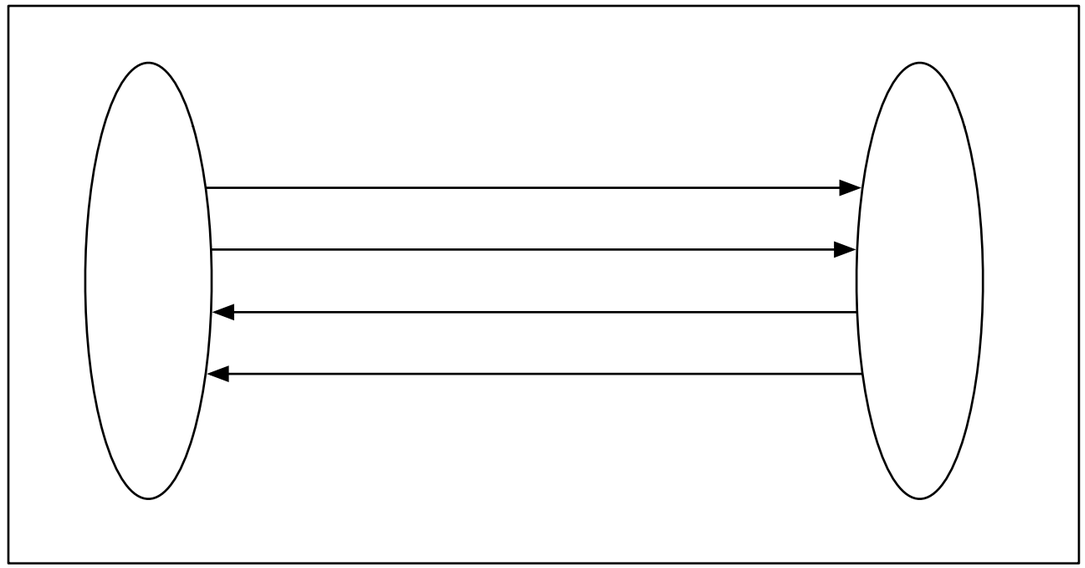
we have Languages to describe data (e.g. : C, JSON, XML)

we have Languages to describe computation (e.g. : C, JS)

we don't have Languages to describe interaction ?????

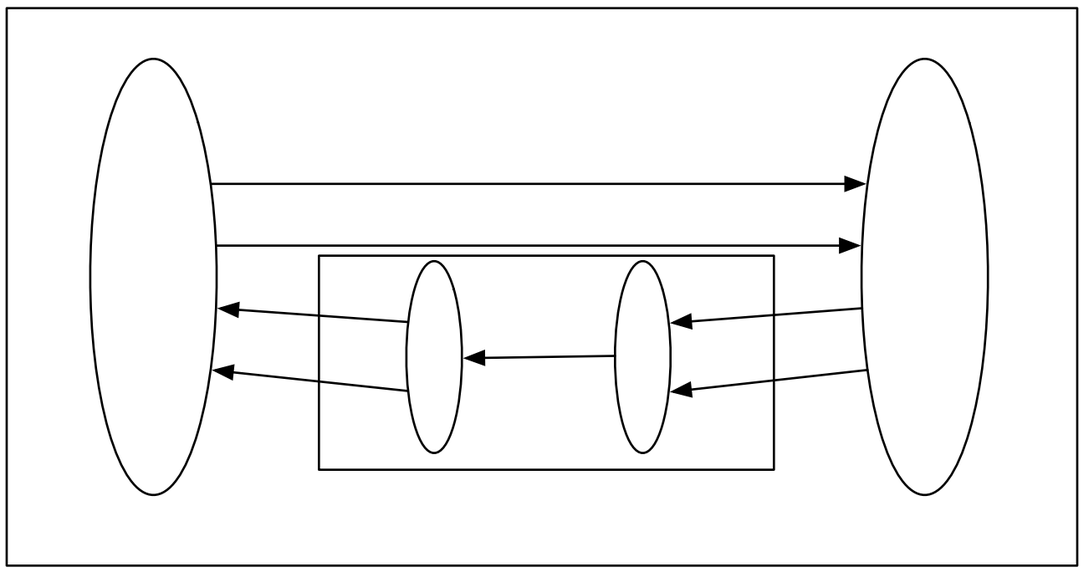
Set of possible interactions and their relationship

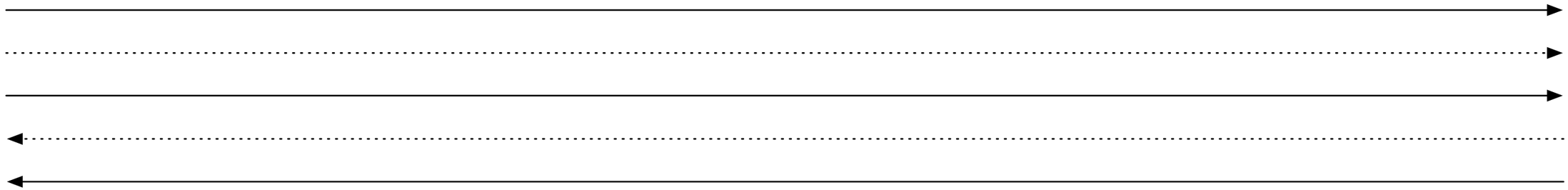
```
Human interaction:
  hands : Hands interaction
  foots : Foots interaction
  eyes : Eyes interaction
  ears : Ears interaction
```



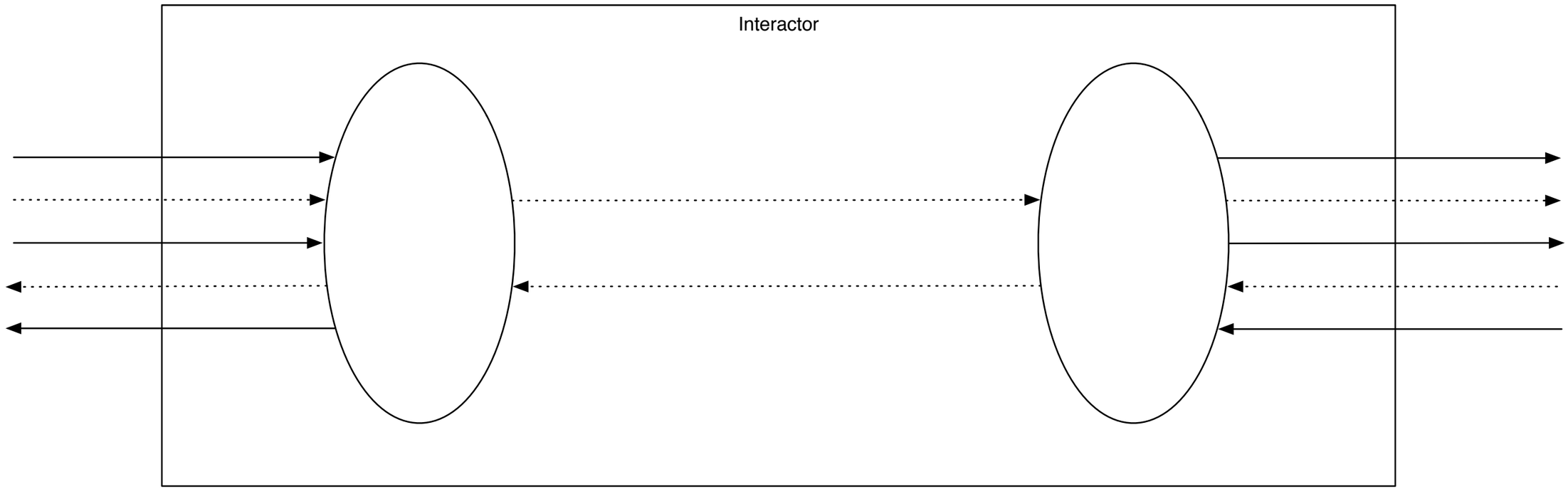
```
SpeedController interaction:
  controls : Controls interaction
  pedals : Pedals interaction
  dashboard : Dashboard interaction
```

```
Controls interaction:
  increment : Void event in
  decrement : Void event in
  mode : Mode flow in
```





=

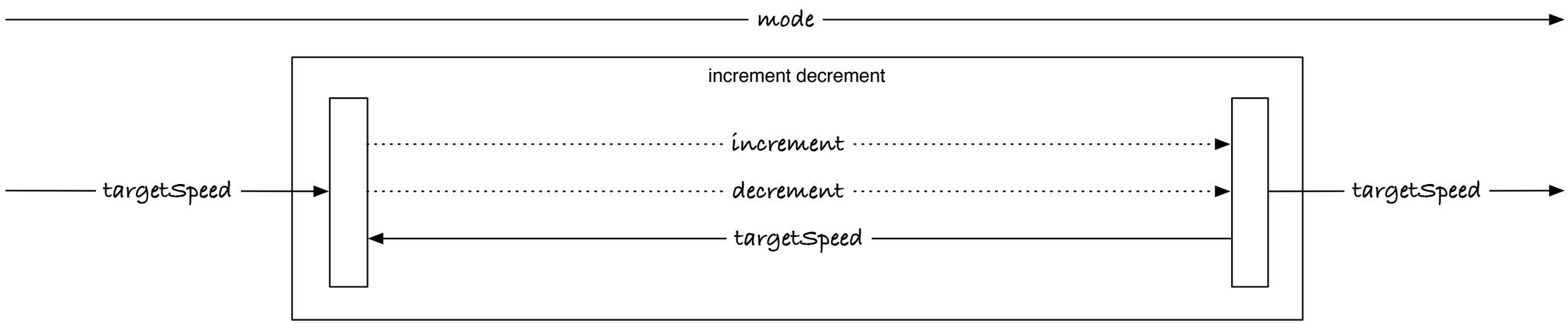

Interactor

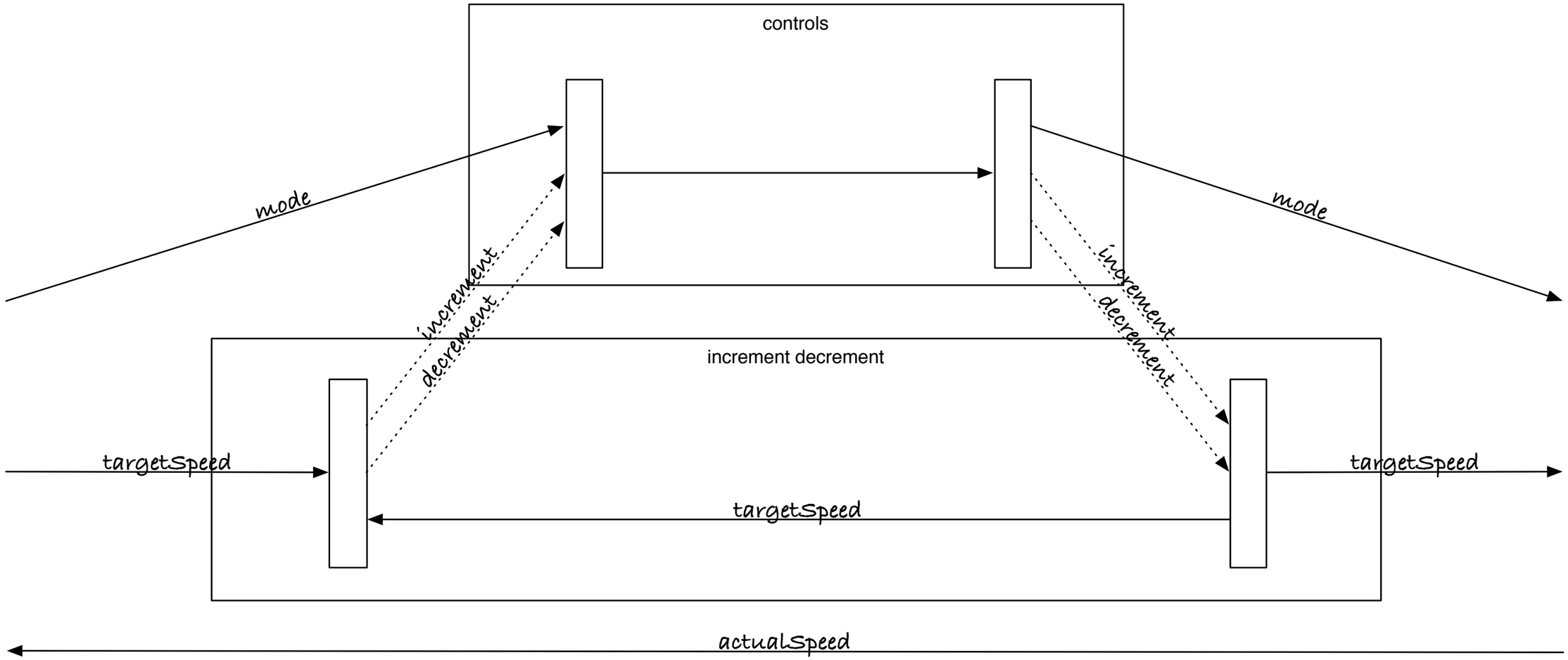
mode

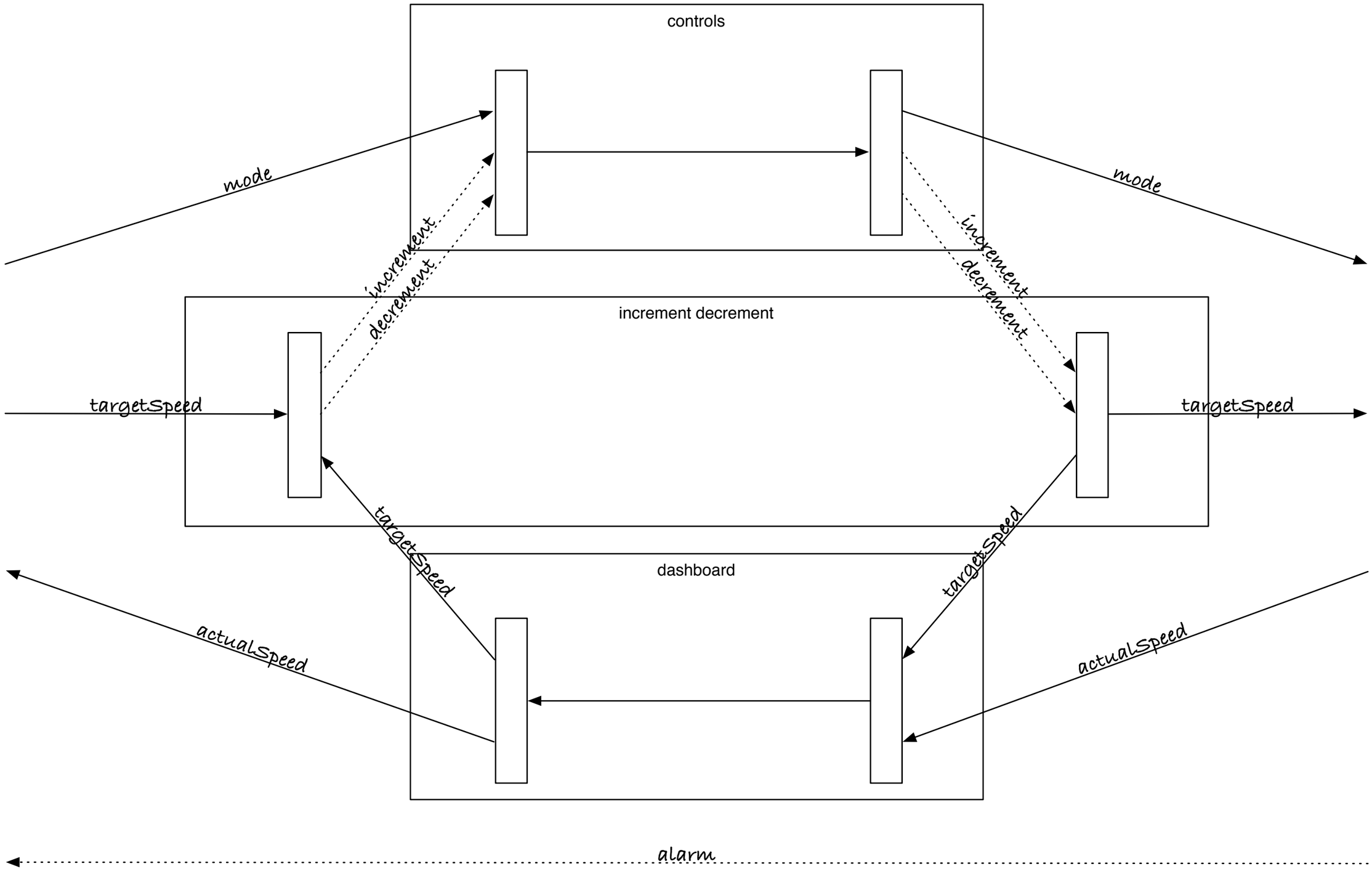
targetSpeed

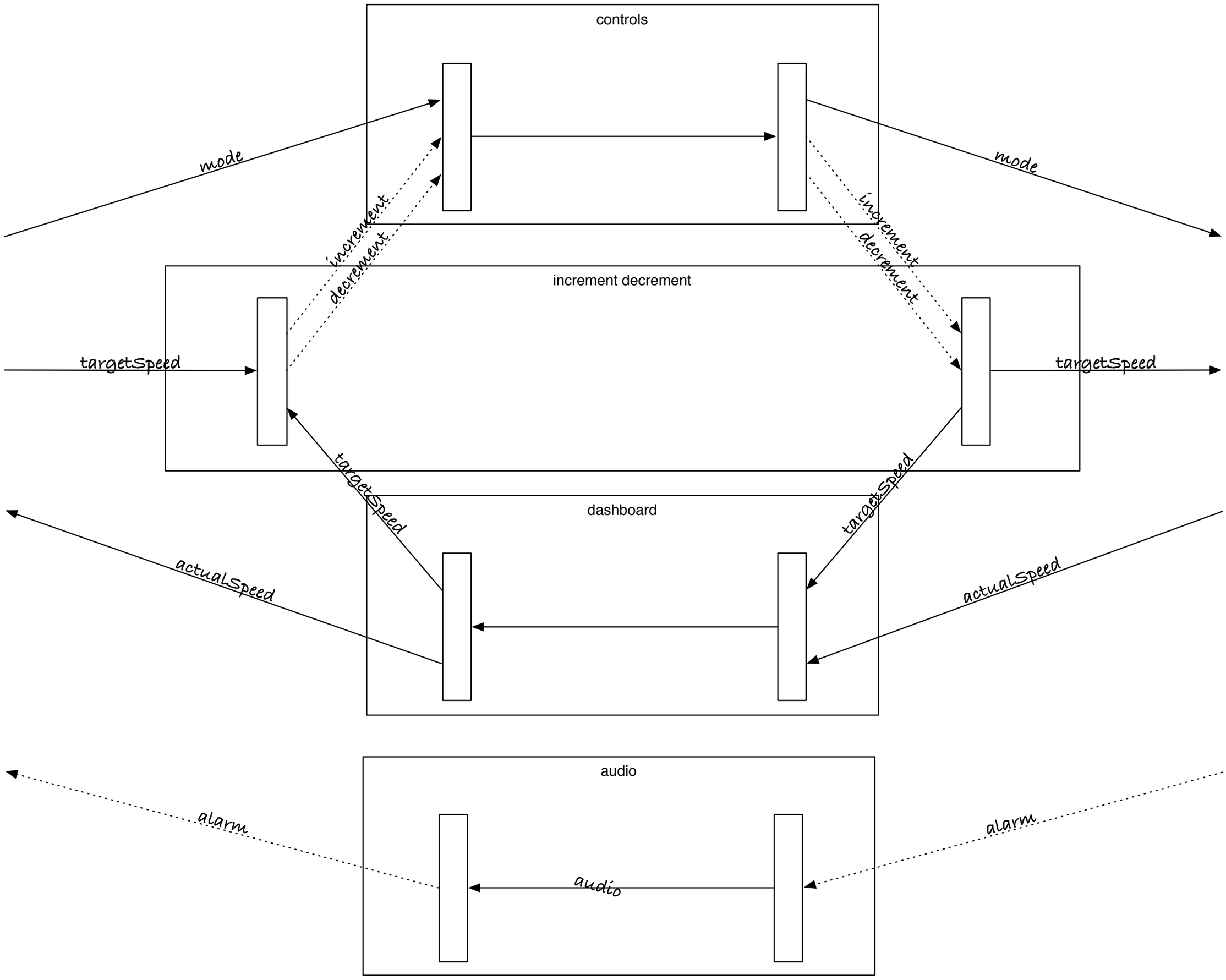
actualSpeed

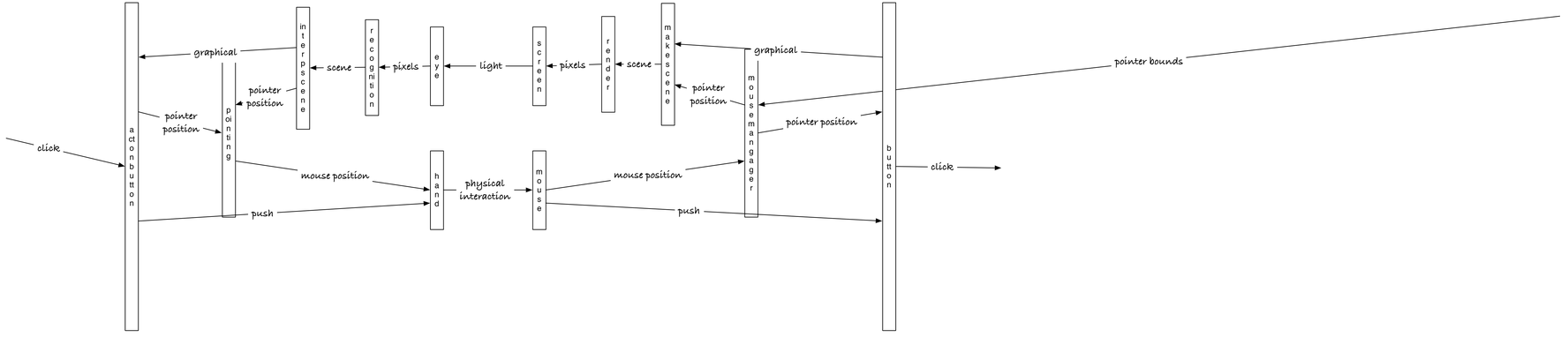
alarm

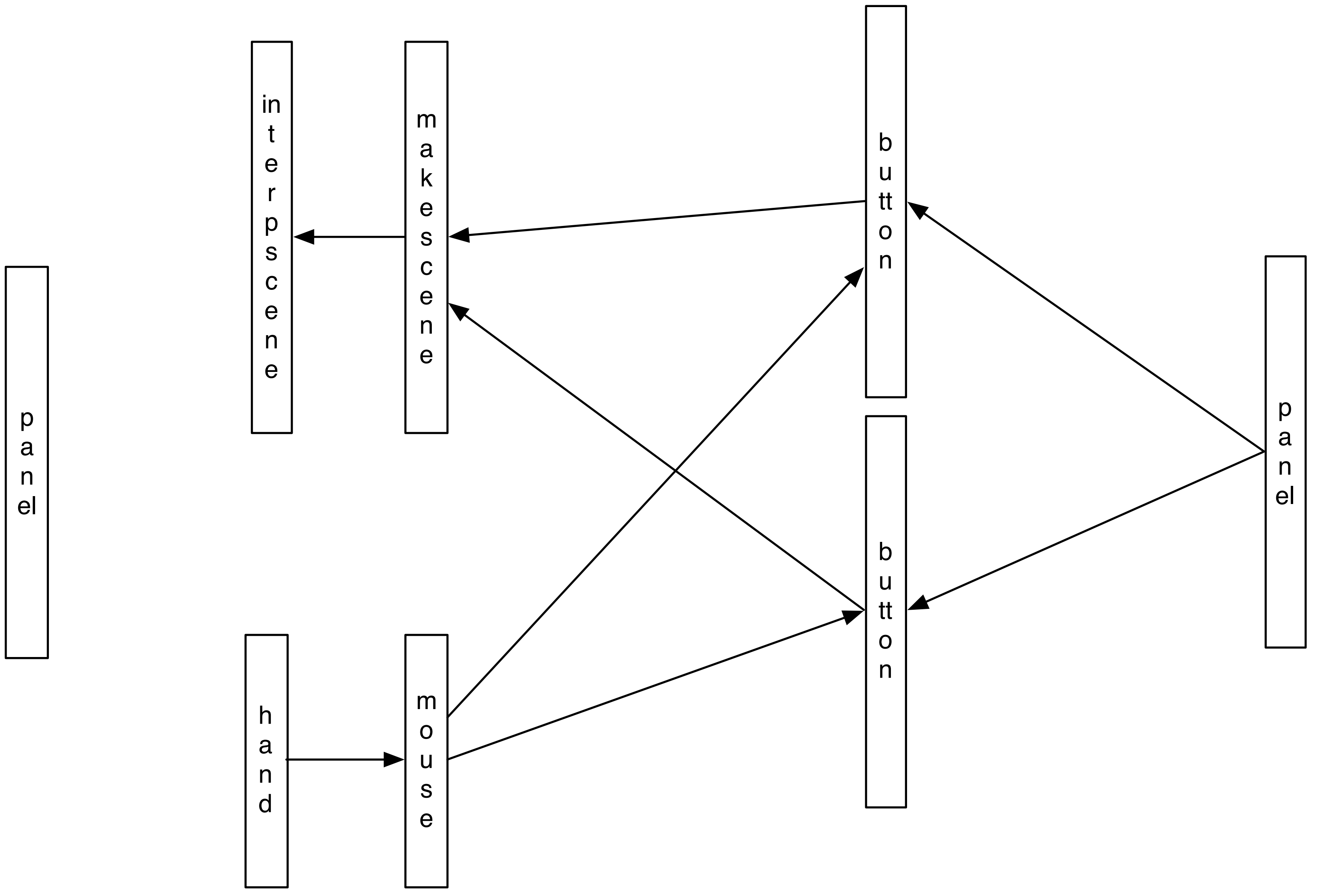

mode
increment decrement
increment
targetSpeed
decrement
targetSpeed
targetSpeed
actualSpeed
alarm


controls
mode
increment
decrement
increment
decrement
mode
increment decrement
targetSpeed
targetSpeed
targetSpeed
actualSpeed
alarm

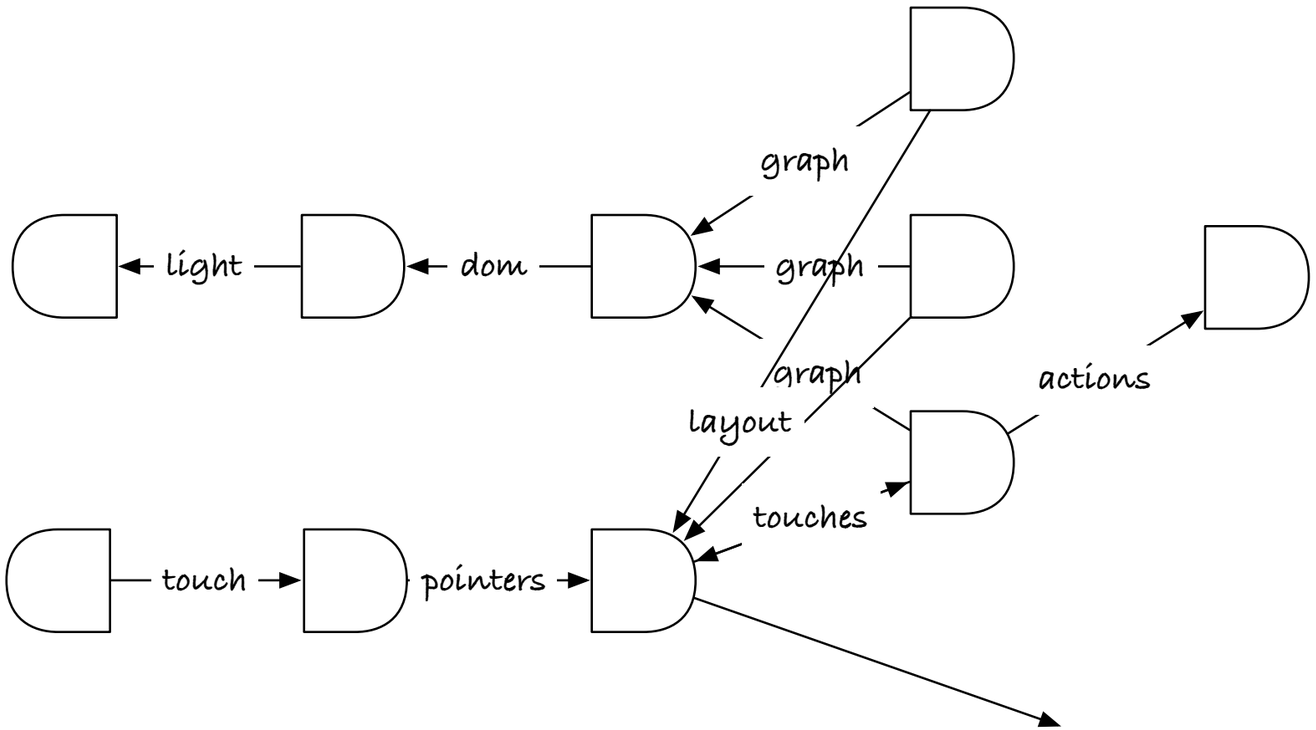

controls
mode
mode
increment
decrement
increment
decrement
increment decrement
targetSpeed
targetSpeed
targetSpeed
targetSpeed
dashboard
actualSpeed
actualSpeed
alarm

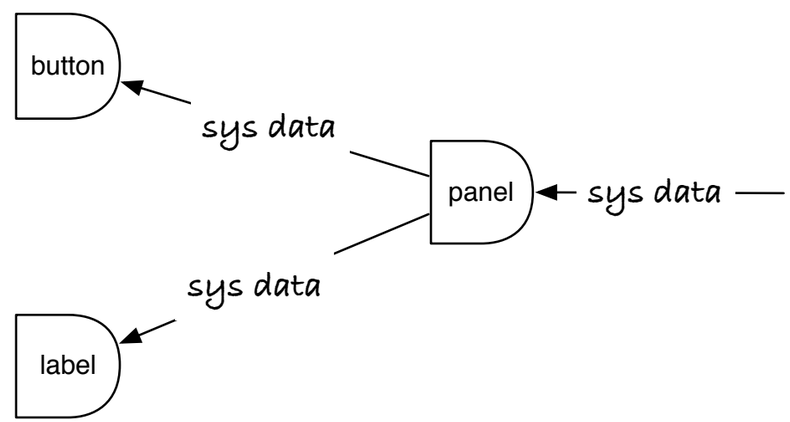

controls
mode
increment
decrement
mode
increment
decrement
increment decrement
targetSpeed
targetSpeed
targetSpeed
targetSpeed
dashboard
actualSpeed
actualSpeed
audio
alarm
audio
alarm

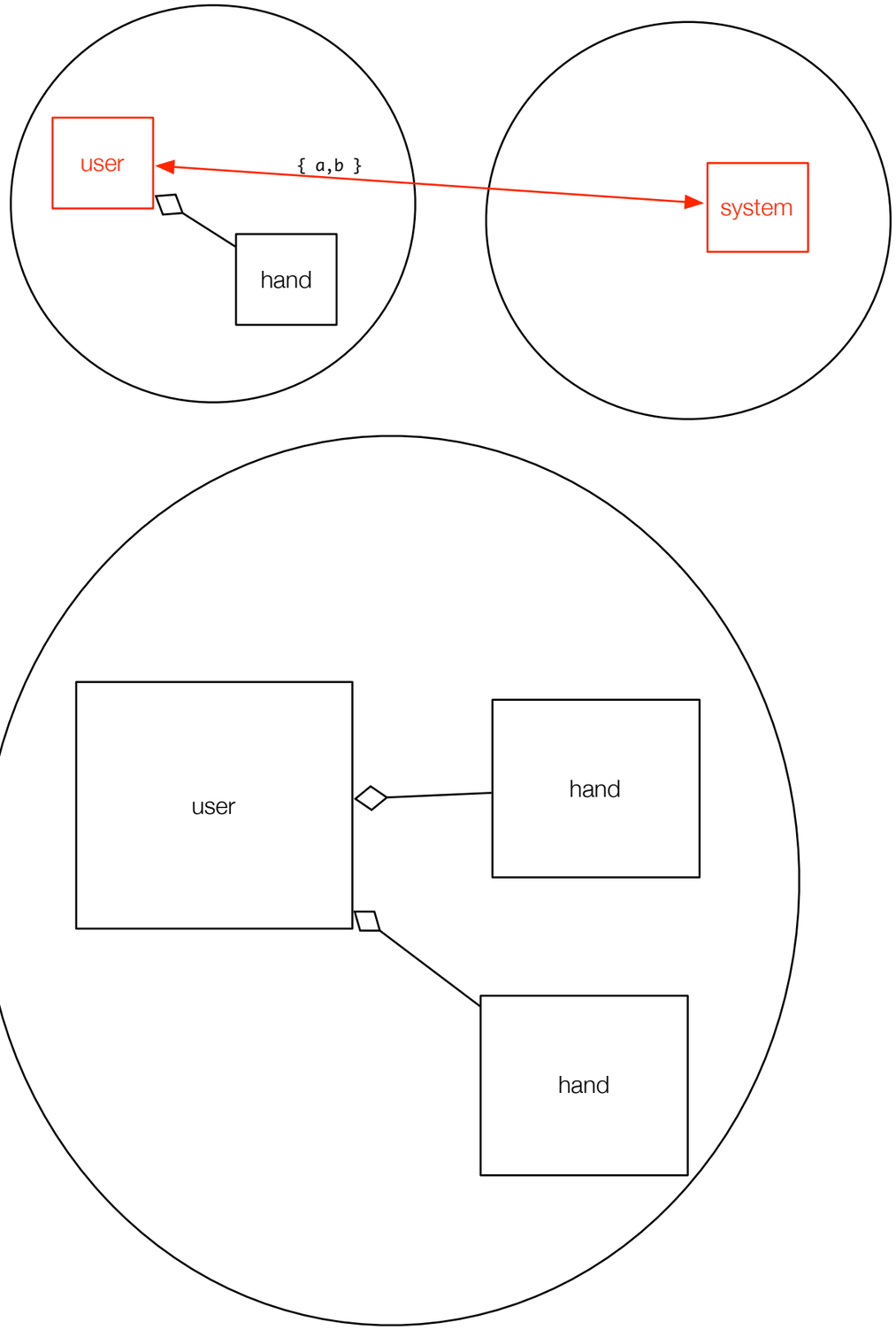

click
actionbutton
pointer position
pointing
graphical
interpscene
scene
recognition
pixels
eye
light
screen
pixels
render
scene
makescene
graphical
pointer position
mousemanager
pointer position
pointer bounds
button
click
pointer position
mouse position
hand
physical interaction
mouse
mouse position
push
push



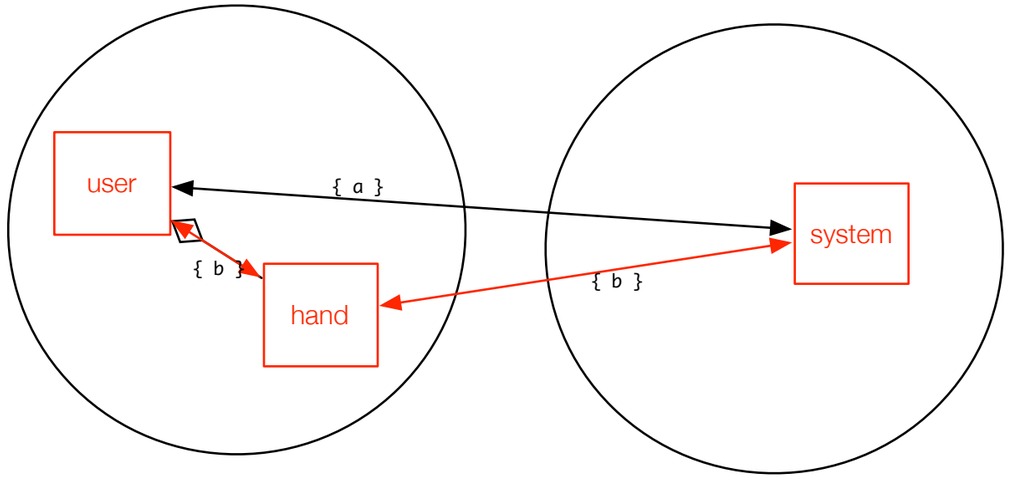
panel
interpscene
makescene
hand
mouse
button
button
panel


graph
light
dom
graph
graph
actions
layout
touches
touch
pointers


button
sys data
panel
sys data
sys data
label

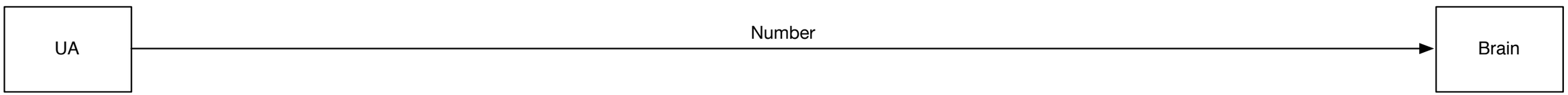


Assign info flow

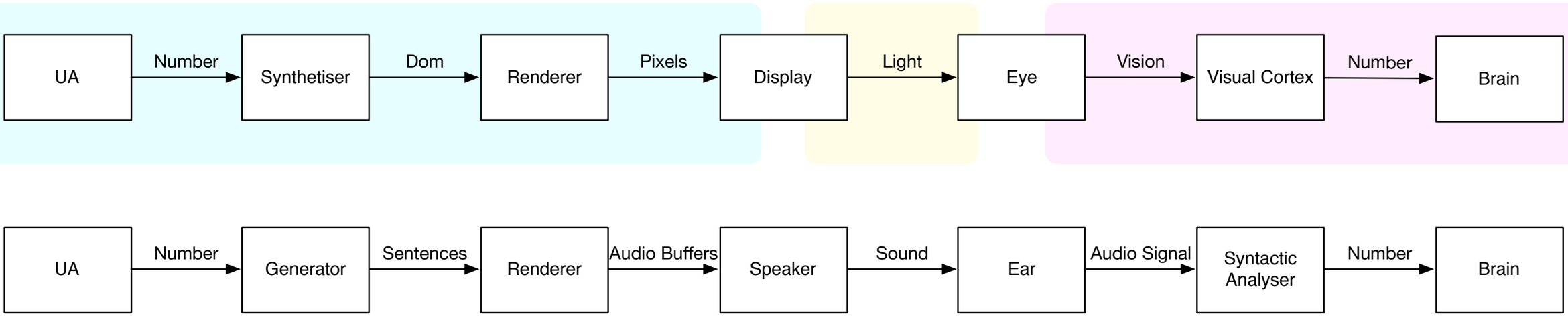


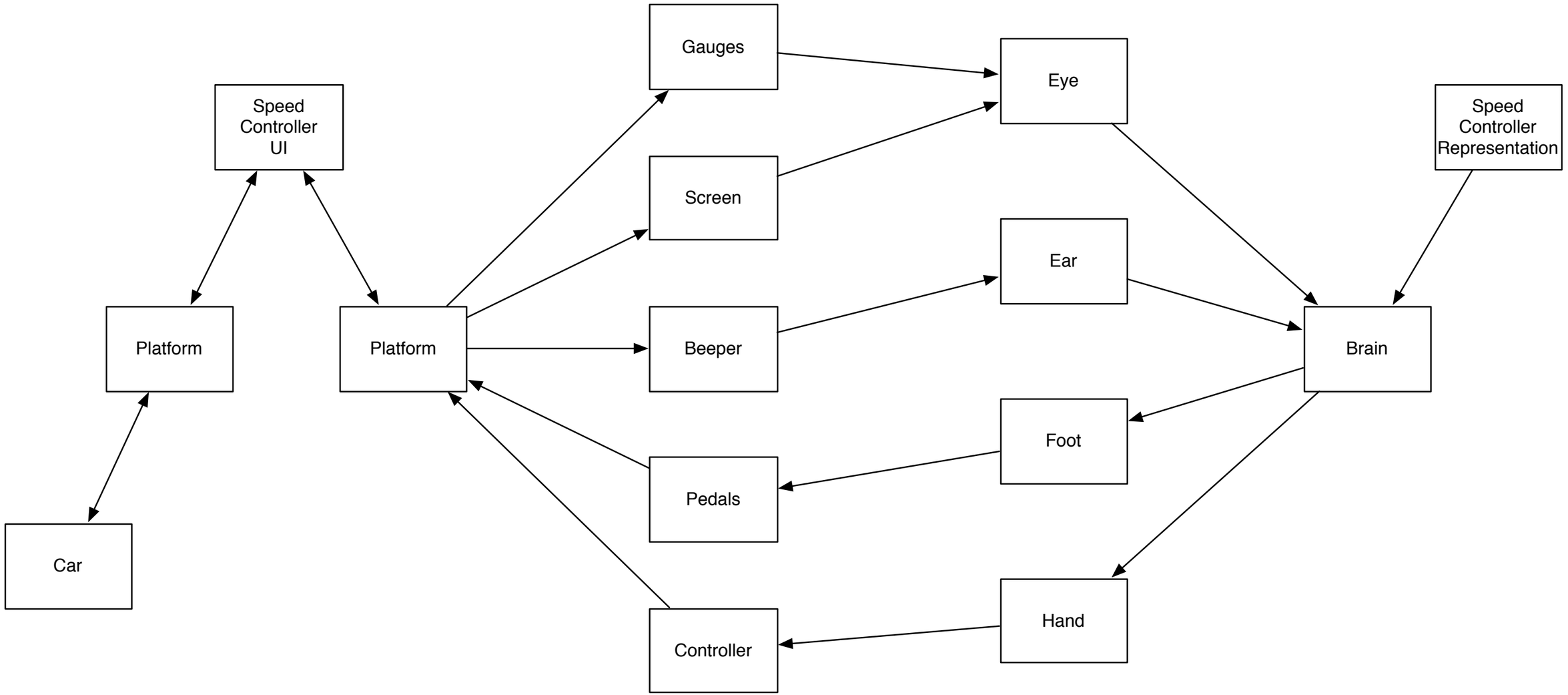
Kind of OSI layer model

Model checking context



Actual context

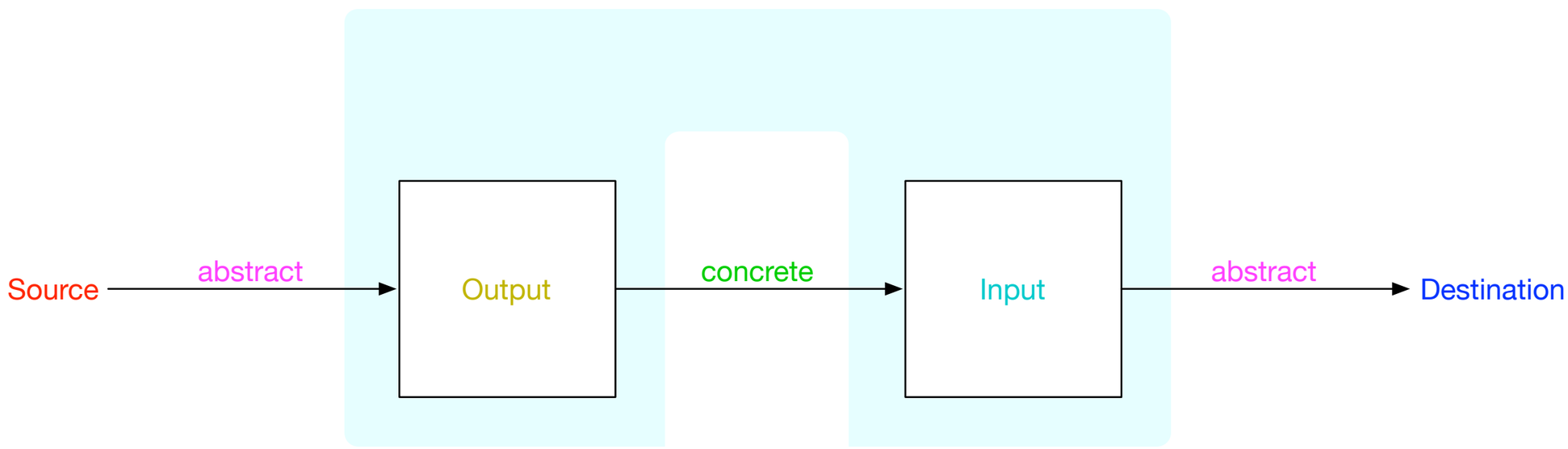


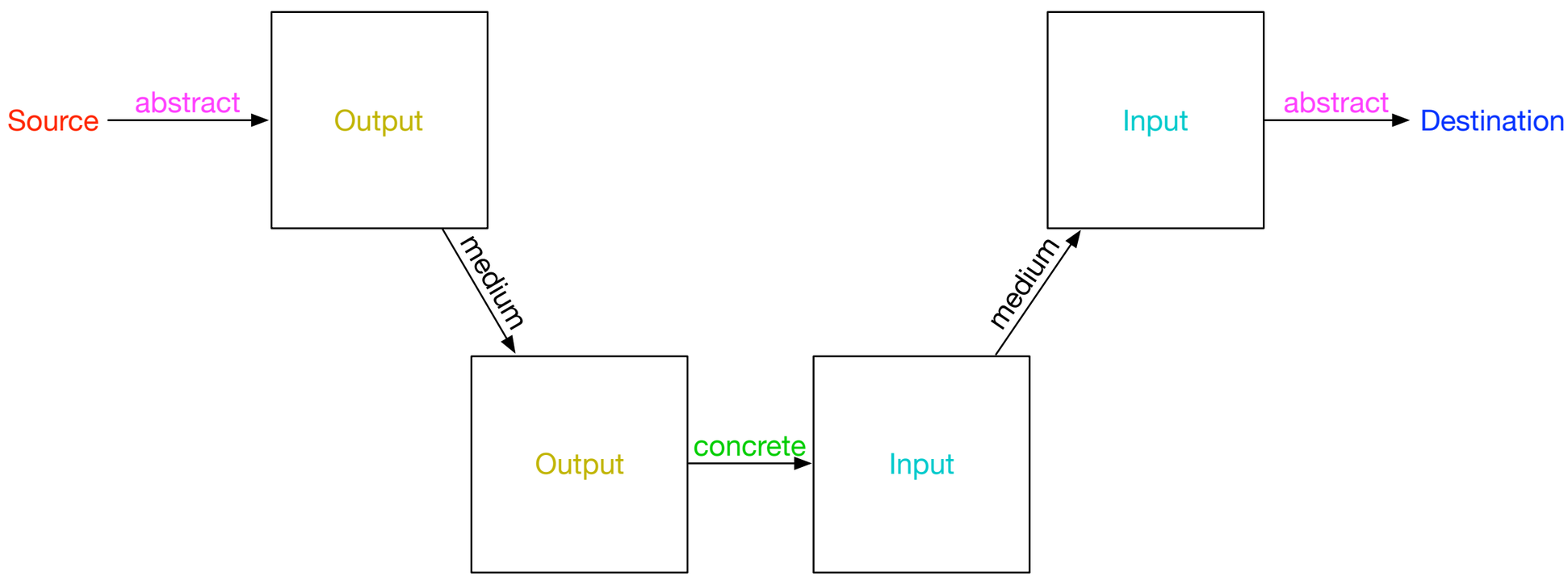

Gauges
Eye
Speed Controller UI
Speed Controller Representation
Screen
Ear
Platform
Platform
Beeper
Brain
Foot
Pedals
Car
Hand
Controller

What is an interactive system ?

A system that takes an abstract flow of information between a source and a destination, and transforms it into a concrete flow of information between an output and an input, mapping the output to the source, and the input to the destination. An important thing is that abstract and concrete are relative notions.

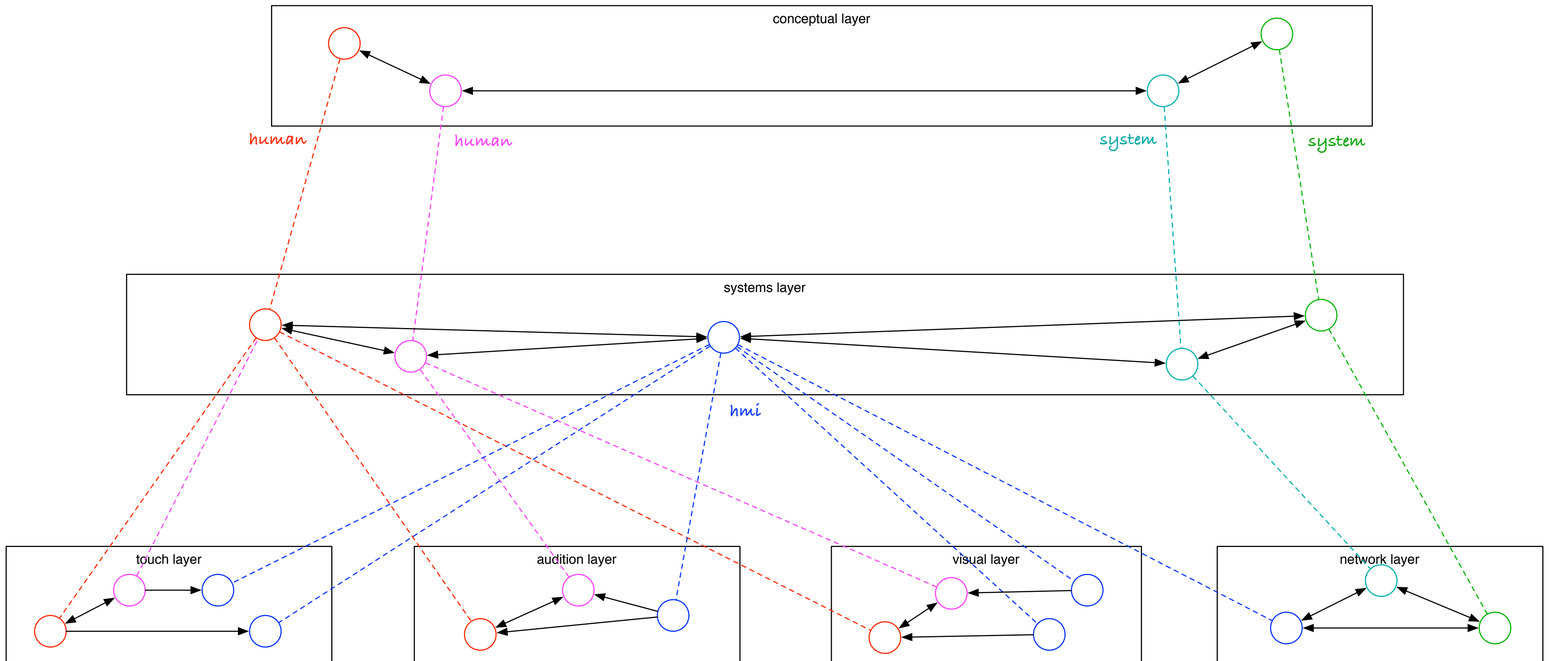
A generalization of the OSI model

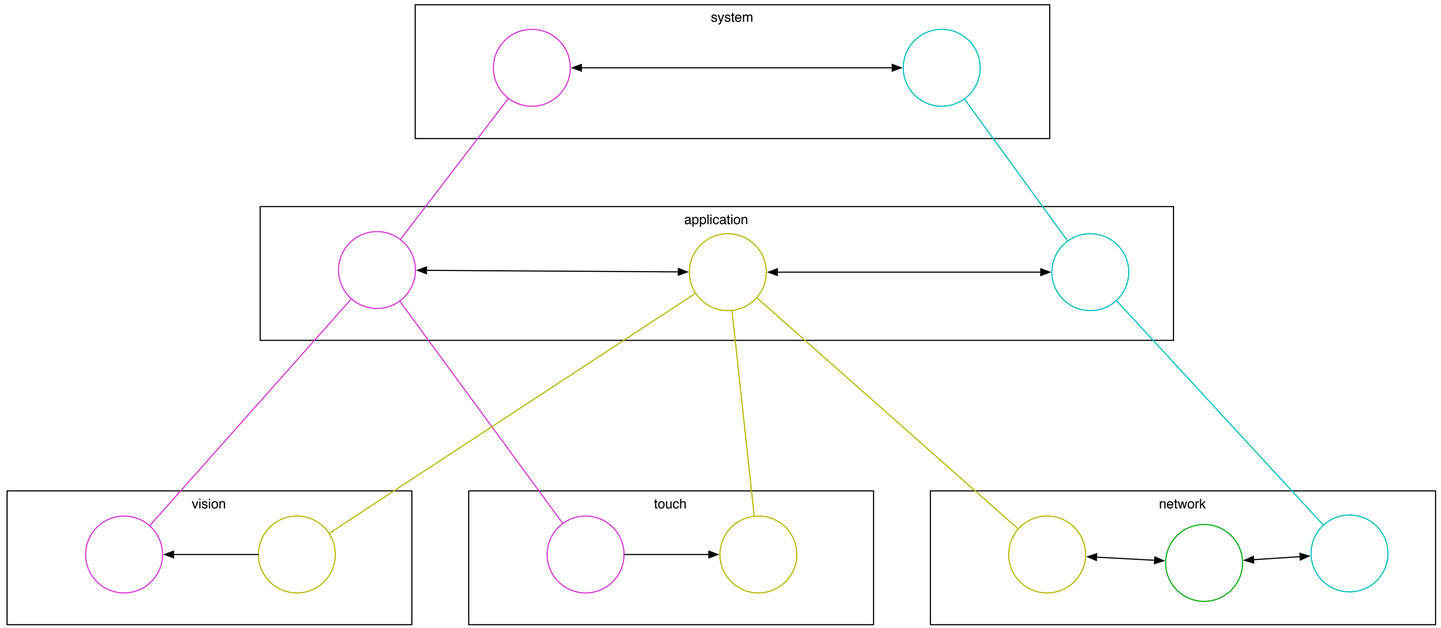


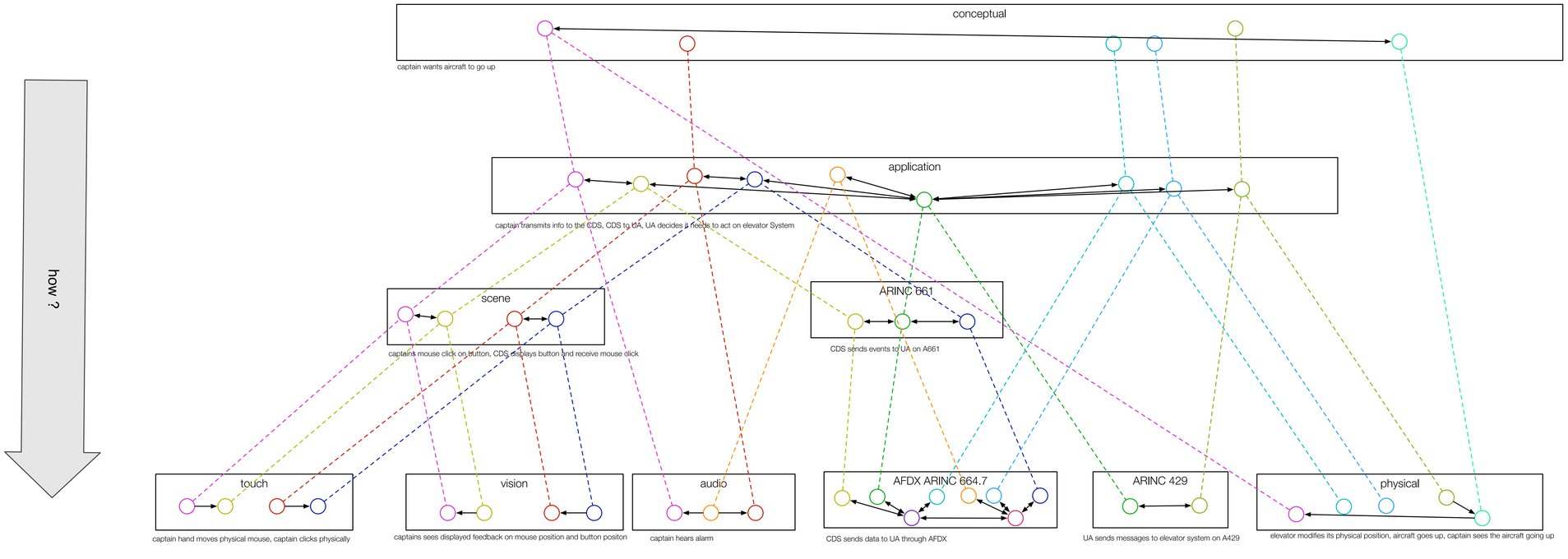


Different ways of concretization
- concretize signals origins and destination
- instantiate concrete sub components

Pamela Zave geomorphic layer model of networks




system
application
vision
touch
network



state variables and dynamic evolution and behavior are INSIDE Circles

flight warning FWS

first officer

captain

right display CDS R1

left display CDS L1

switch AFDX 1 1

switch AFDX 1 2

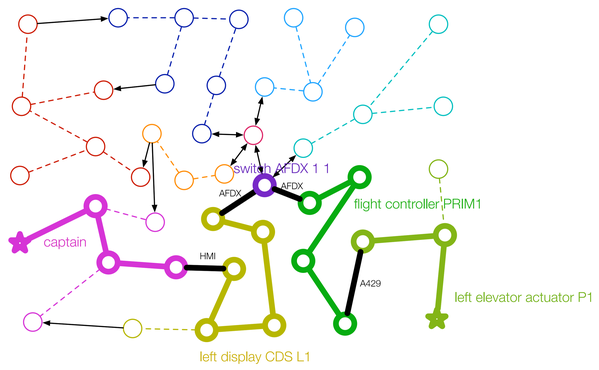
engine EEC2

engine EEC1

flight controller PRIM1

left elevator actuator P1

aircraft


switch AFDX 1 1
AFDX
AFDX
flight controller PRIM1
captain
HMI
A429
left elevator actuator P1
left display CDS L1

emission:

active(identifier)
value(identifier)
{identifier:emission,…,identifier:emission}

reception:

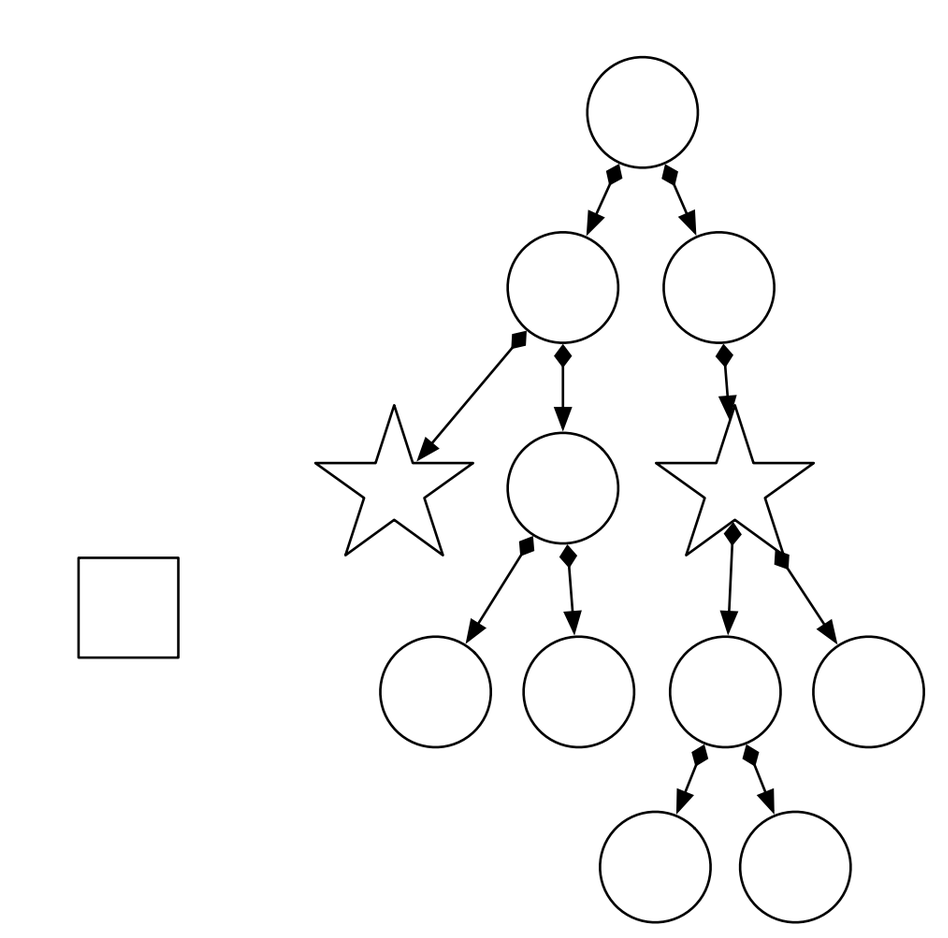
active(identifier)
value(identifier)
operator(reception,…,reception)
{identifier:reception,…,identifier:reception}

behavior:

affect(reception,emission)
on(reception,behavior)
always(behavior)
and(behavior,…,behavior)
or(behavior,…,behavior)

```
stepManage::behavior(min::reception,max::reception,init::reception,step::reception,var::emission,increment::reception,decrement::reception) :=
and(
    declare(temp),
    on(init(),
        affect(init,var)
    ),
    on(increment,
        affect(add(pre(var),+1), temp)
    ),
    on(decrement,
        affect(add(pre(var),-1), temp)
    ),
    ensureIsInRange(0,100,temp,var)
)
```


```
stepManage::behavior(min::reception,max::reception,init::reception,step::reception,var::emission,increment::reception,decrement::reception) :=
and(
    declare(temp),
    on(init(),
        affect(init,var)
    ),
    on(increment,
        affect(add(pre(var),+1), temp)
    ),
    on(decrement,
        affect(add(pre(var),-1), temp)
    ),
    ensureIsInRange(0,100,temp,var)
)
```

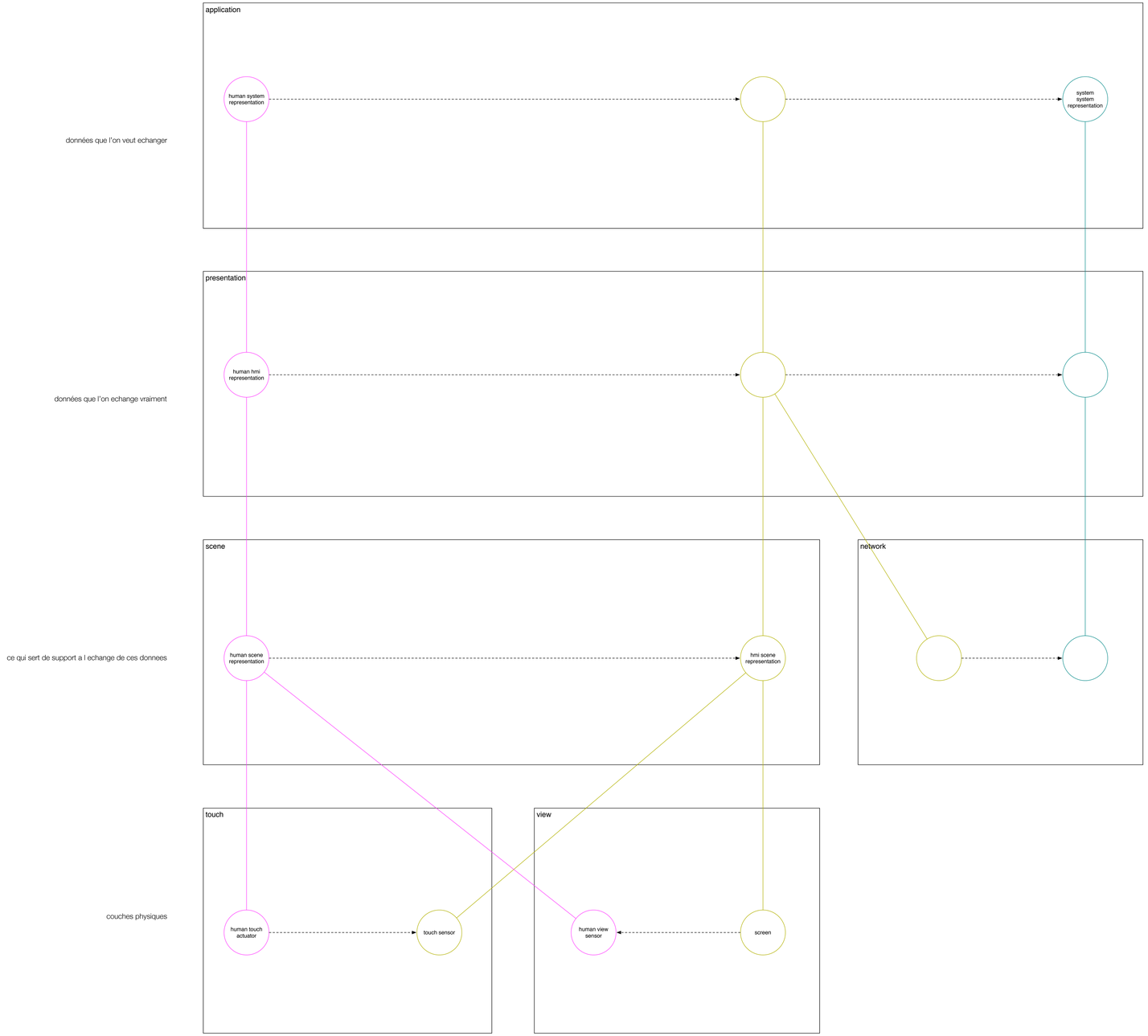



```
ensureIsInRange::behavior(min::reception,max::reception,temp::reception,var::emission) :=
                                                          affect(max(min,min(max,temp)),var)
```

```
trigger::behavior(a::identifier,b::reception) :=
                  and(affect(true,active(a)),affect(b,value(a)))
```

```
triggerntimeswithintervalt::behavior(n::reception::number,t::reception::time) :=
                                                                     last:time flow
```

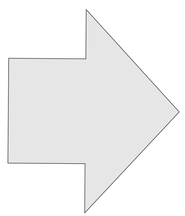
```
speedcontroller:continuous:{
    driver:continuous:{
        actualspeed:continuous:number:in,
        desiredspeed:continuous:number:out,
        increment:discrete:void:in,
        decrement:discrete:void:in,
        toggle:discrete:void:in,


    }
}
```


application
human system representation
system system representation
données que l'on veut echanger
presentation
human hmi representation
données que l'on echange vraiment
scene
network
human scene representation
hmi scene representation
ce qui sert de support a l echange de ces donnees
touch
view
human touch actuator
touch sensor
human view sensor
screen
couches physiques

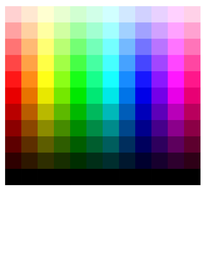
behavior:

affect(reception,emission)
on(reception,behavior)
always(behavior)
and(behavior,…,behavior)
or(behavior,…,behavior)

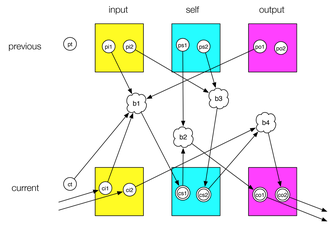
```
on(a,
    on(b,
        c = d
    )
)

on(e,
    on(f,
        c = g
    )
)
```



```
c =
    if((a&&b)&&not(e&&f)),
        d,
        if(e&&f,
            g,
            pre(c)
        )
    )
```



Transition Function Structure



Transition Function Rules

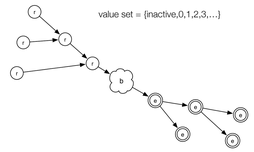
ct > pt t is for time

for all pi_n, exists ci_n
for all ps_n, exists cs_n
for all po_n, exists co_n

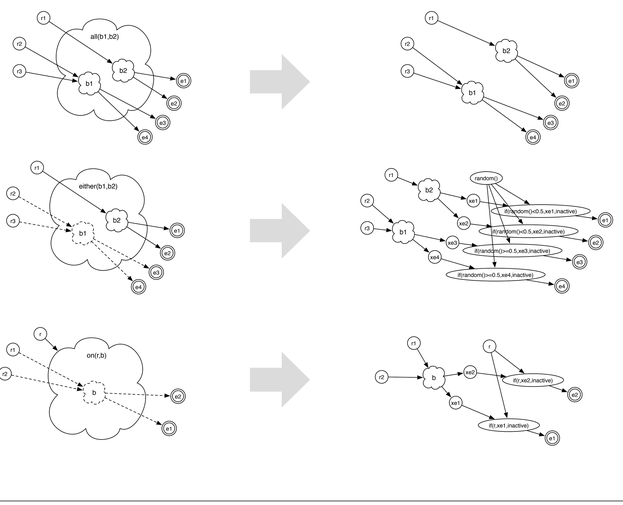
for all pi_n, in(pi_n) = 0
for all ps_n, in(ps_n) = 0
for all po_n, in(po_n) = 0
for all ci_n, in(ci_n) = 0
for all cs_n, in(cs_n) >= 0
for all co_n, in(co_n) >= 0

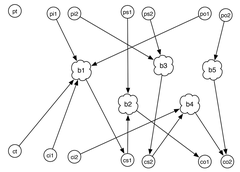
for all b_n, out(b_n) >= 1

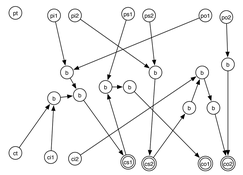
all nodes are reception node (noted e)
only cs_n and co_n are emission nodes (noted r)



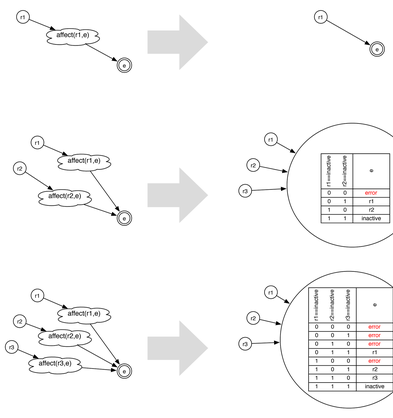
First transform : Behaviour elimination





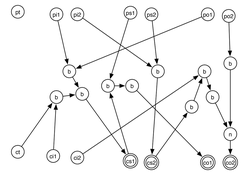


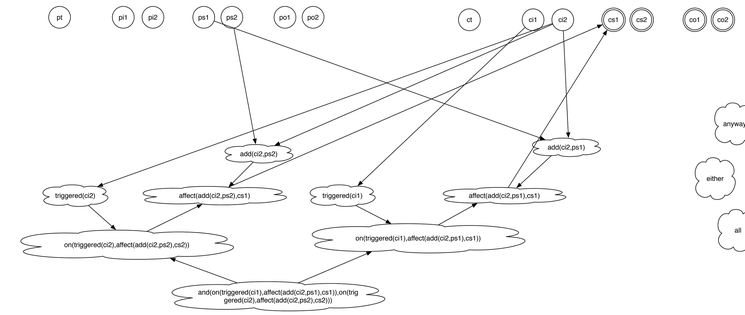
Second transform : Affectation elimination



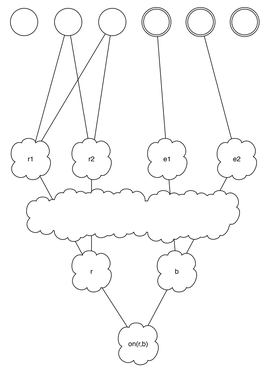
Quand on a plusieurs affectations sur le meme id, il faut prouver avec un soldeur SMT que la condition est UNSAT

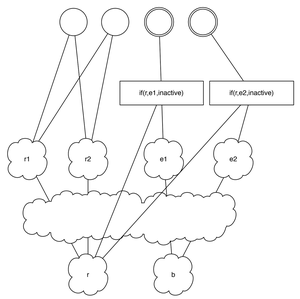
quand il y a un cycle, is faut prouver avec un solver SMT que la condition de bouclage est UNSAT

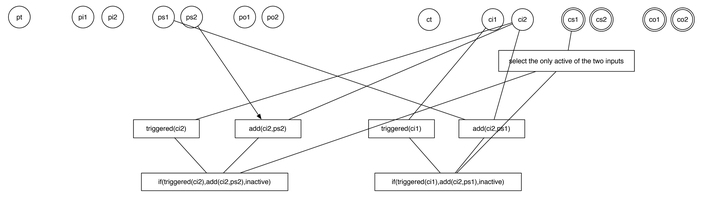


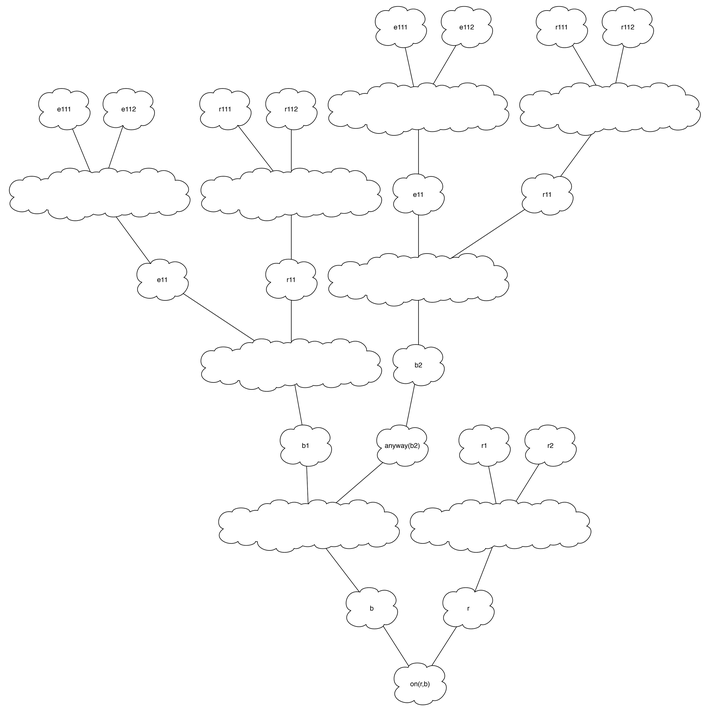

pt
pi1
pi2
ps1
ps2
po1
po2
ct
ci1
ci2
cs1
cs2
co1
co2
add(ci2,ps2)
add(ci2,ps1)
triggered(ci2)
affect(add(ci2,ps2),cs1)
triggered(ci1)
affect(add(ci2,ps1),cs1)
on(triggered(ci2),affect(add(ci2,ps2),cs2))
on(triggered(ci1),affect(add(ci2,ps1),cs1))
and(on(triggered(ci1),affect(add(ci2,ps1),cs1)),on(triggered(ci2),affect(add(ci2,ps2),cs2)))

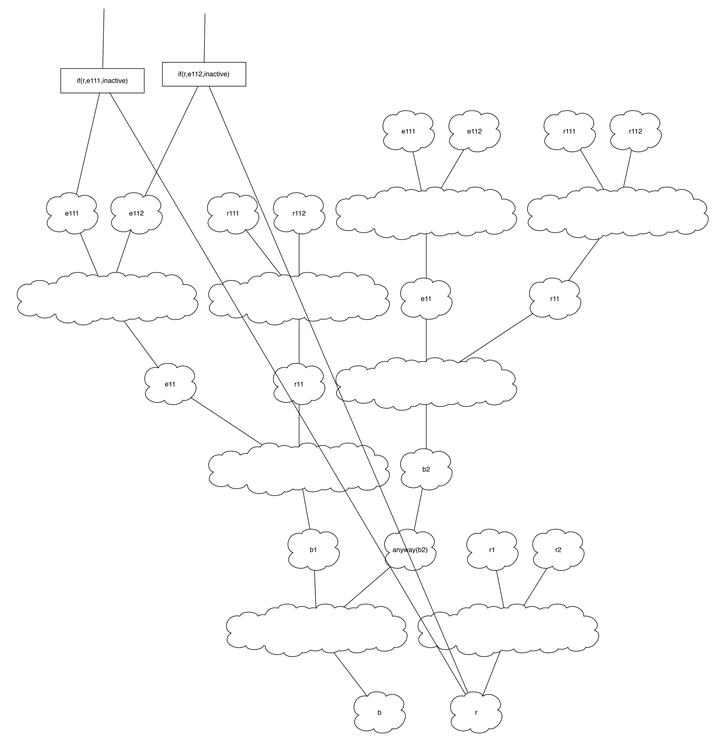

anyway
either
affect
all

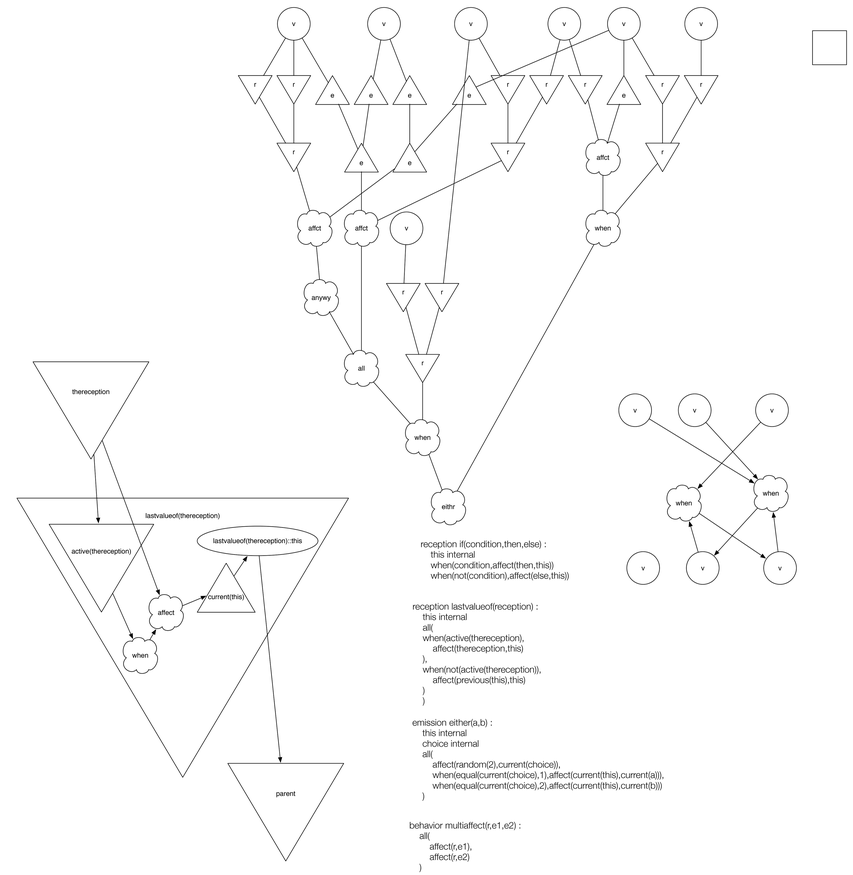

r1
r2
e1
e2
r
b
on(r,b)


if(r,e1,inactive)
if(r,e2,inactive)
r1
r2
e1
e2
r
b


pt
pi1
pi2
ps1
ps2
po1
po2
ct
ci1
ci2
cs1
cs2
co1
co2
select the only active of the two inputs
triggered(ci2)
add(ci2,ps2)
triggered(ci1)
add(ci2,ps1)
if(triggered(ci2),add(ci2,ps2),inactive)
if(triggered(ci1),add(ci2,ps1),inactive)


e111
e112
r111
r112
e11
r11
b2
b1
anyway(b2)
r1
r2
b
r
on(r,b)


if(r,e111,inactive)
if(r,e112,inactive)
e111
e112
r111
r112
e11
r11
b2
b1
anyway(b2)
r1
r2
b
r



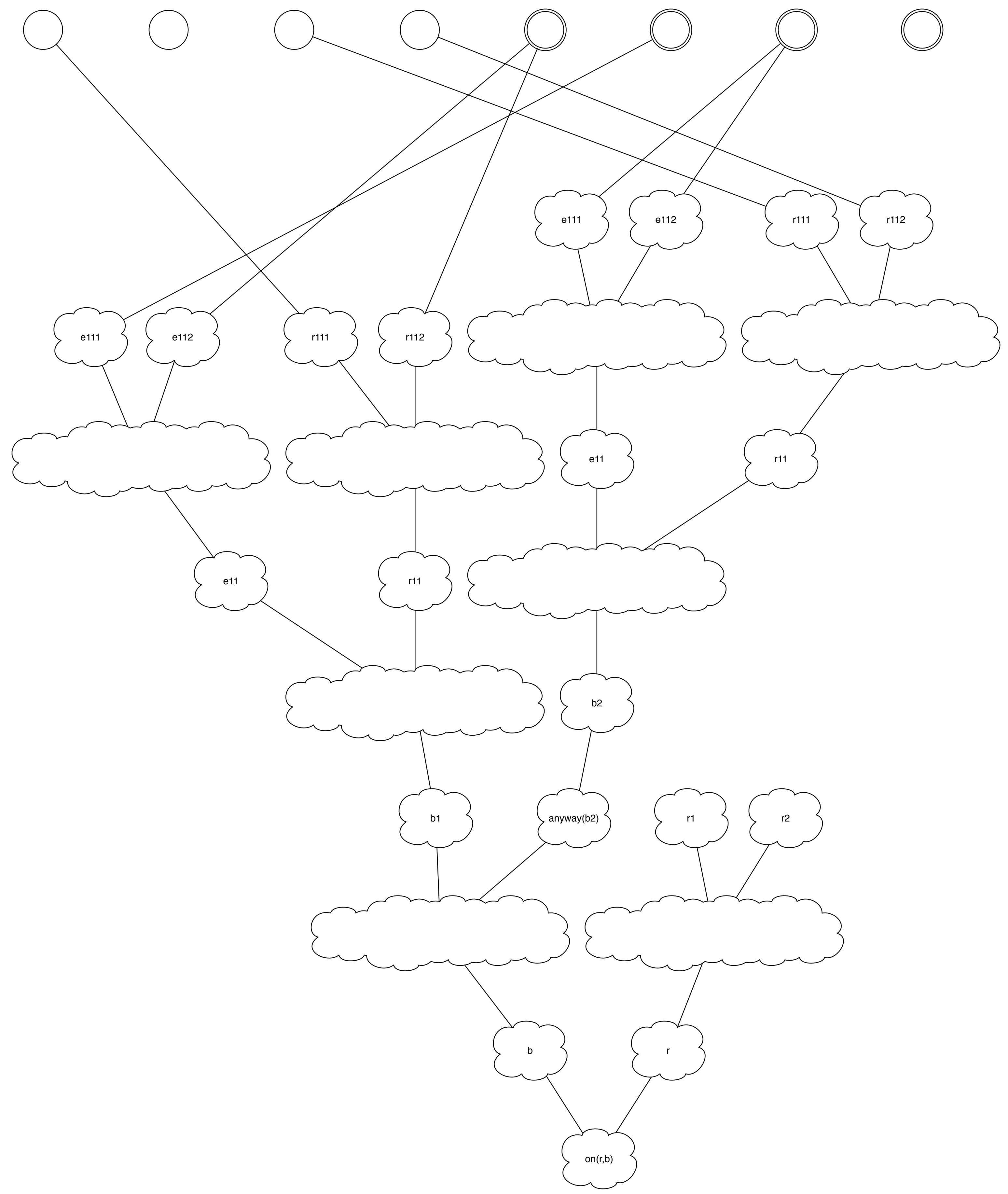

```
reception if(condition,then,else) :
    this internal
    when(condition,affect(then,this))
    when(not(condition),affect(else,this))
```

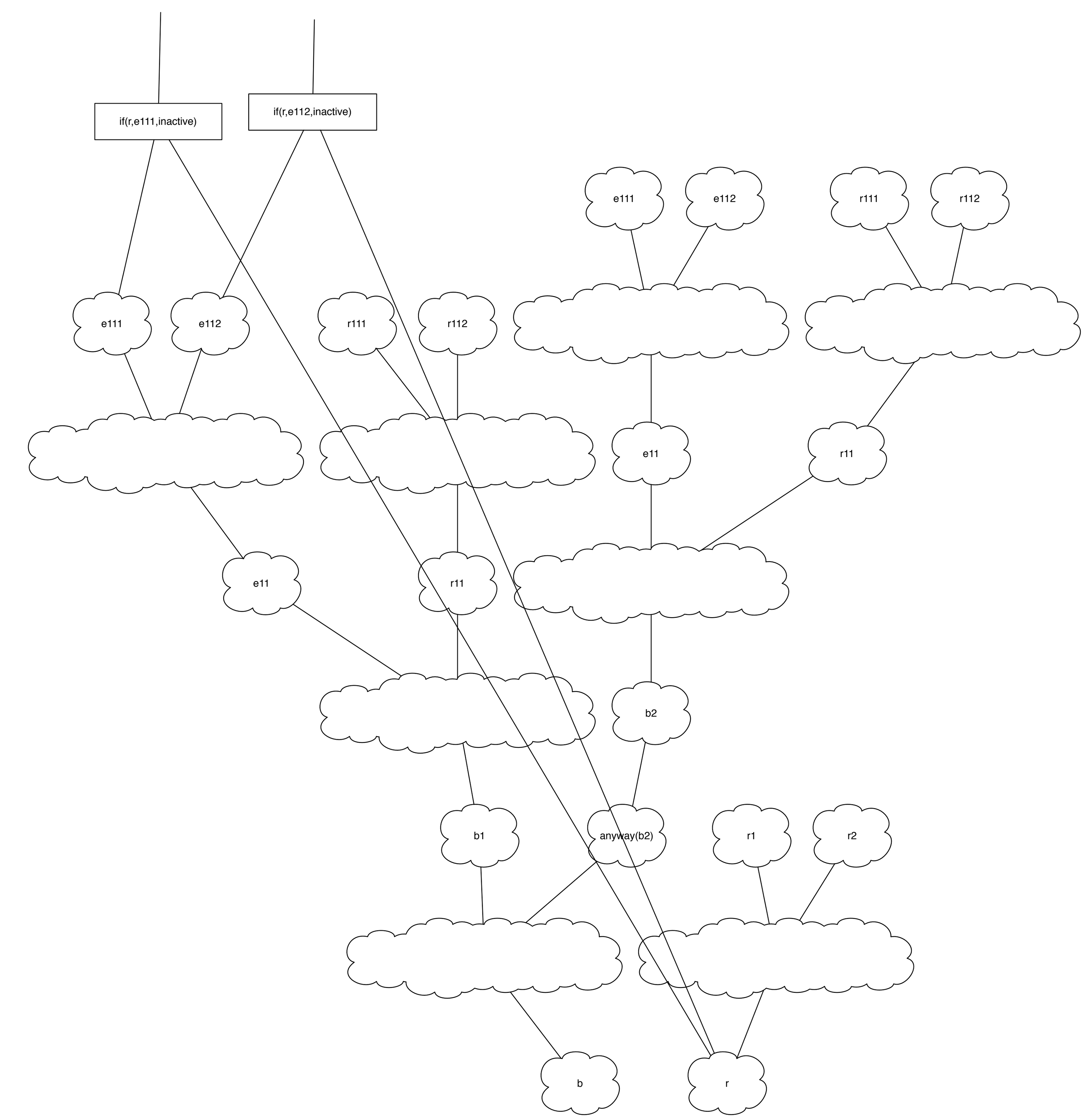
```
reception lastvalueof(reception) :
    this internal
    all(
    when(active(thereception),
        affect(thereception,this)
    ),
    when(not(active(thereception)),
        affect(previous(this),this)
    )
    )
```

```
emission either(a,b) :
    this internal
    choice internal
    all(
        affect(random(2),current(choice)),
        when(equal(current(choice),1),affect(current(this),current(a))),
        when(equal(current(choice),2),affect(current(this),current(b)))
    )
```

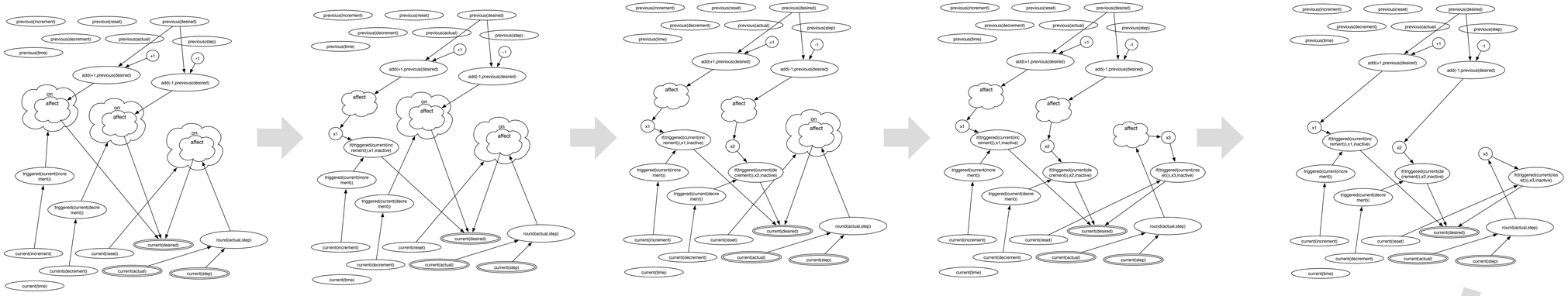
```
behavior multiaffect(r,e1,e2) :
    all(
        affect(r,e1),
        affect(r,e2)
    )
```

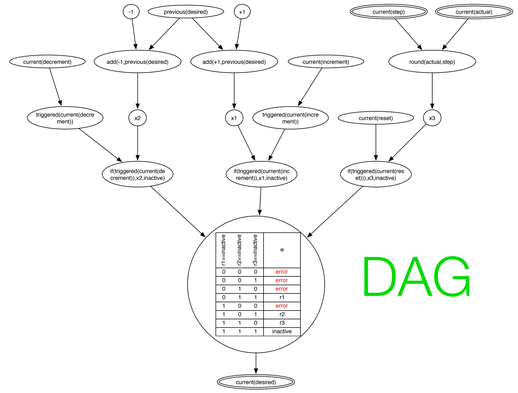
```
behavior executebehavioraftertime(b,delay,time) :
    this internal
    choice internal
    all(
        affect(random(2),current(choice)),
        when(equal(current(choice),1),affect(current(this),current(a))),
        when(equal(current(choice),2),affect(current(this),current(b)))
    )
```


e111
e112
r111
r112
e111
e112
r111
r112
e11
r11
e11
r11
b2
b1
anyway(b2)
r1
r2
b
r
on(r,b)


if(r,e111,inactive)
if(r,e112,inactive)
e111
e112
r111
r112
e111
e112
r111
r112
e11
r11
e11
r11
b2
b1
anyway(b2)
r1
r2
b
r

```
increment input
decrement input
reset input
actual output
desired output
step internal

on(triggered(increment),
     affect(add(+1,previous(desired)),current(desired))
)
on(triggered(decrement),
     affect(add(-1,previous(desired)),desired)
)
on(triggered(reset),
     affect(round(actual,step),desired)
)
```



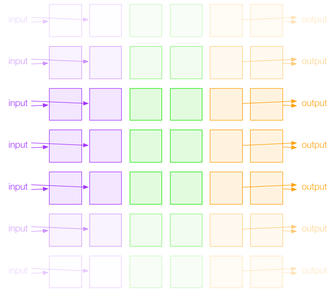


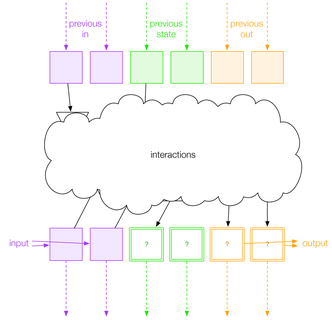
Merge Reception-Emissio-behavior

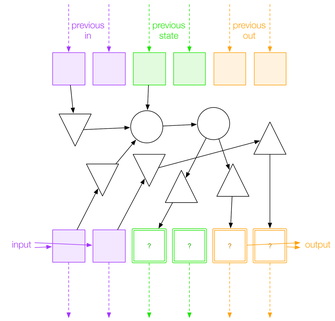
It's all the same thing finally

NO, because we could imagine receptions with side effects for example.

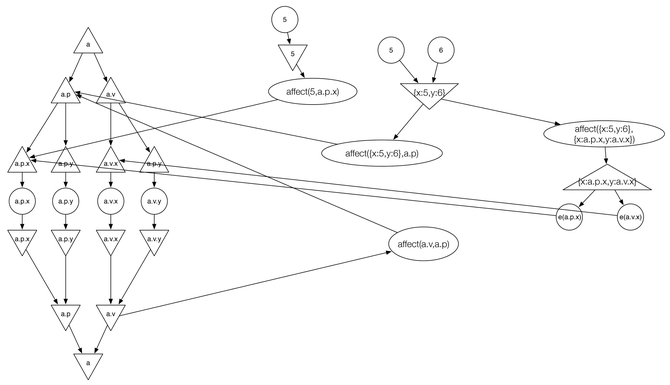
AND this would be wrong, because a reception with a side effect is misleading

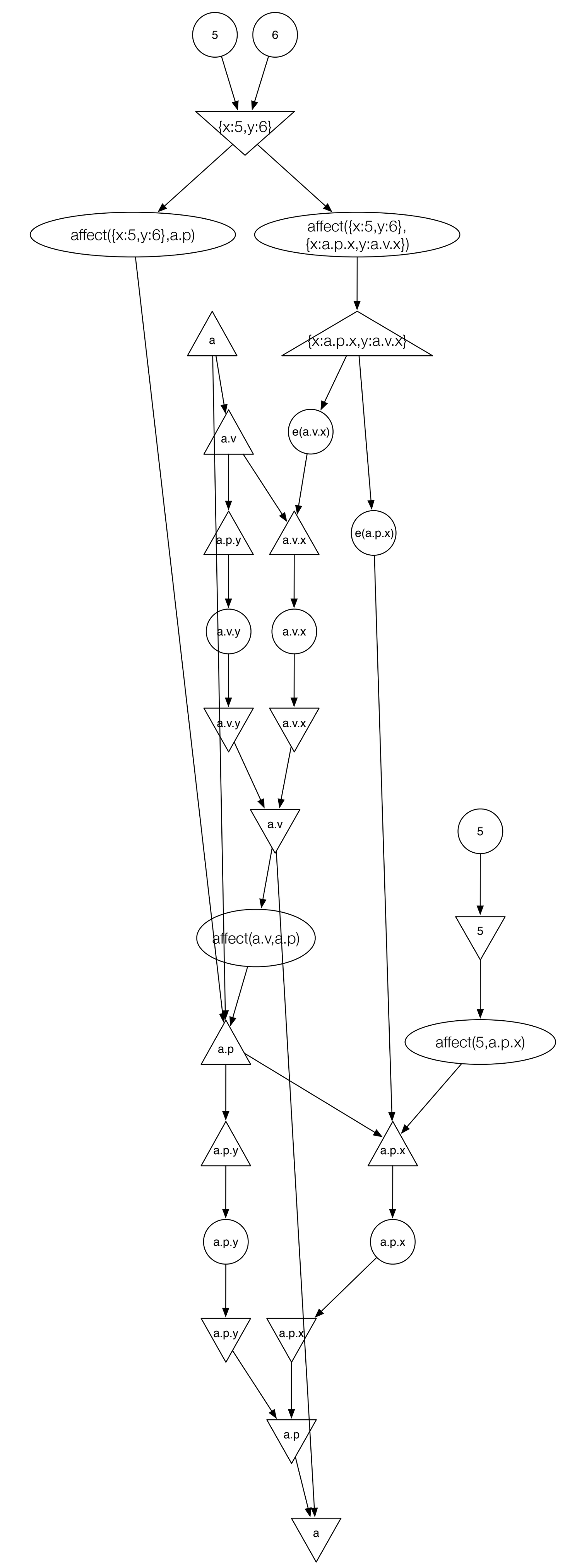

input
output
input
output
input
output
input
output
input
output
input
output
input
output

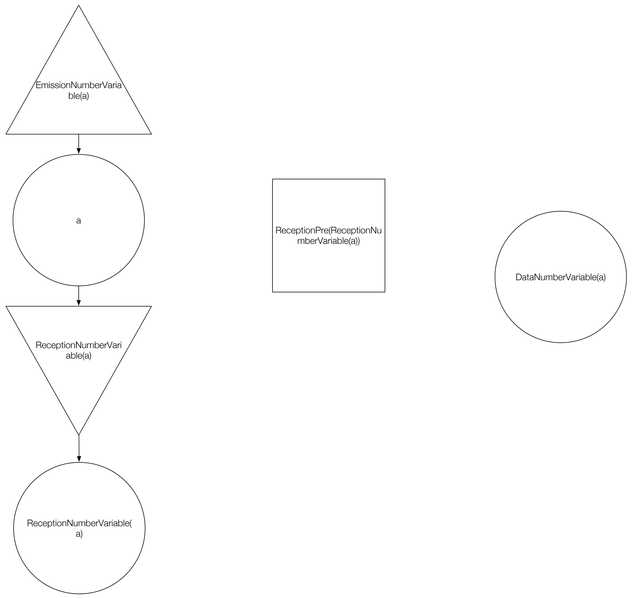

previous in
previous state
previous out
interactions
input
?
?
?
?
output


previous in
previous state
previous out
input
?
?
?
?
output

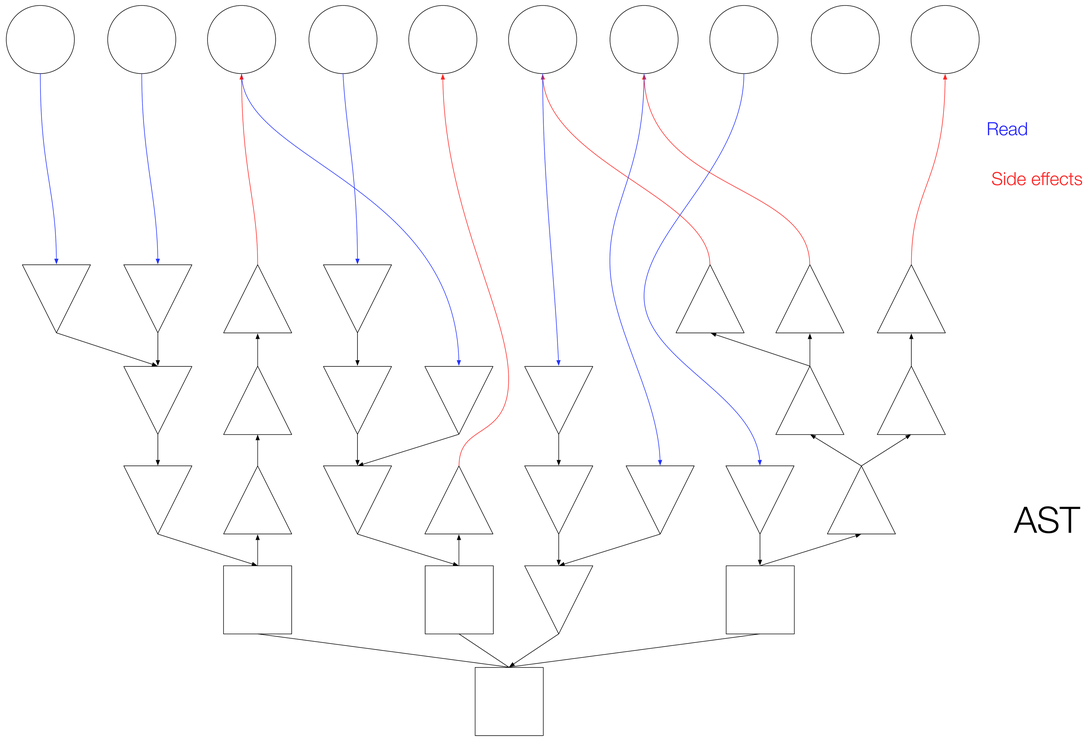
Composed data types : a=[p:[x:Number,y:Number],v:[x:Number,y:Number]]



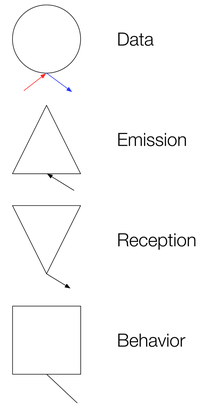

5
6
{x:5,y:6}
affect({x:5,y:6},a.p)
affect({x:5,y:6},
{x:a.p.x,y:a.v.x})
a
{x:a.p.x,y:a.v.x}
a.v
e(a.v.x)
a.p.y
a.v.x
e(a.p.x)
a.v.y
a.v.x
a.v.y
a.v.x
a.v
5
5
affect(a.v,a.p)
affect(5,a.p.x)
a.p
a.p.y
a.p.x
a.p.y
a.p.x
a.p.y
a.p.x
a.p
a



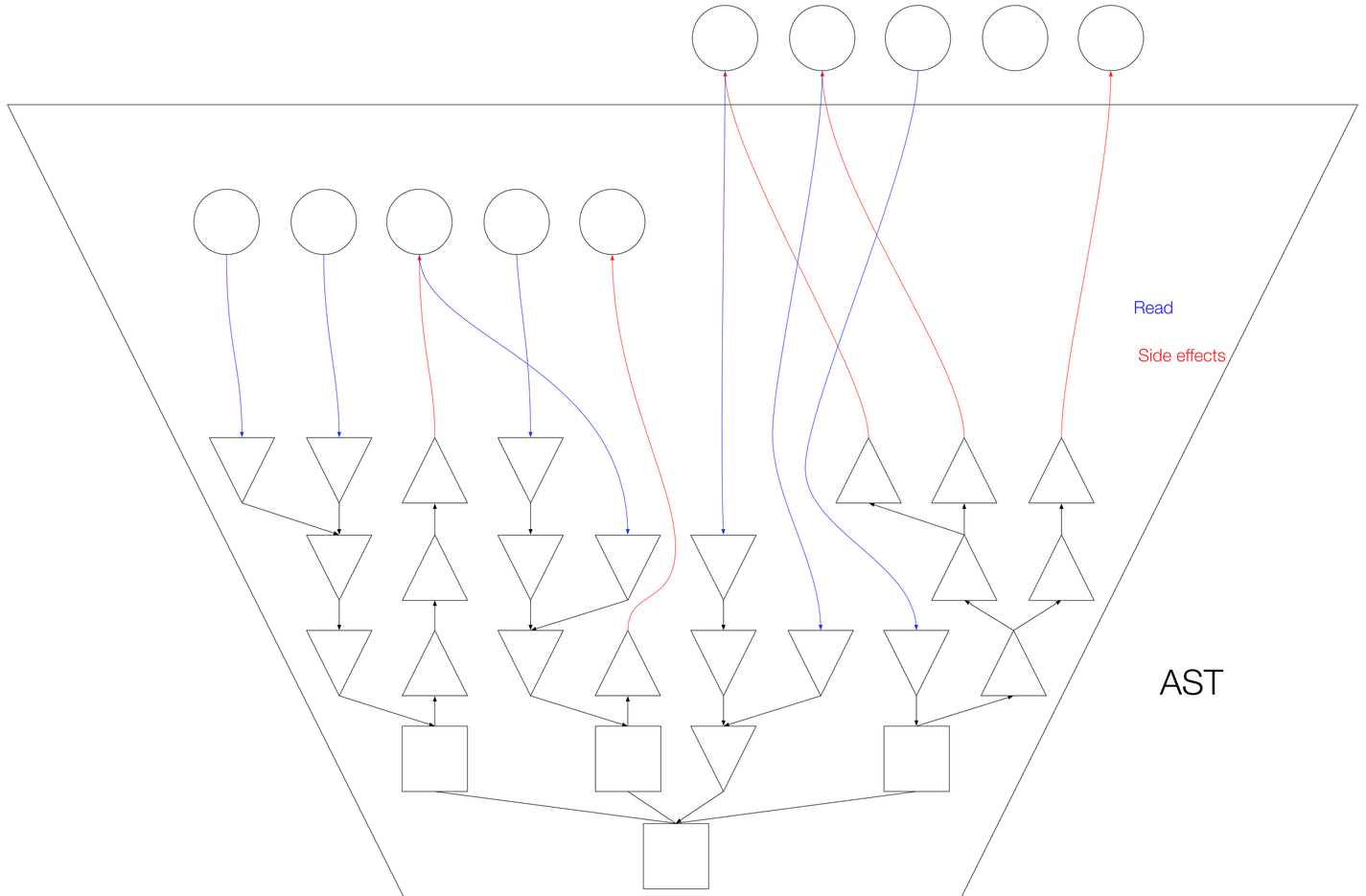
Data -> iii
Computation -> custom iii receptions
Interaction -> iii

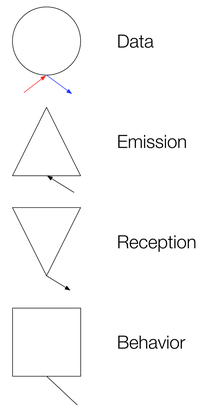


AST



1 - Unfold custom interactions receptions and emissions
2 - Eliminate whens
3 - Eliminate anyways
4 - Eliminate alls





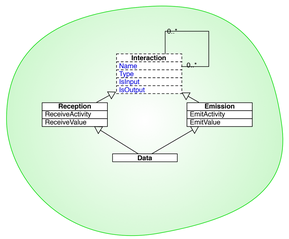
AST

1 - Unfold custom interactions receptions and emissions
2 - Eliminate whens
3 - Eliminate anyways
4 - Eliminate alls

ACTIVE FOR ALL ! NOT for the data, but for the actual NODES

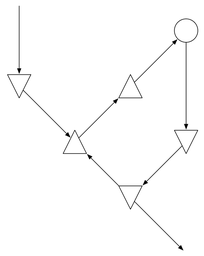
Activity for input = input received
Activity for output = we send this output
activity for variable = this thing is defined
activity for behaviour = execute this now

Unified into node activity




```
emission affectOrFlow(data b):
  affect(
    if(active(this),this,previous(b)),
    b
  )
```

```
void emission affectAfterDelay(time reception delay,reception source, emission destination):
  when(this,
     all(
       affect(firstActive(source,previous(local.value)),local.value),
       affect(firstActive(time,previous(local.time)),local.time),
       anyway(
         when(becameTrue(moreThan(time,add(local.time,delay))),
           affect(local.value,destination)
         )
       )
     )
   )
```



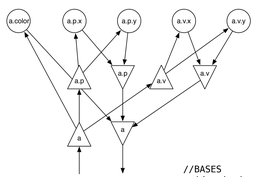
```
boolean reception becameTrue(boolean reception condition) :
  and(condition,not(previous(condition)))


void reception flow(data a,reception newA):
  affect(if(active(newA),newA,previous(a)),a)


 when(conditionMet(equals(mode,1)),
   all(
     triggerEventsOfQueue(thequeue),
     when(click,
       all(
         addEventToQueue(thequeue,counter),
         affect(add(1,previous(counter)),newcounter),
         anyway(flow(counter,newcounter))
       )
     )
   )
 )
```

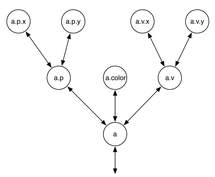
```
number reception incDec(
  void reception inc, void reception dec, number reception step,
  number reception min, number reception max, number reception actual ) :

all(
  when(increment,
    affect(add(previous(this),step),desired)
  ),
  when(decrement,
    affect(sub(previous(this),step),desired)
  ),
  when(reset,
    affect(round(actual,step),desired)
  ),
  affect(crop(desired,min,max),this)
)
```

```
<type> emission flow(<type> data target):
  affect(
    if(active(emitted),emitted,previous(target)),
    target
  )
```

```
mark data mark(color data color, point data p,point data v):
  when(emitted.color,affect(emitted.color,color))
  when(emitted.p,affect(emitted.p,p))
  when(emitted.v,affect(emitted.v,v))

  when(color,affect(color,received.color))
  when(p,affect(p,received.p))
  when(v,affect(v,received.v))
```

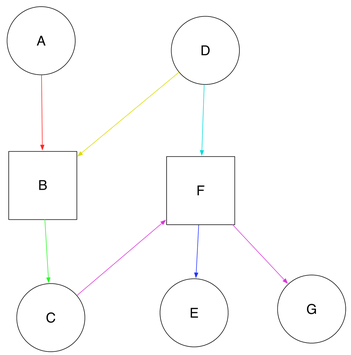
```
point data point(number data x, number data y):
  when(emitted.x,affect(emitted.x,x))
  when(emitted.y,affect(emitted.y,y))

  when(x,affect(x,received.x))
  when(y,affect(y,received.y))
```

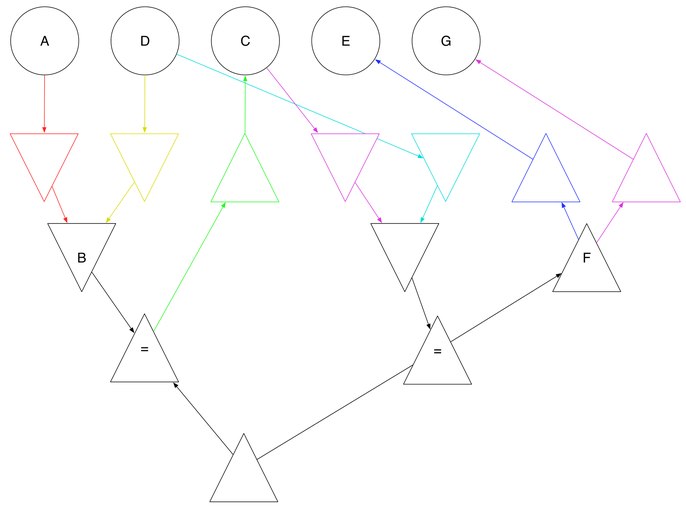


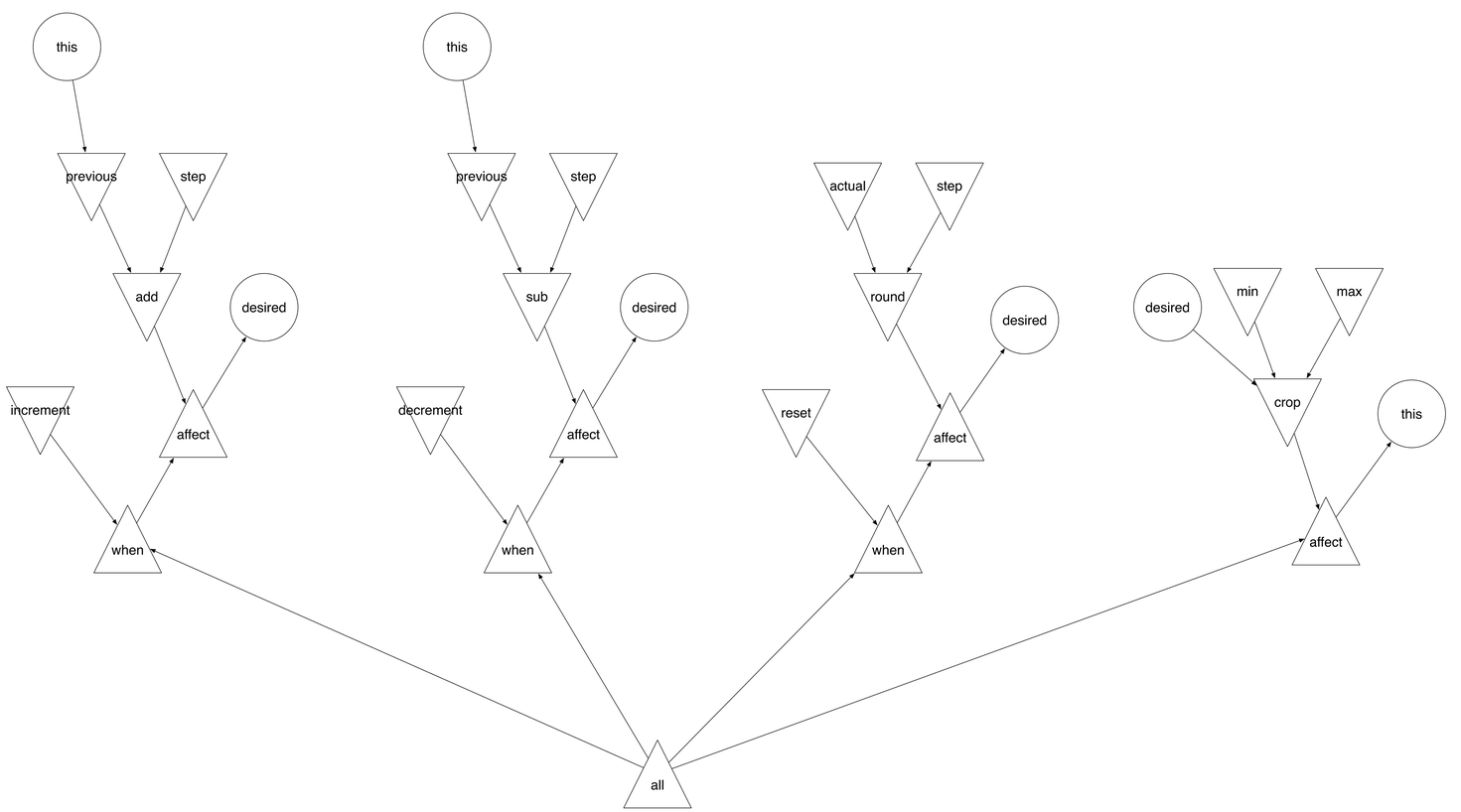


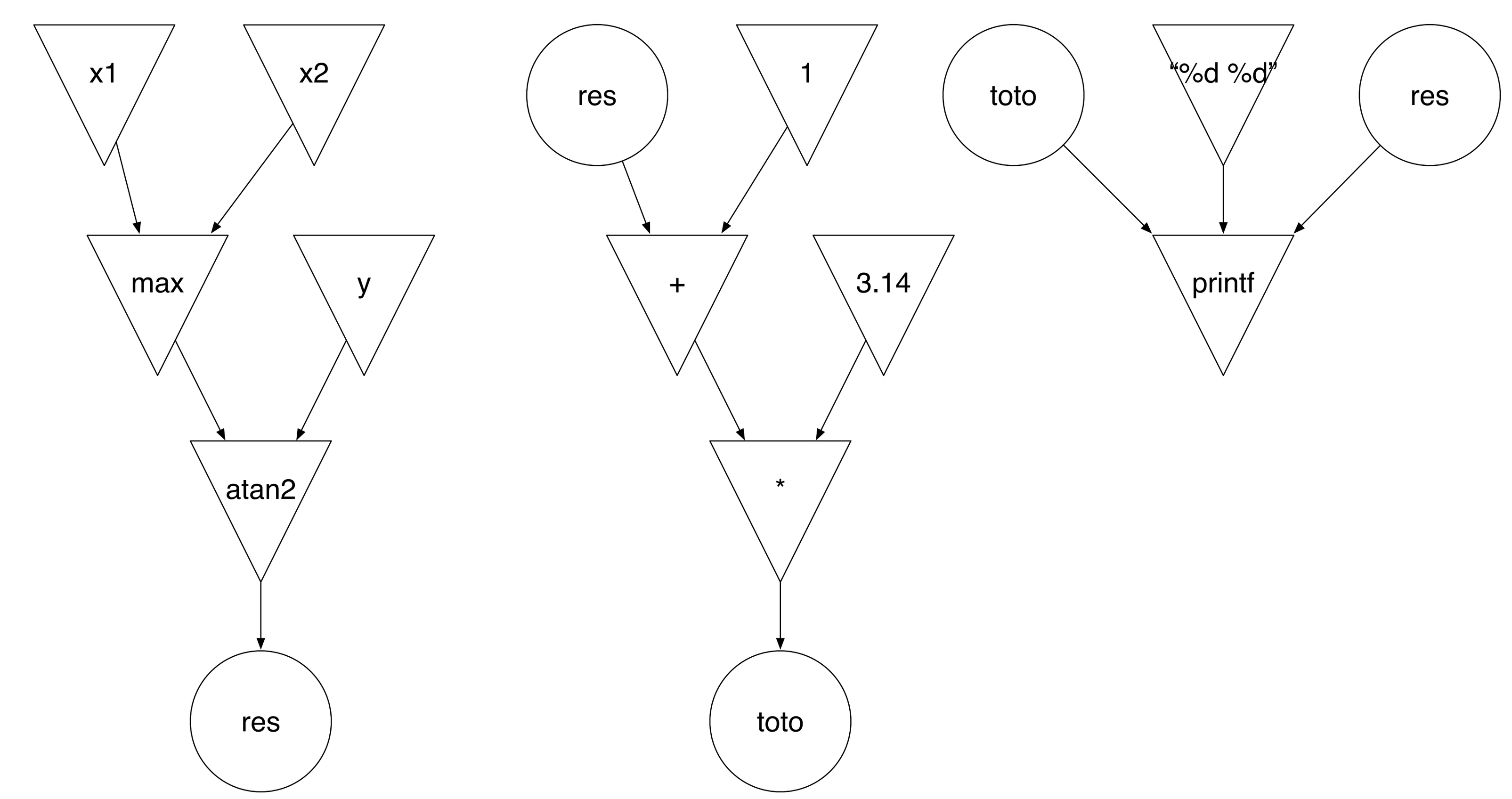

```
//BASES
void emission all()

void emission when(void reception activation, void emission do)

reception if(boolean reception if, reception then, reception else)

text data text(identifier)

number data number(identifier)

boolean data boolean(identifier)
```

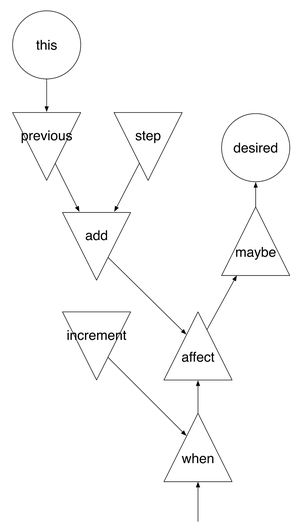
```
void emission affect(<type> reception source, <type> emission destination)

<type> emission all(<type> emission interactions...)

<type> emission either(<type> emission interactions...)

void emission when(void reception activation, void emission interaction)

void emission anyway(void emission interaction)

void emission doNot(void emission interaction)

void reception isTrue(boolean reception value)

<type> reception previous(<type> reception value)

boolean reception not(boolean reception value)

<type> reception if(boolean reception if, <type> reception then, <type> reception else)
```

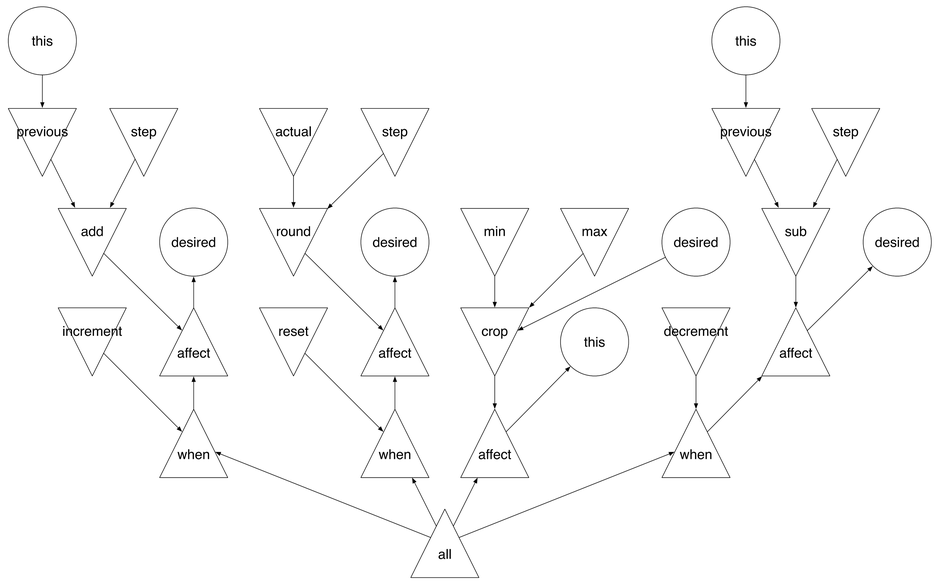

A
D
B
F
C
E
G

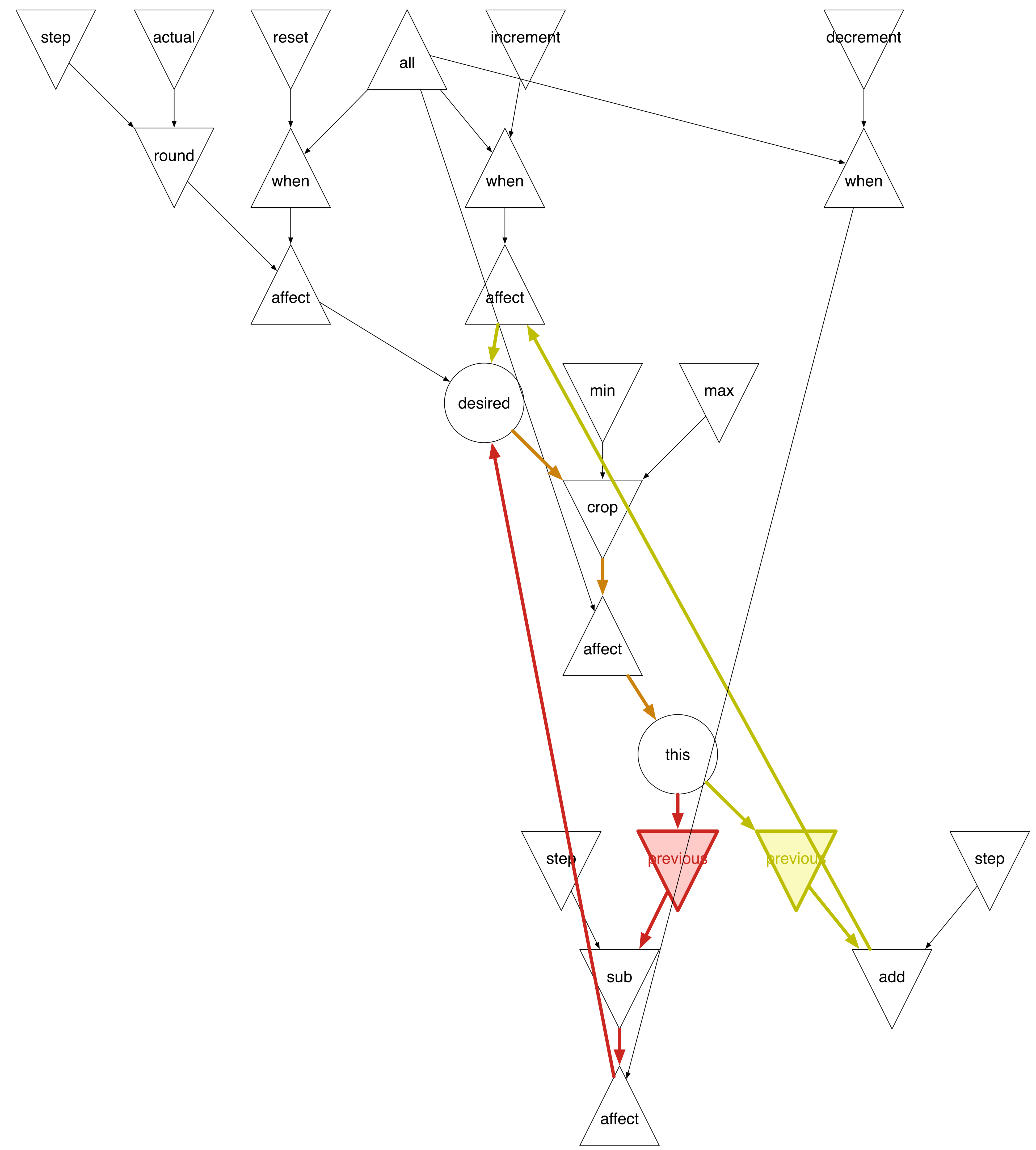

A
D
C
E
G
B
=
=
F


this
previous
step
add
desired
increment
affect
when
this
previous
step
sub
desired
decrement
affect
when
actual
step
round
desired
reset
affect
when
desired
min
max
crop
this
affect
all

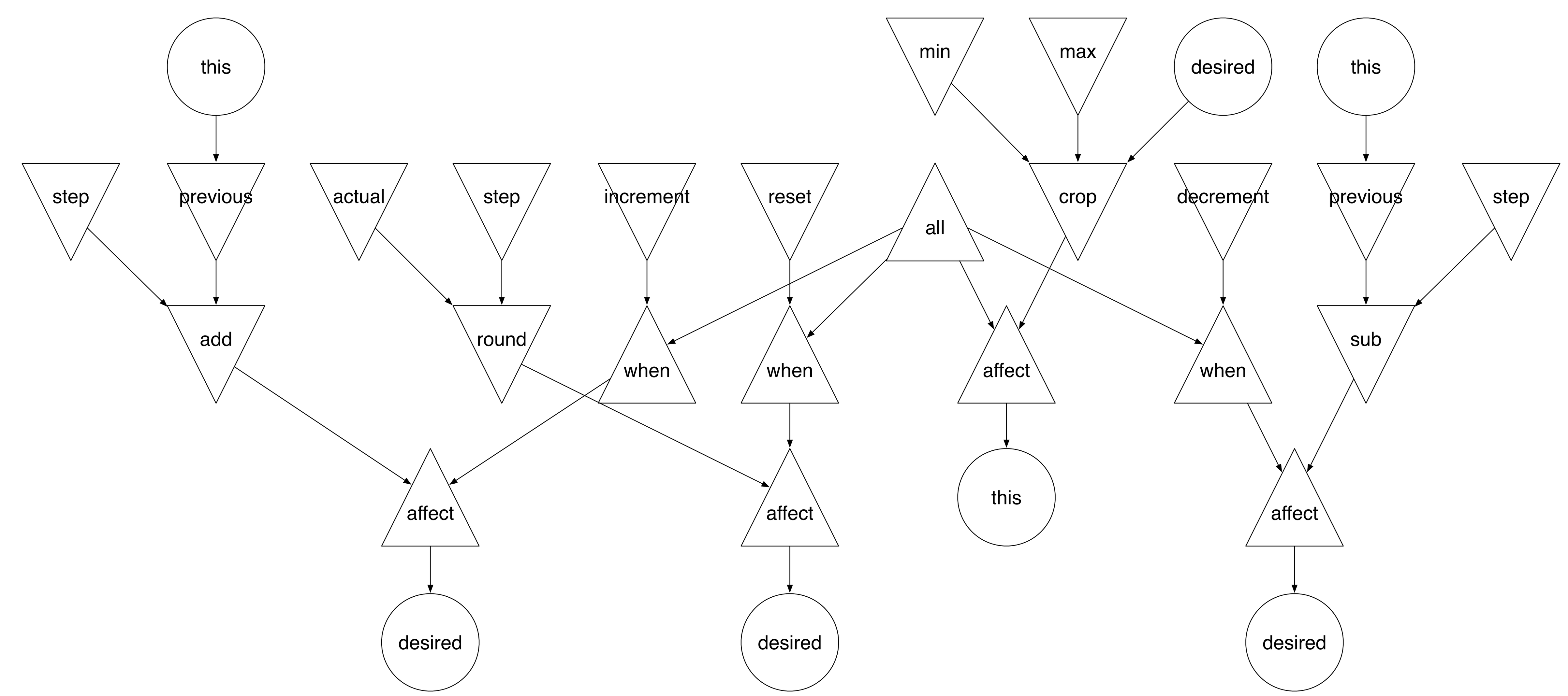

x1
x2
max
y
atan2
res
res
1
+
3.14
*
toto
toto
"%d %d"
res
printf


this
previous
step
desired
add
maybe
increment
affect
when

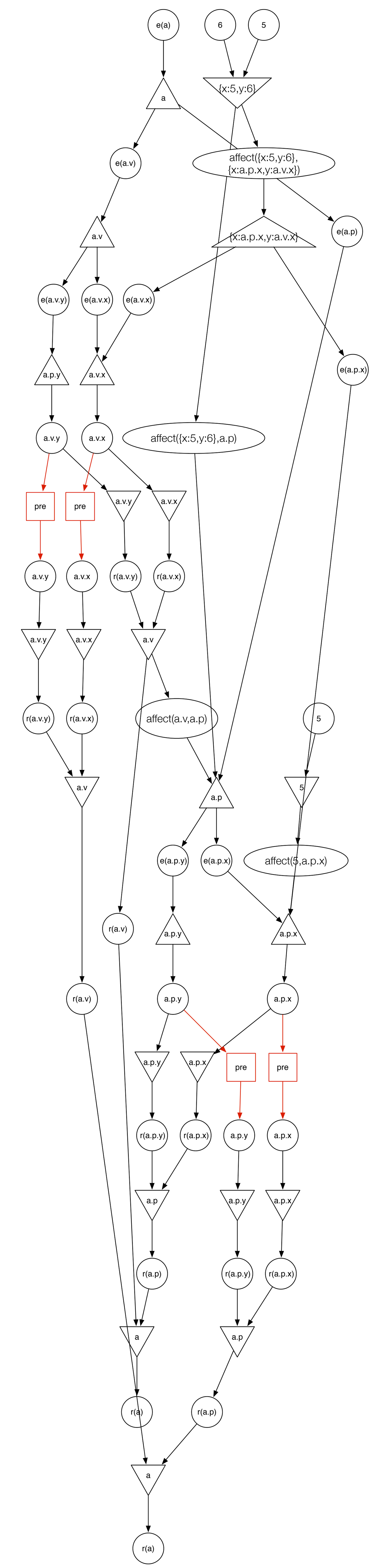

this
previous
step
add
desired
increment
affect
when
actual
step
round
desired
reset
affect
when
min
max
desired
crop
this
affect
all
this
previous
step
sub
desired
decrement
affect
when

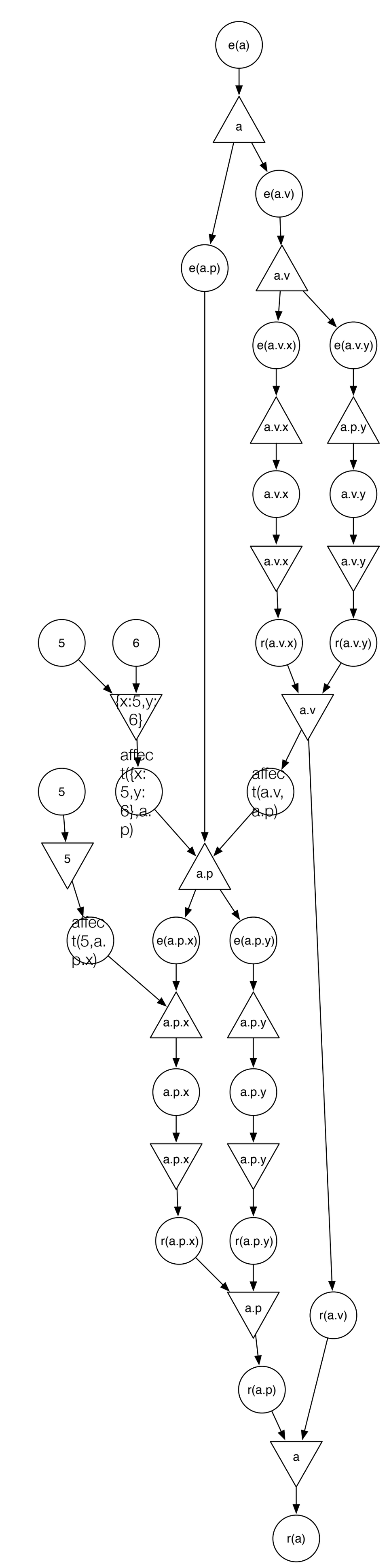


step
actual
reset
all
increment
decrement
round
when
when
when
affect
affect
desired
min
max
crop
affect
this
step
previous
previous
step
sub
add
affect



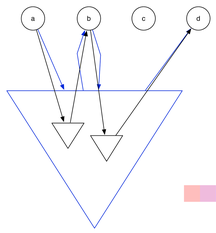
this
min
max
desired
this
step
previous
actual
step
increment
reset
all
crop
decrement
previous
step
add
round
when
when
affect
when
sub
affect
affect
this
affect
desired
desired
desired


e(a)
6
5
a
{x:5,y:6}
e(a.v)
affect({x:5,y:6},
{x:a.p.x,y:a.v.x})
a.v
e(a.p)
{x:a.p.x,y:a.v.x}
e(a.v.y)
e(a.v.x)
e(a.v.x)
a.p.y
a.v.x
e(a.p.x)
a.v.y
a.v.x
affect({x:5,y:6},a.p)
pre
pre
a.v.y
a.v.x
a.v.y
a.v.x
r(a.v.y)
r(a.v.x)
a.v.y
a.v.x
a.v
r(a.v.y)
r(a.v.x)
affect(a.v,a.p)
5
a.v
a.p
5
e(a.p.y)
e(a.p.x)
affect(5,a.p.x)
r(a.v)
a.p.y
a.p.x
r(a.v)
a.p.y
a.p.x
a.p.y
a.p.x
pre
pre
r(a.p.y)
r(a.p.x)
a.p.y
a.p.x
a.p
a.p.y
a.p.x
r(a.p)
r(a.p.y)
r(a.p.x)
a
a.p
r(a)
r(a.p)
a
r(a)



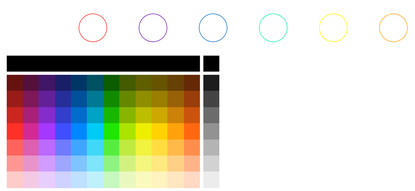
e(a)
a
e(a.v)
e(a.p)
a.v
e(a.v.x)
e(a.v.y)
a.v.x
a.p.y
a.v.x
a.v.y
a.v.x
a.v.y
5
6
r(a.v.x)
r(a.v.y)
{x:5,y:6}
a.v
affect({x:5,y:6},a.p)
5
affect(a.v,a.p)
5
a.p
affect(5,a.p.x)
e(a.p.x)
e(a.p.y)
a.p.x
a.p.y
a.p.x
a.p.y
a.p.x
a.p.y
r(a.p.x)
r(a.p.y)
a.p
r(a.v)
r(a.p)
a
r(a)

Pathologic



cycle when folded

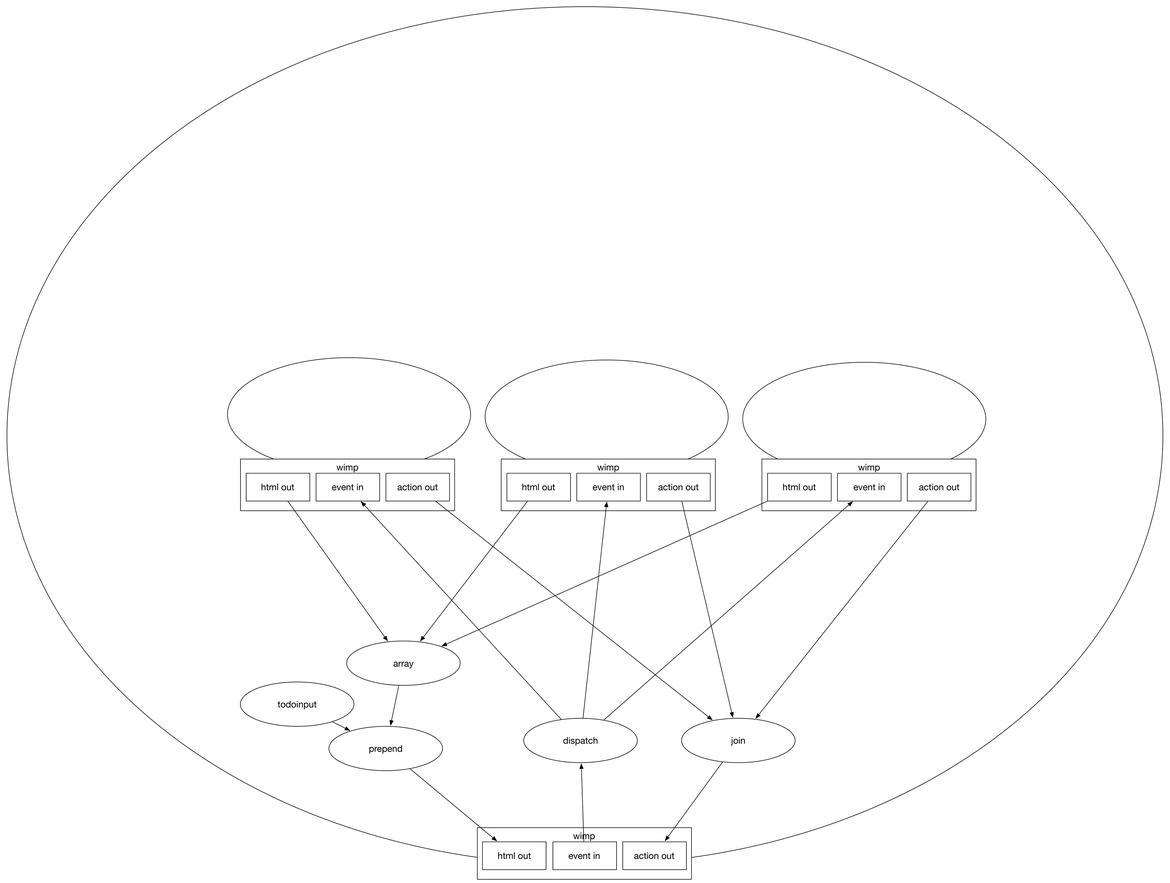
nocycle when unfolded



Duck typing

Multiple dispatch

Functional Reactive Programming



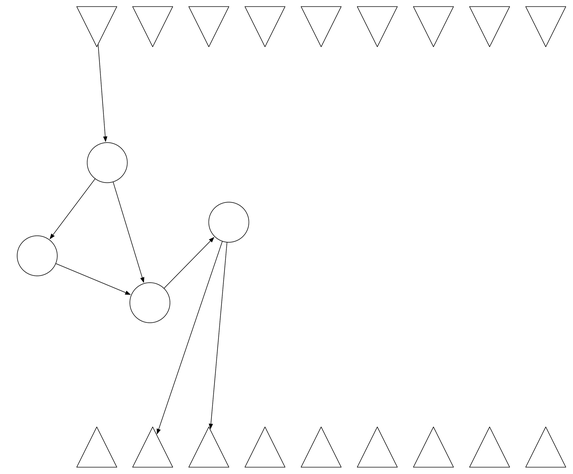
Position philosophique

Symétrie avec previous et next

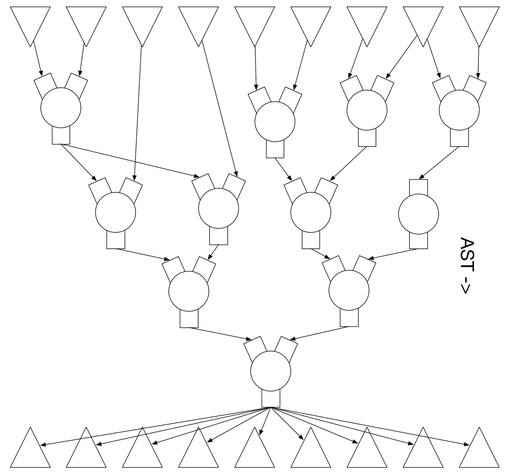
Ou

Asymétrie du temps avec seulement previous

Swing

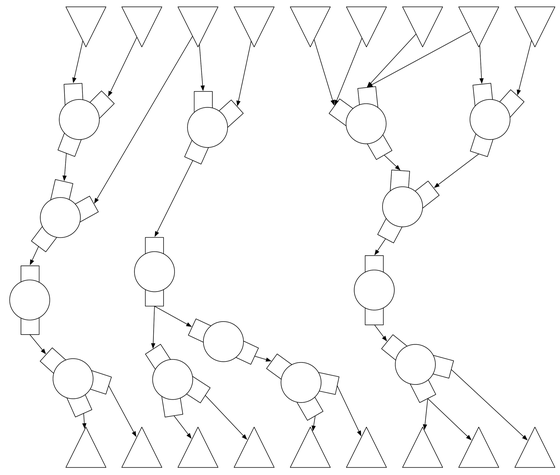


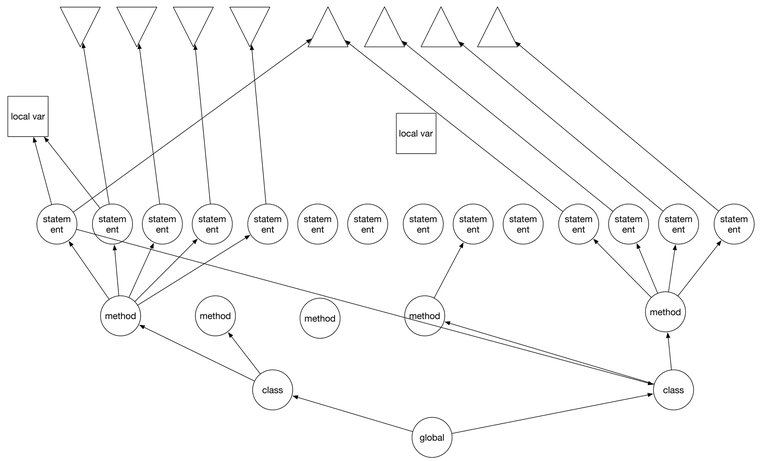
FRP

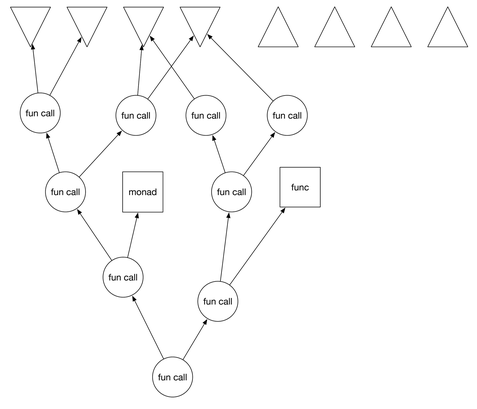


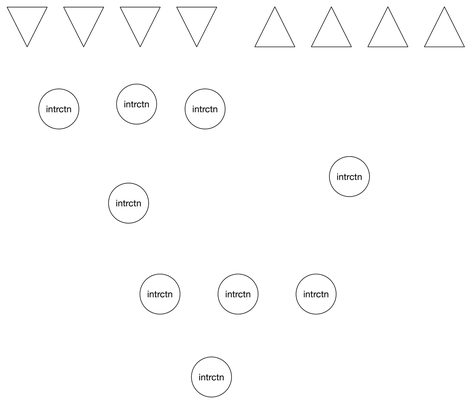
III

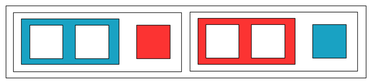
AST ->

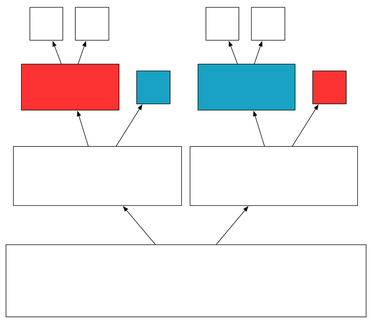


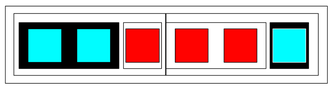


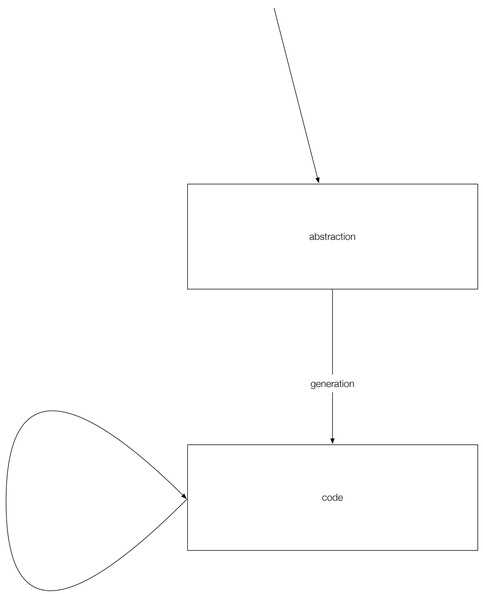


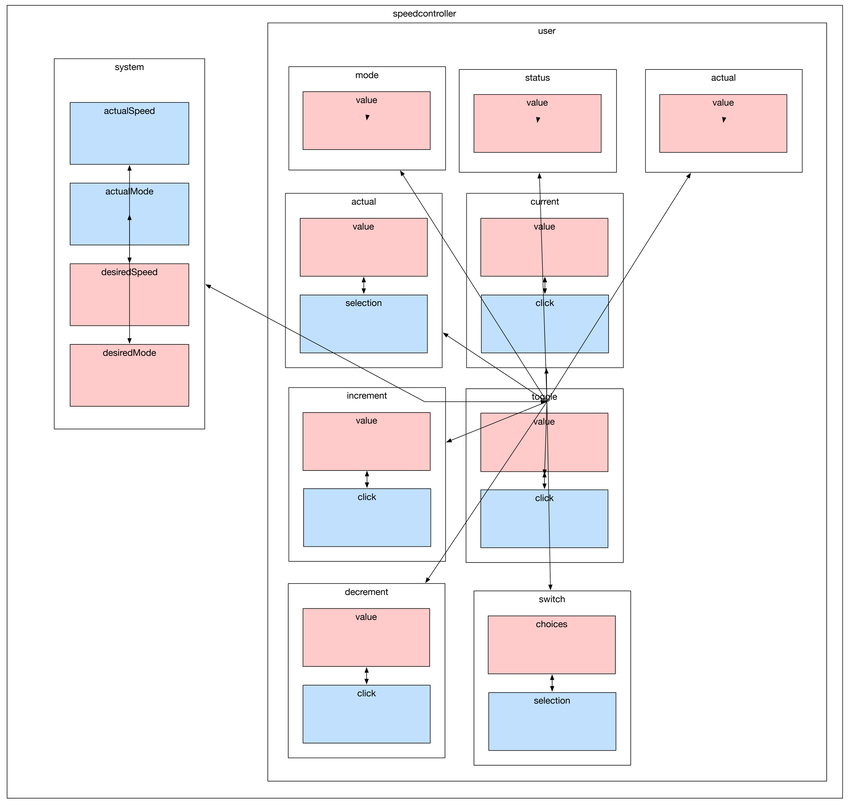


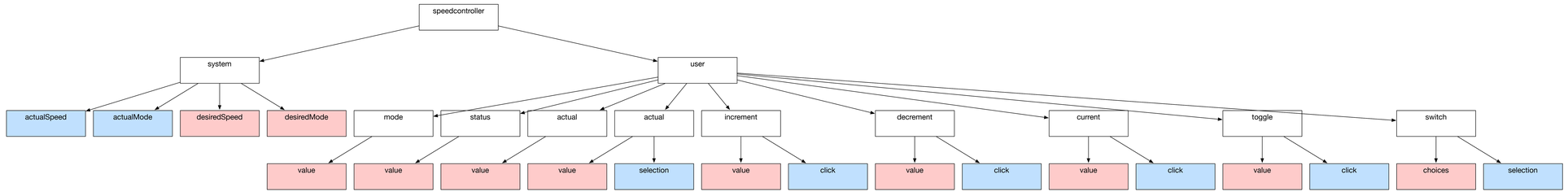


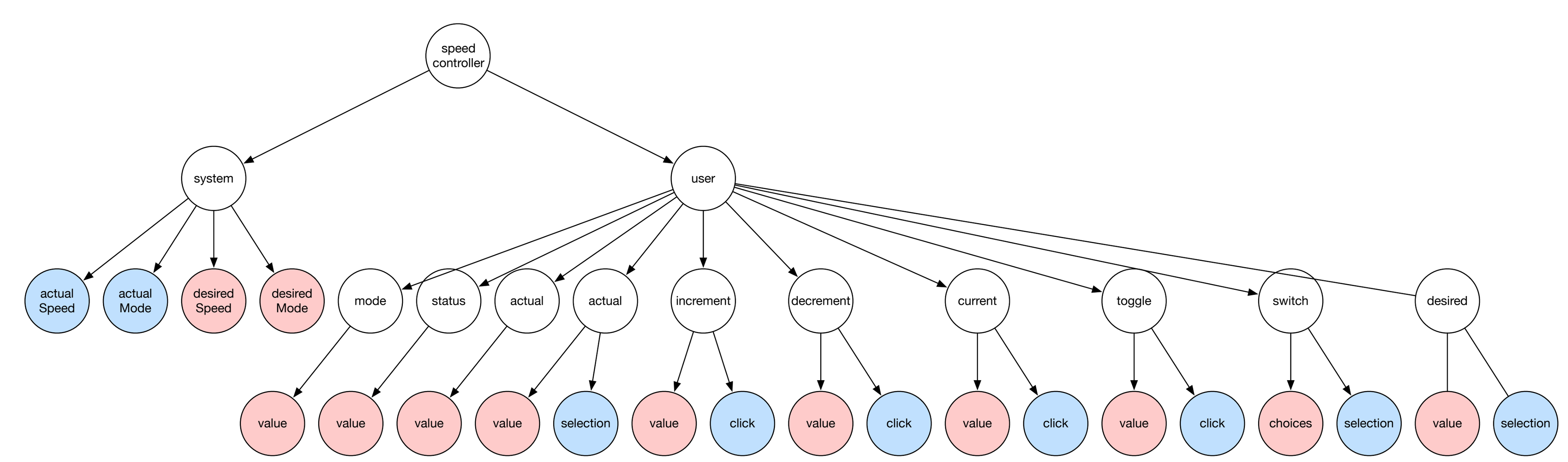


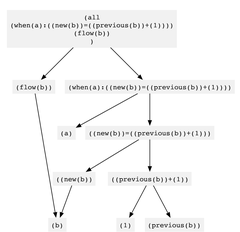


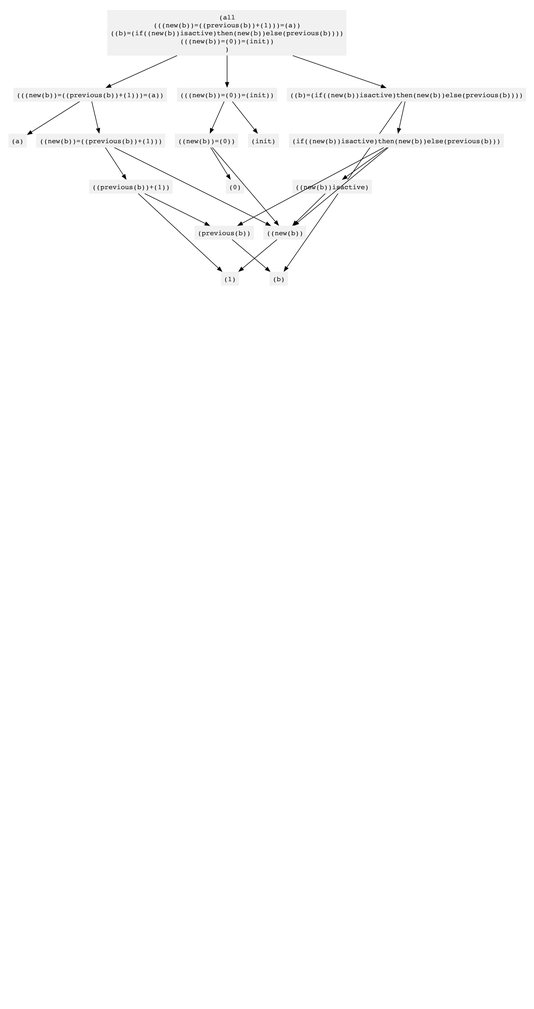

abstraction
generation
code


speedcontroller
user
system
actualSpeed
actualMode
desiredSpeed
desiredMode
mode
value
status
value
actual
value
actual
value
selection
current
value
click
increment
value
click
toggle
value
click
decrement
value
click
switch
choices
selection

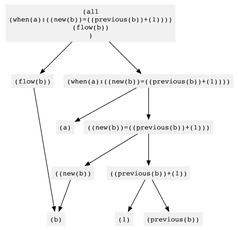

speedcontroller
system
user
actualSpeed
actualMode
desiredSpeed
desiredMode
mode
status
actual
actual
increment
decrement
current
toggle
switch
value
value
value
value
selection
value
click
value
click
value
click
value
click
choices
selection


speed controller
system
user
actual Speed
actual Mode
desired Speed
desired Mode
mode
status
actual
actual
increment
decrement
current
toggle
switch
desired
value
value
value
value
selection
value
click
value
click
value
click
value
click
choices
selection
value
selection

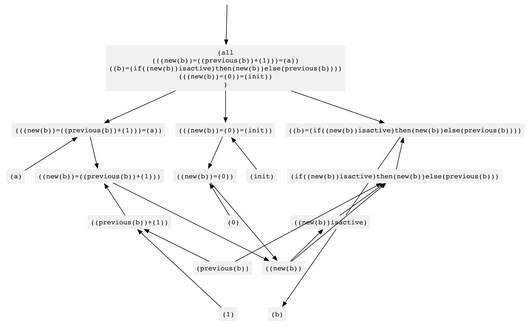

(all
(when(a):((new(b))=((previous(b))+(1))))
(flow(b))
)
(flow(b))
(when(a):((new(b))=((previous(b))+(1))))
(a)
((new(b))=((previous(b))+(1)))
((new(b))
((previous(b))+(1))
(b)
(1)
(previous(b))


(all
(((new(b))=((previous(b))+(1)))=(a))
((b)=(if((new(b))isactive)then(new(b))else(previous(b))))
(((new(b))=(0))=(init))
)
(((new(b))=((previous(b))+(1)))=(a))
(((new(b))=(0))=(init))
((b)=(if((new(b))isactive)then(new(b))else(previous(b))))
(a)
((new(b))=((previous(b))+(1)))
((new(b))=(0))
(init)
(if((new(b))isactive)then(new(b))else(previous(b)))
((previous(b))+(1))
(0)
((new(b))isactive)
(previous(b))
((new(b))
(1)
(b)

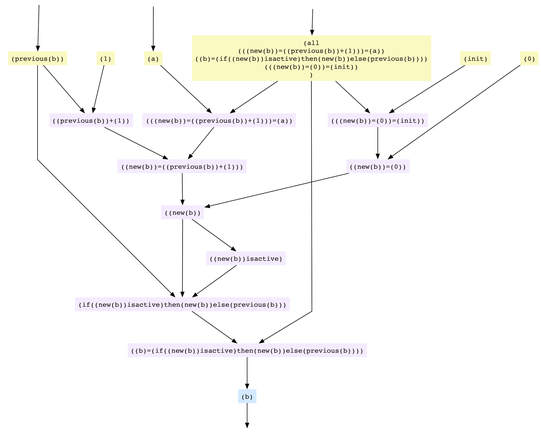
Parse



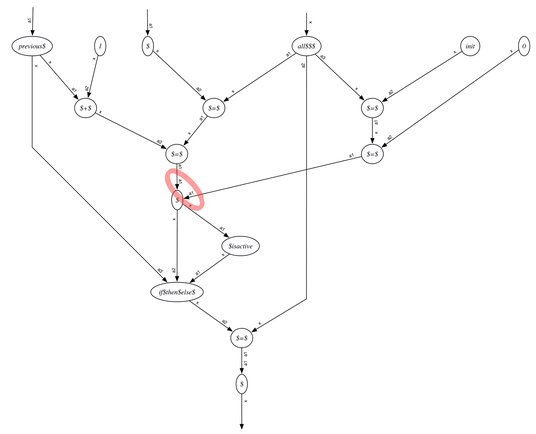
Expand



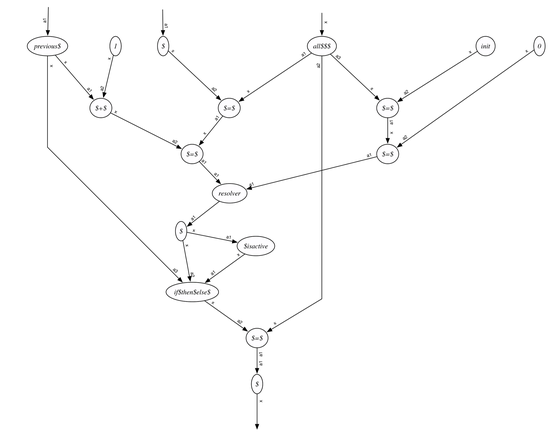
Order



Instantiate

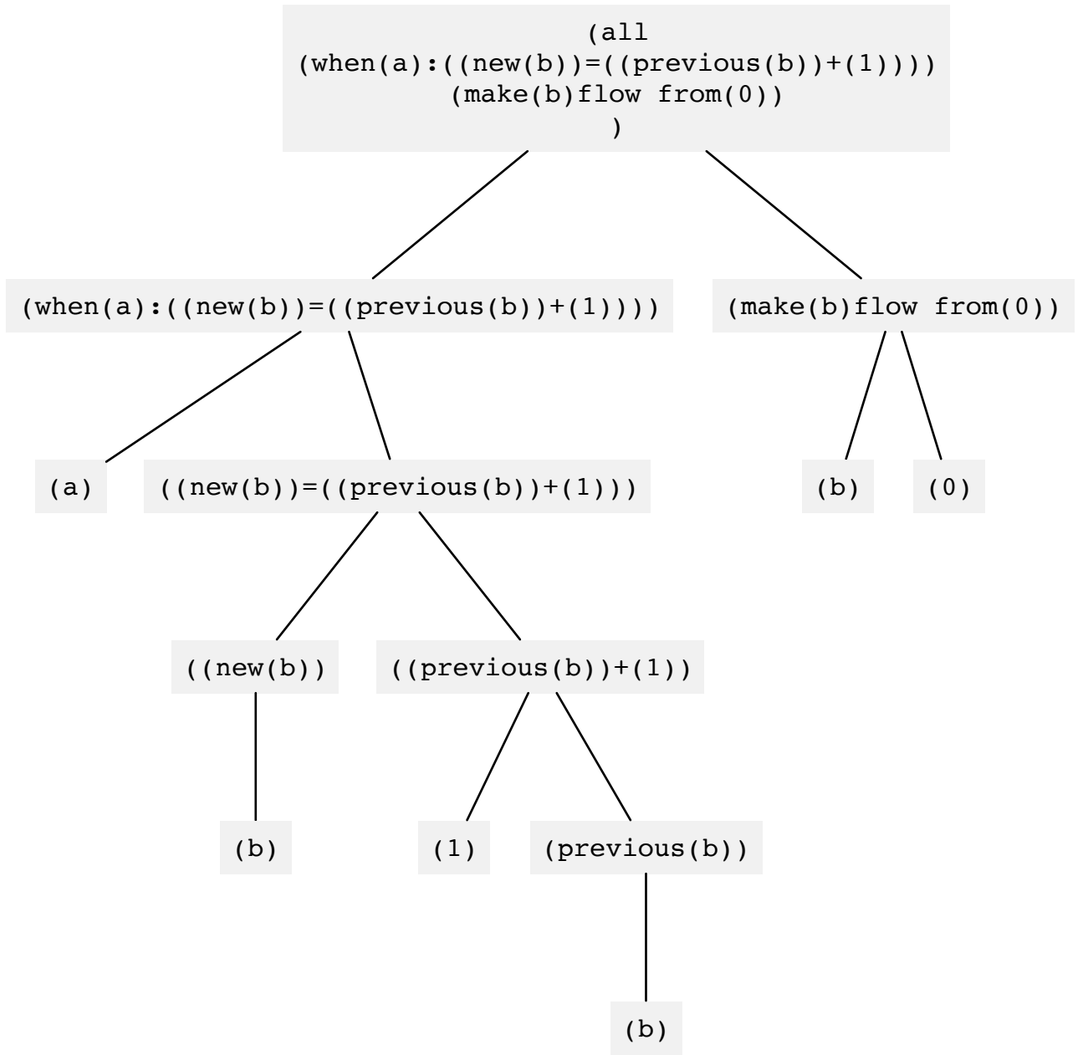


Resolve



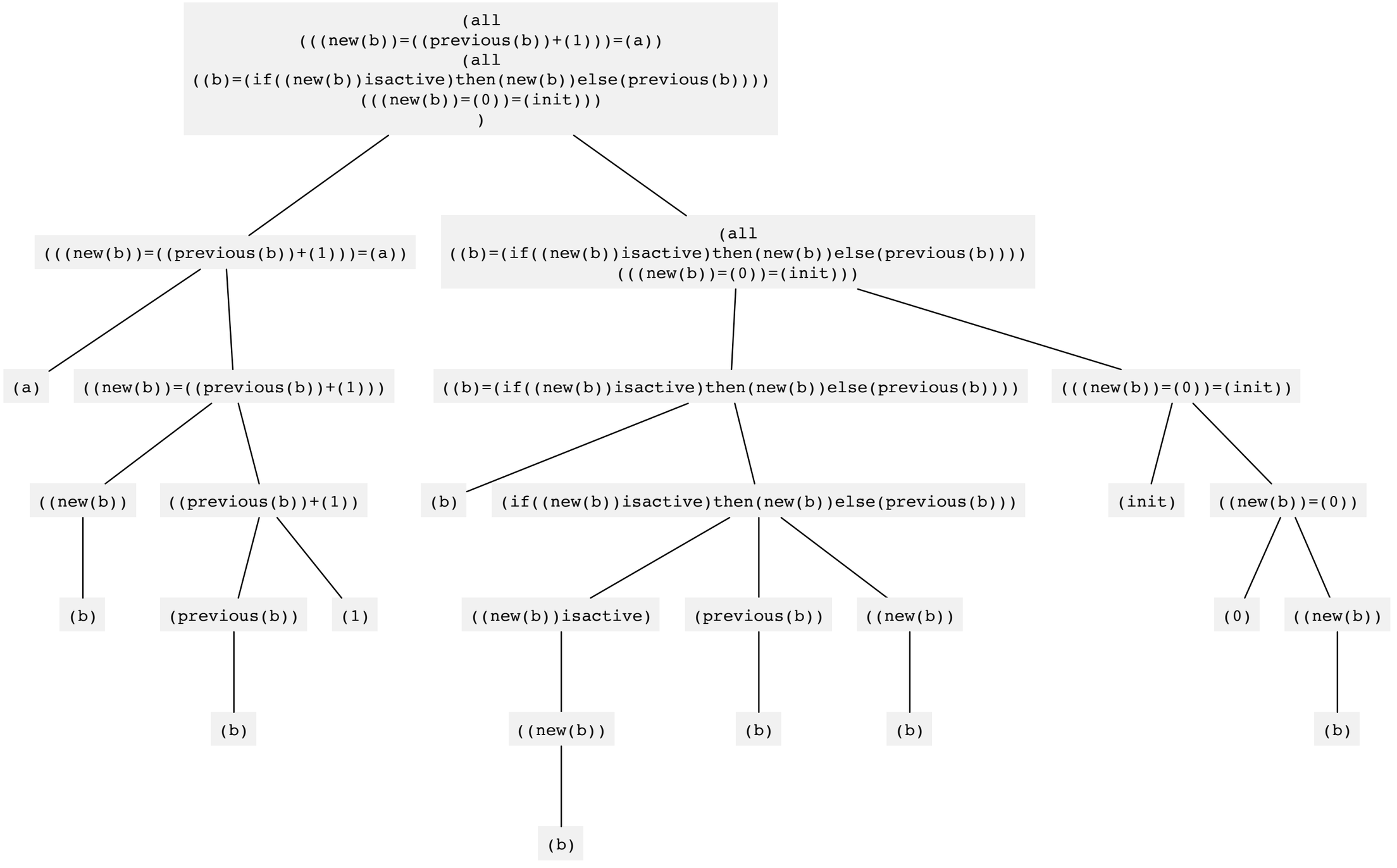
Parse

Here we parse the code and made the expression tree



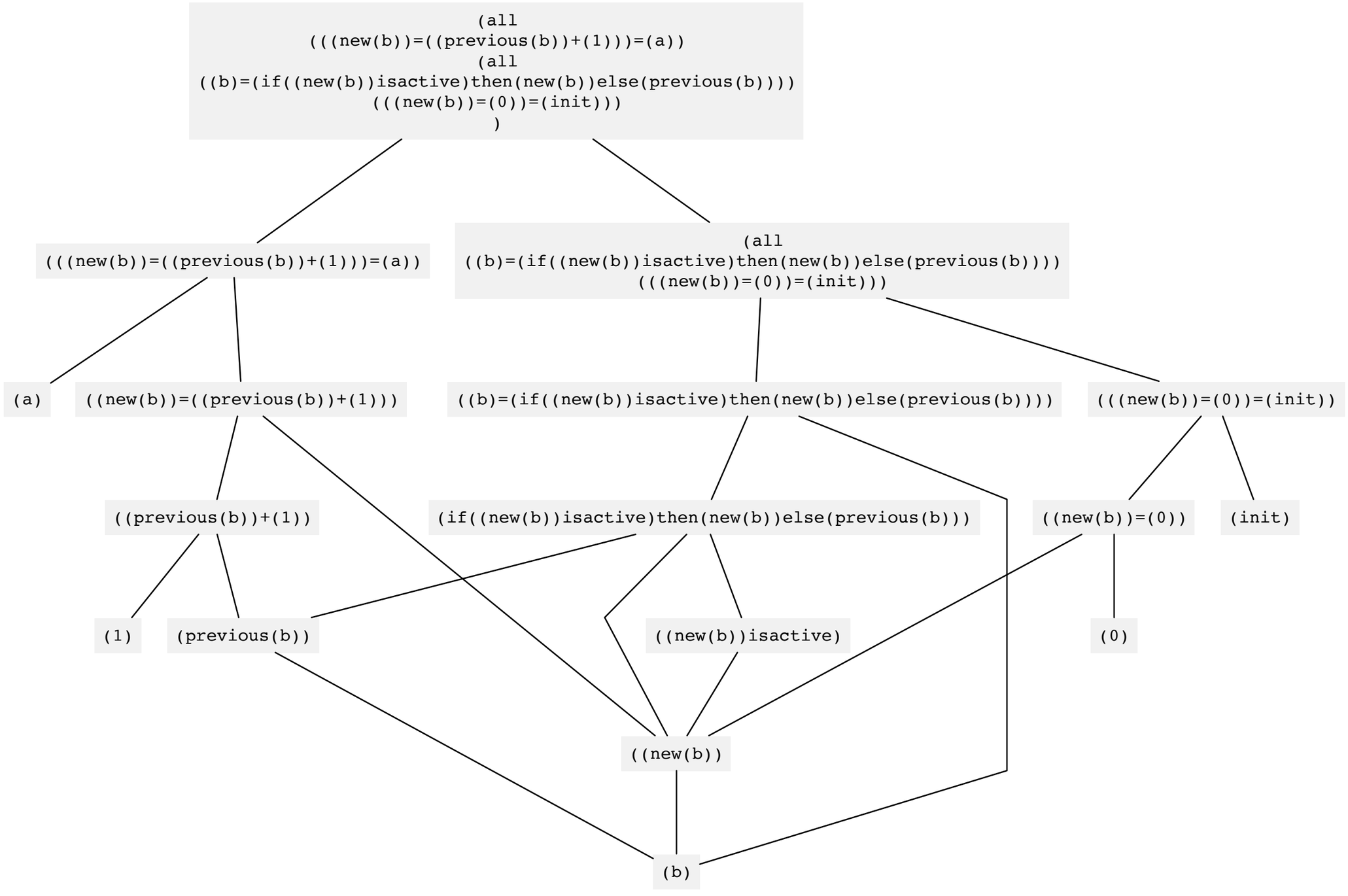
Expand

Then we expand the tree, using the definitions of interactions, stopping when all interactions are native interactions



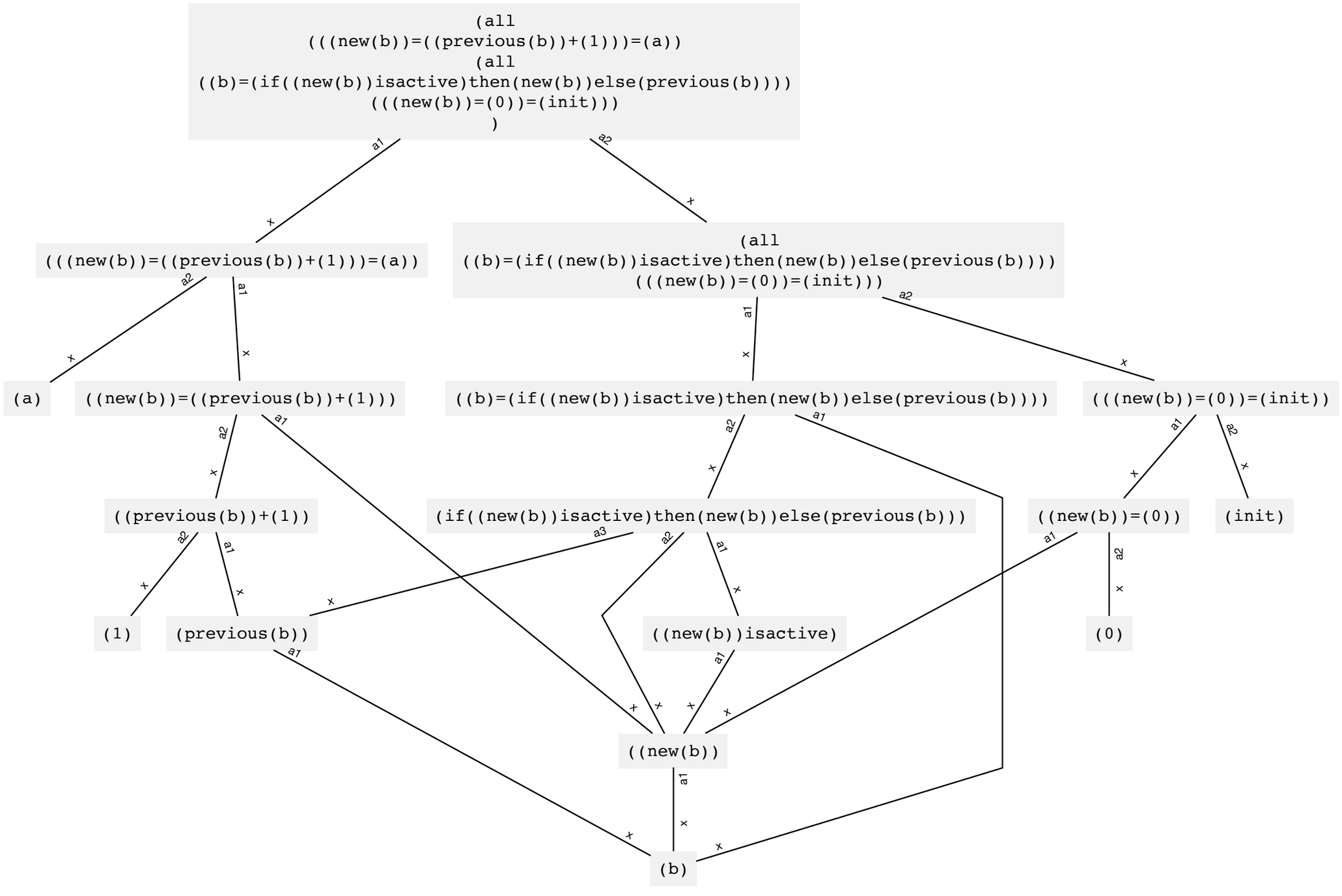
Link

Here we removed duplicate nodes, effectively linking references to the same node



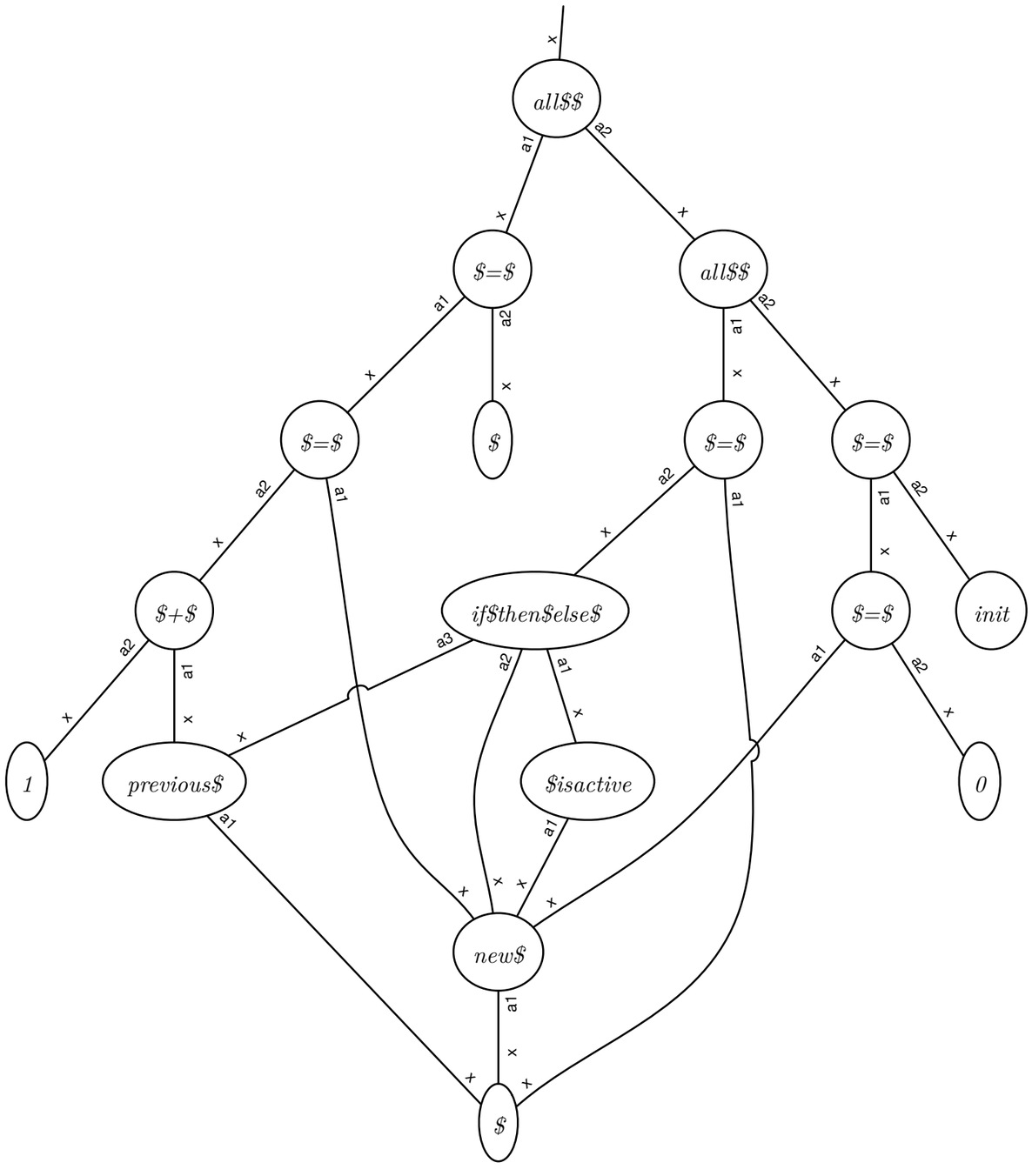
Annotate

Here we added annotations to the graph edges, these annotations describe the role of each element in the expression
(a1 for argument1, a2, …, and x for the expression itself)
Note that adding annotations is straightforward, as the bottom end of each edge is an “x”, and the top end is an “a” numbered according to the argument number.



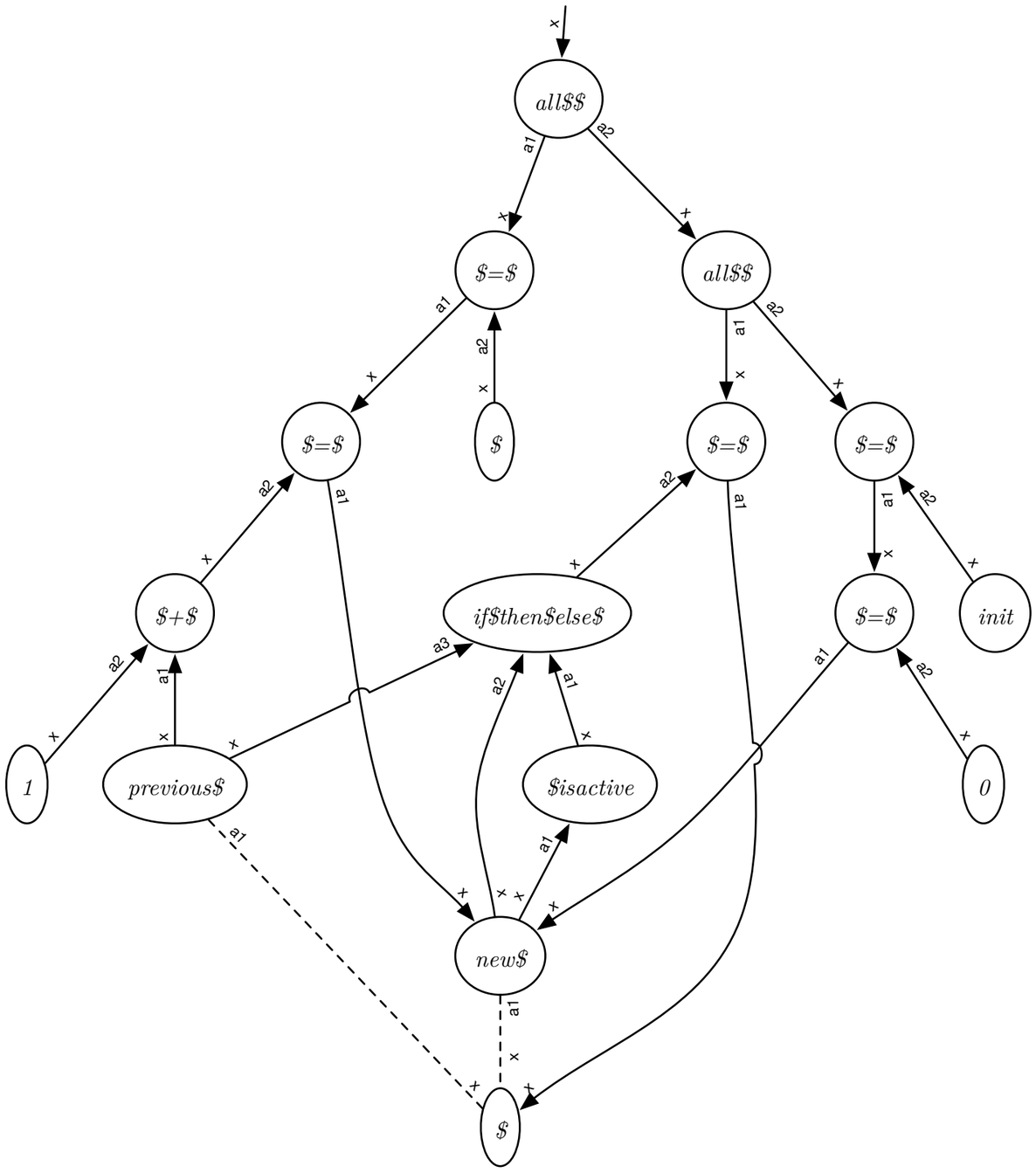
Instantiate

From here we don’t need identifiers anymore, we can remove the notion of identifier, and rename nodes according to their operator instead of their identifier



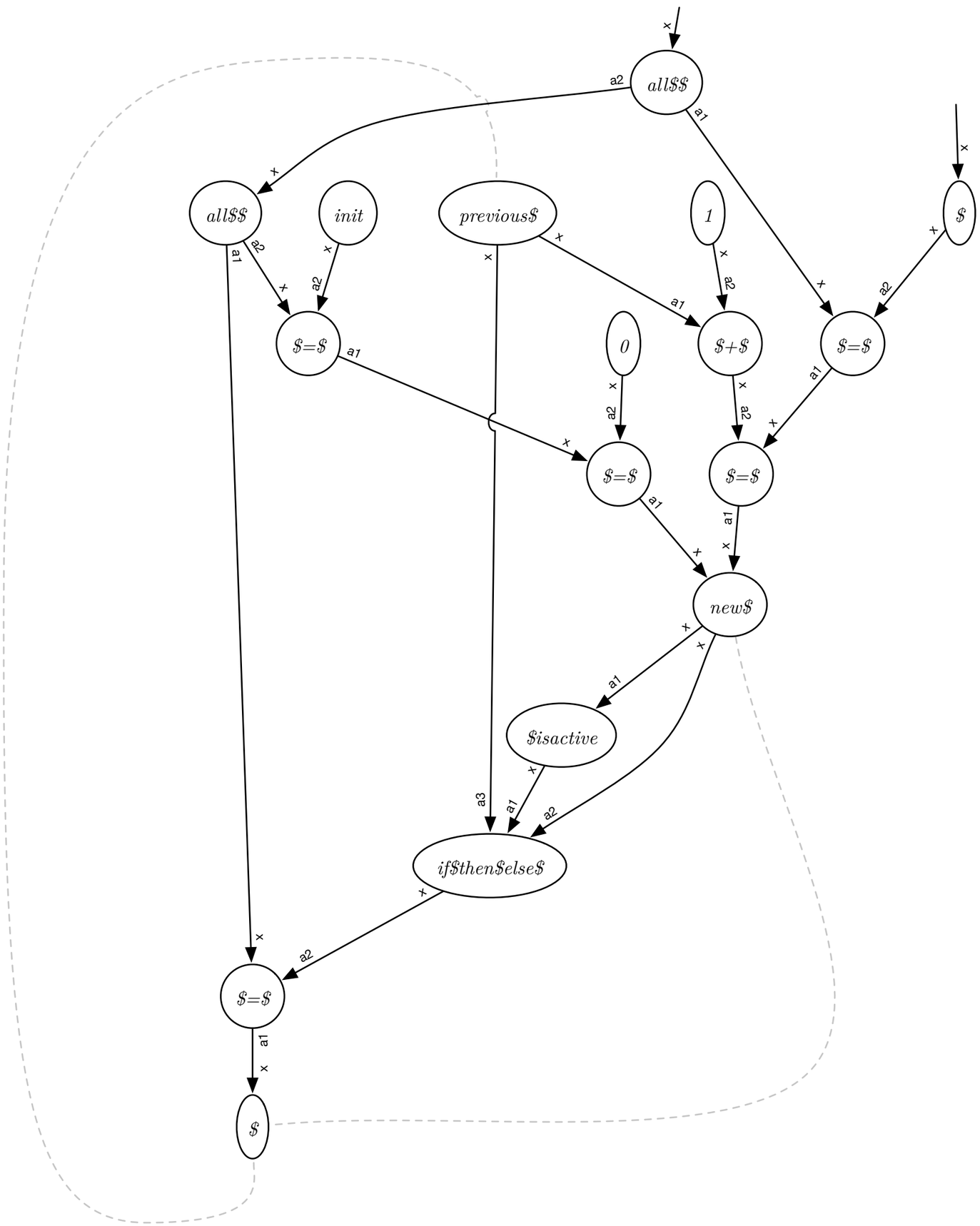
Direct

Here we keep the same graph, but we give directions to edges, according to the direction of the data flow. For this step to be possible, the program has to be consistent according to interfaces flow directions. (i.e. the OUTs must match the INs)
Some edges do not denote a data flow, so they are dashed



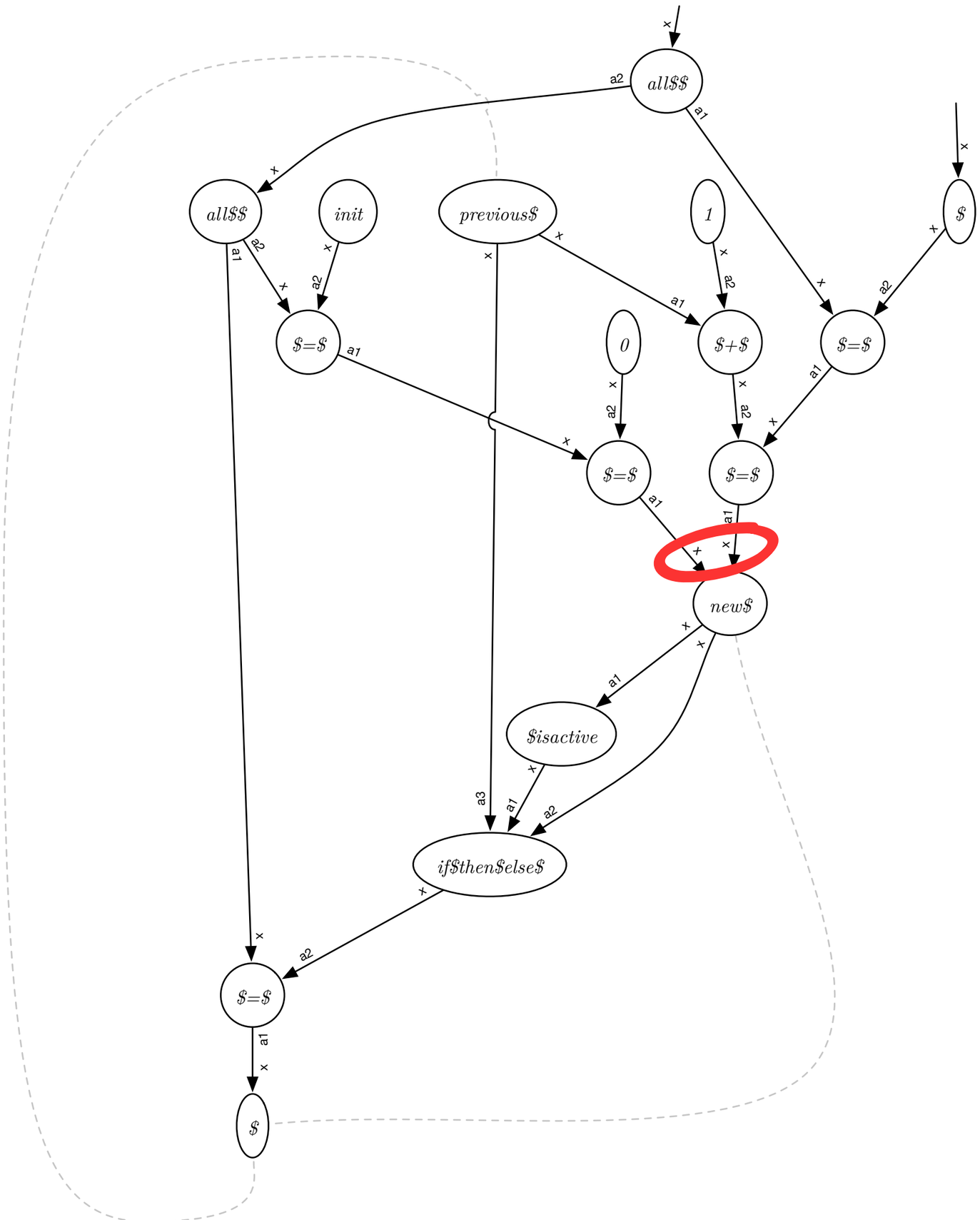
Order

Again, this is the same graph, but we changed its layout so all arrows go down



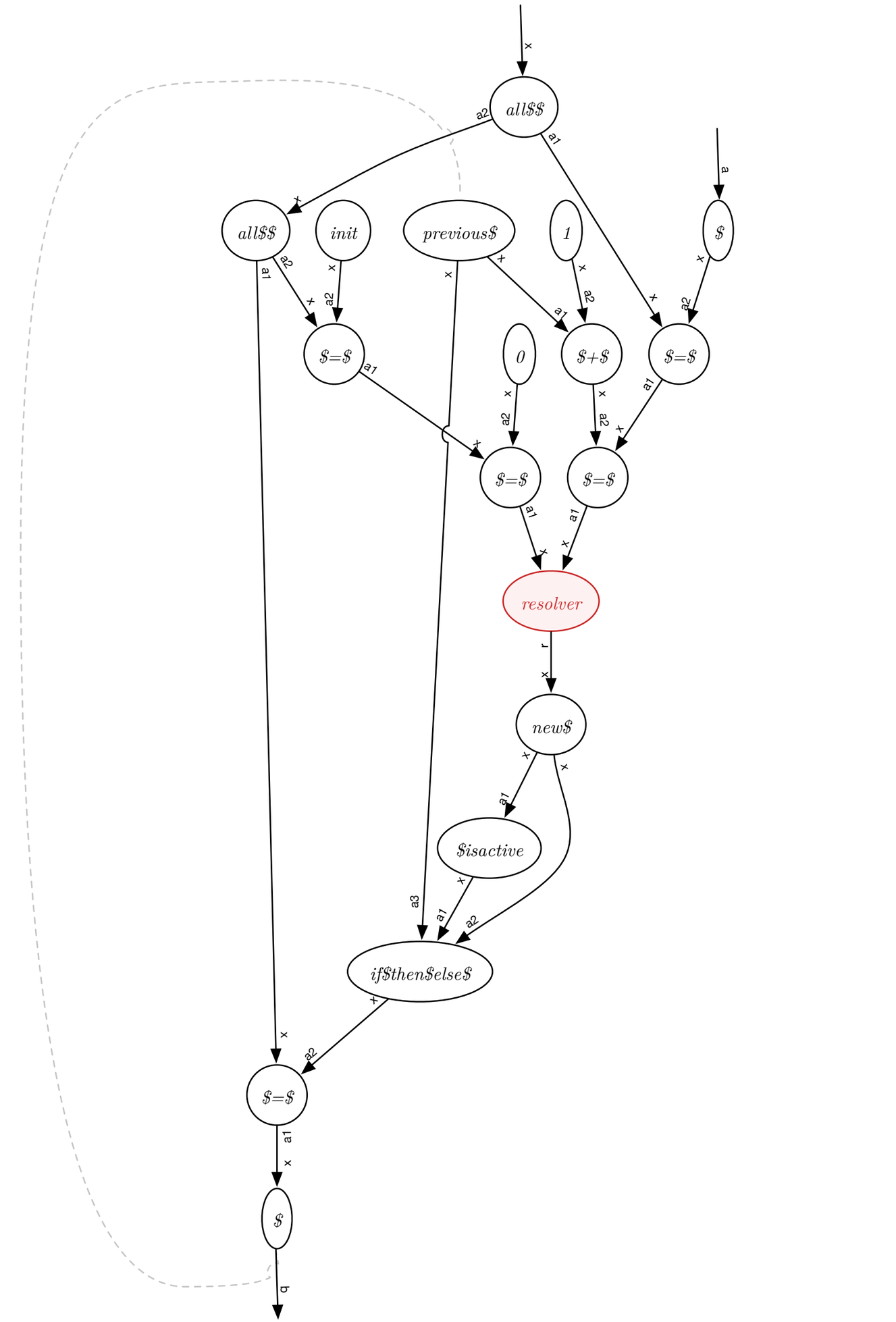
Detect conflicts

We detect nodes where two incoming edges have the same annotations



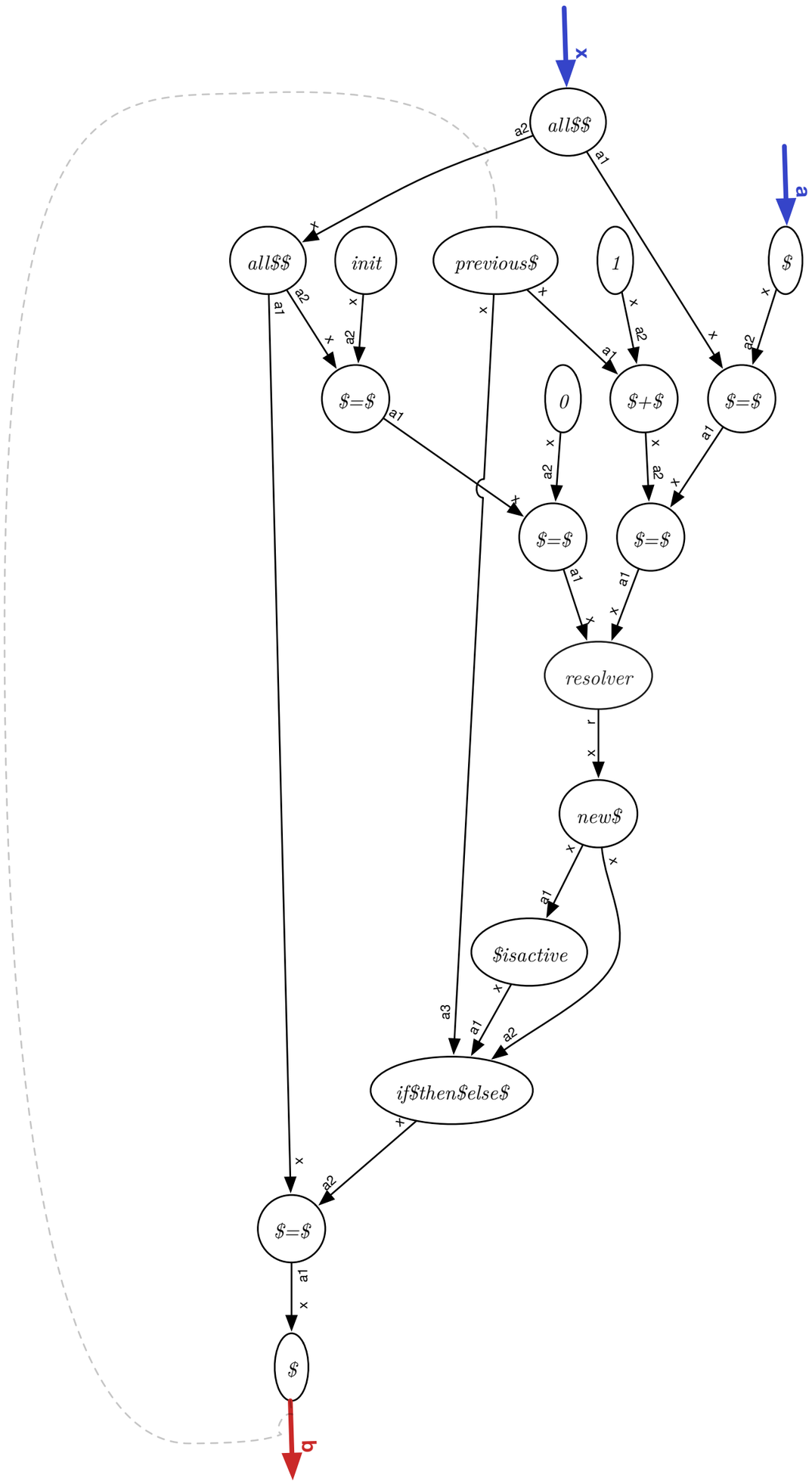
Resolve conflicts

Add a resolver where conflicts exist



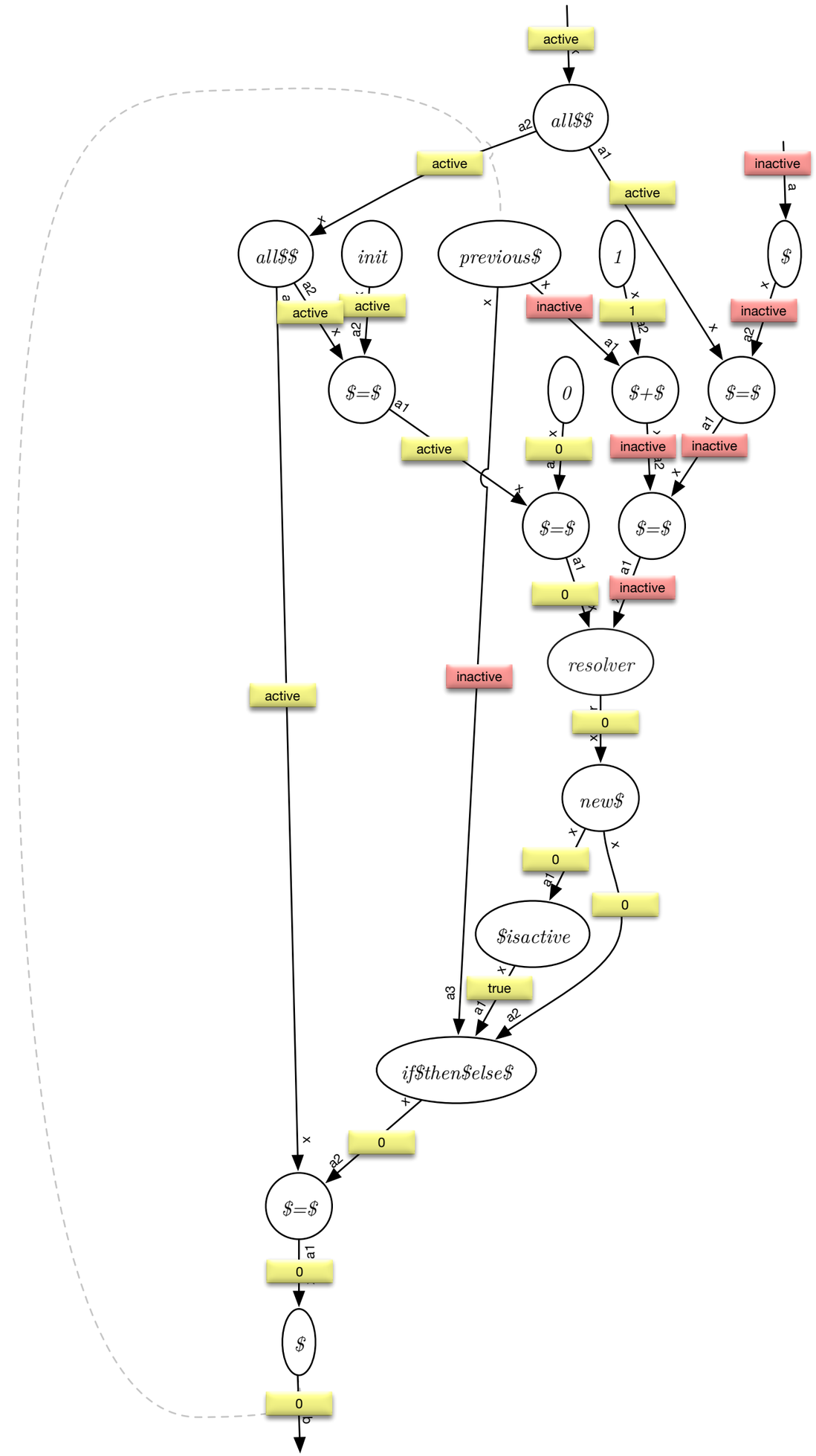
Execution

From here, the interaction is ready to be executed.
To execute, assign values to edges, from top to bottom.
System inputs are incoming edges.
System outputs are outgoing edges.



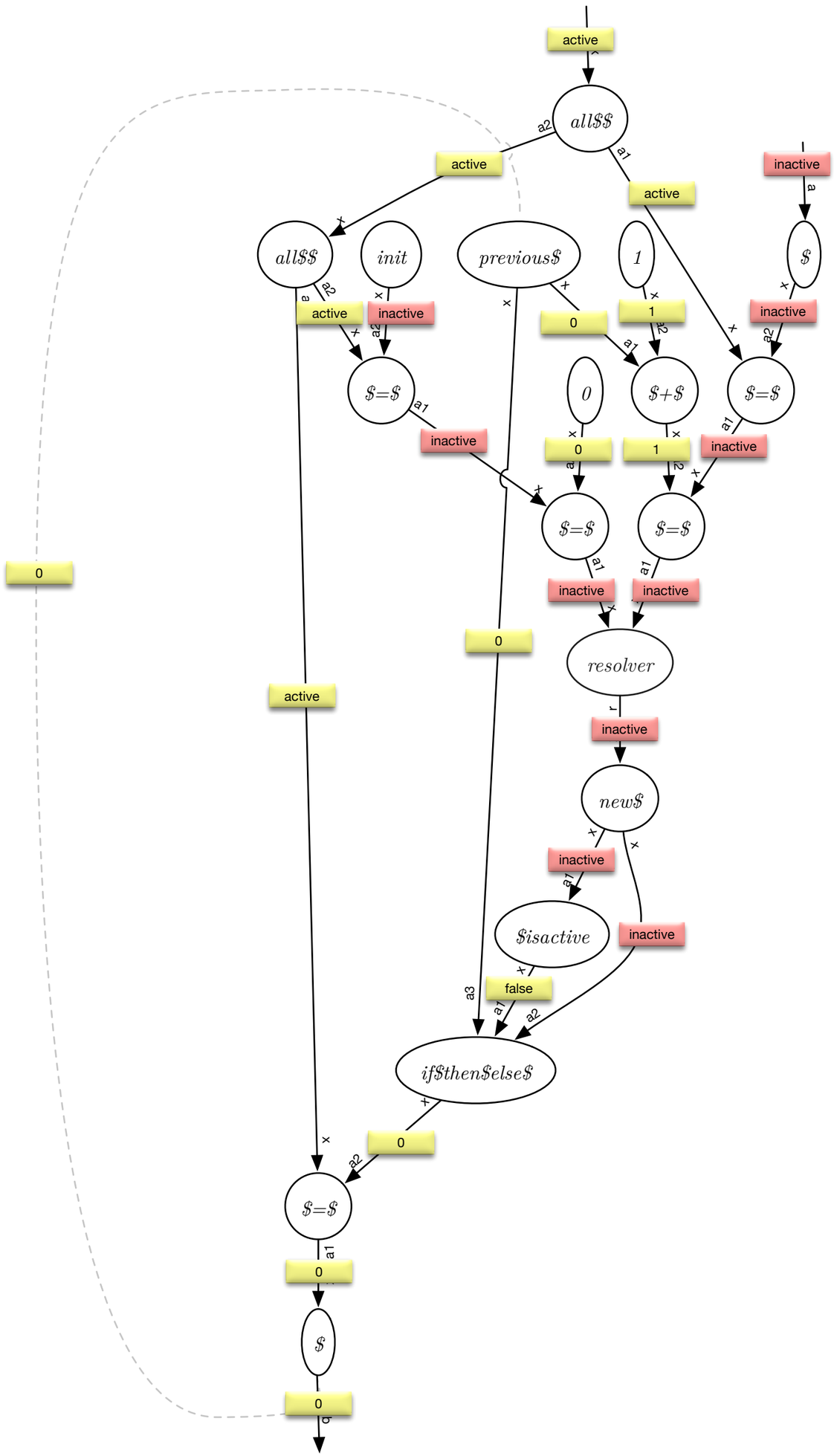
Execute 0

First execution step, so (init) outputs active
Inputs :
a=inactive,
the root interaction is active



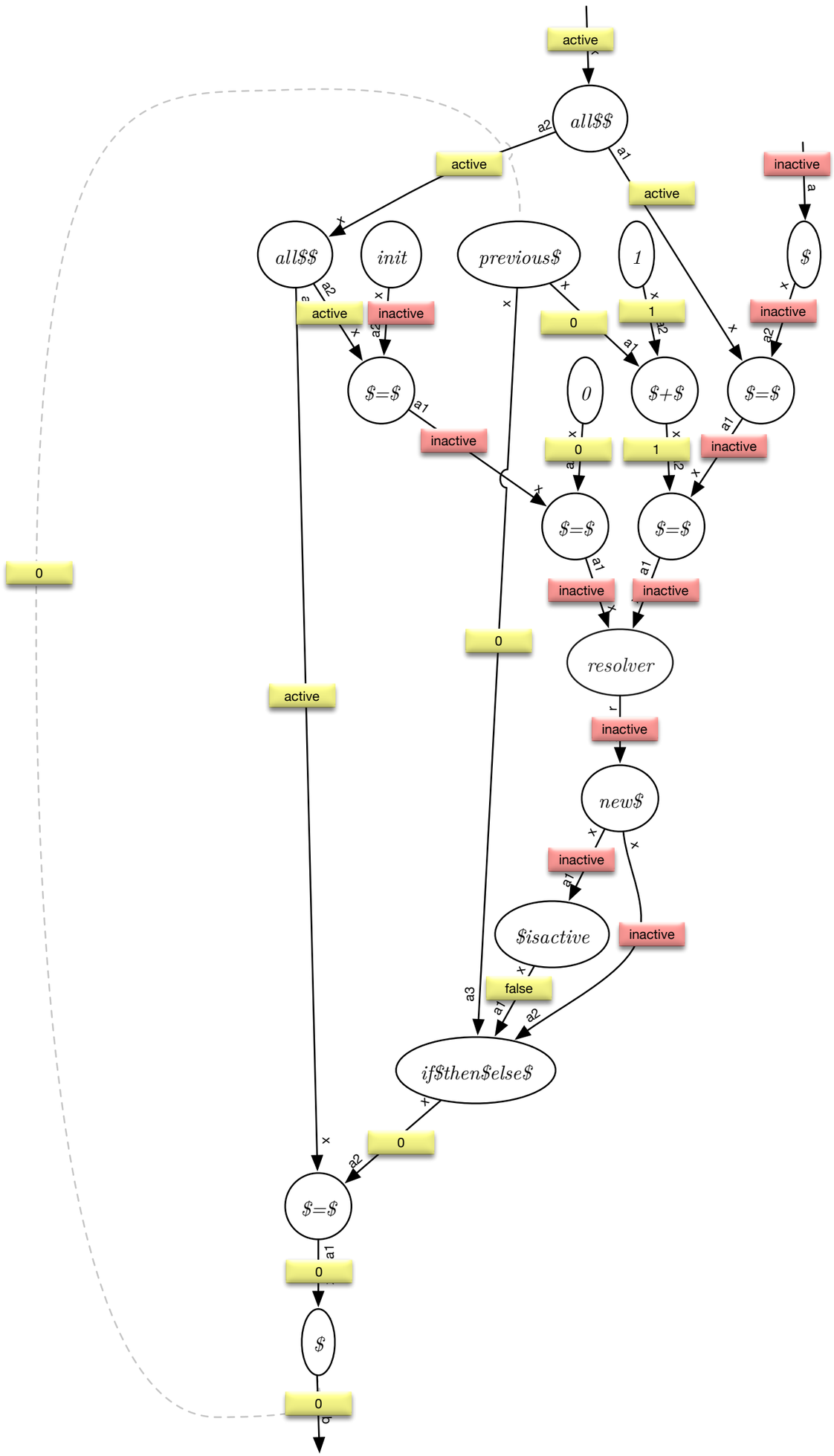
Execute 1

Second execution step,
still no input on a,
But notice how the output is still 0 (not inactive), it “flows”



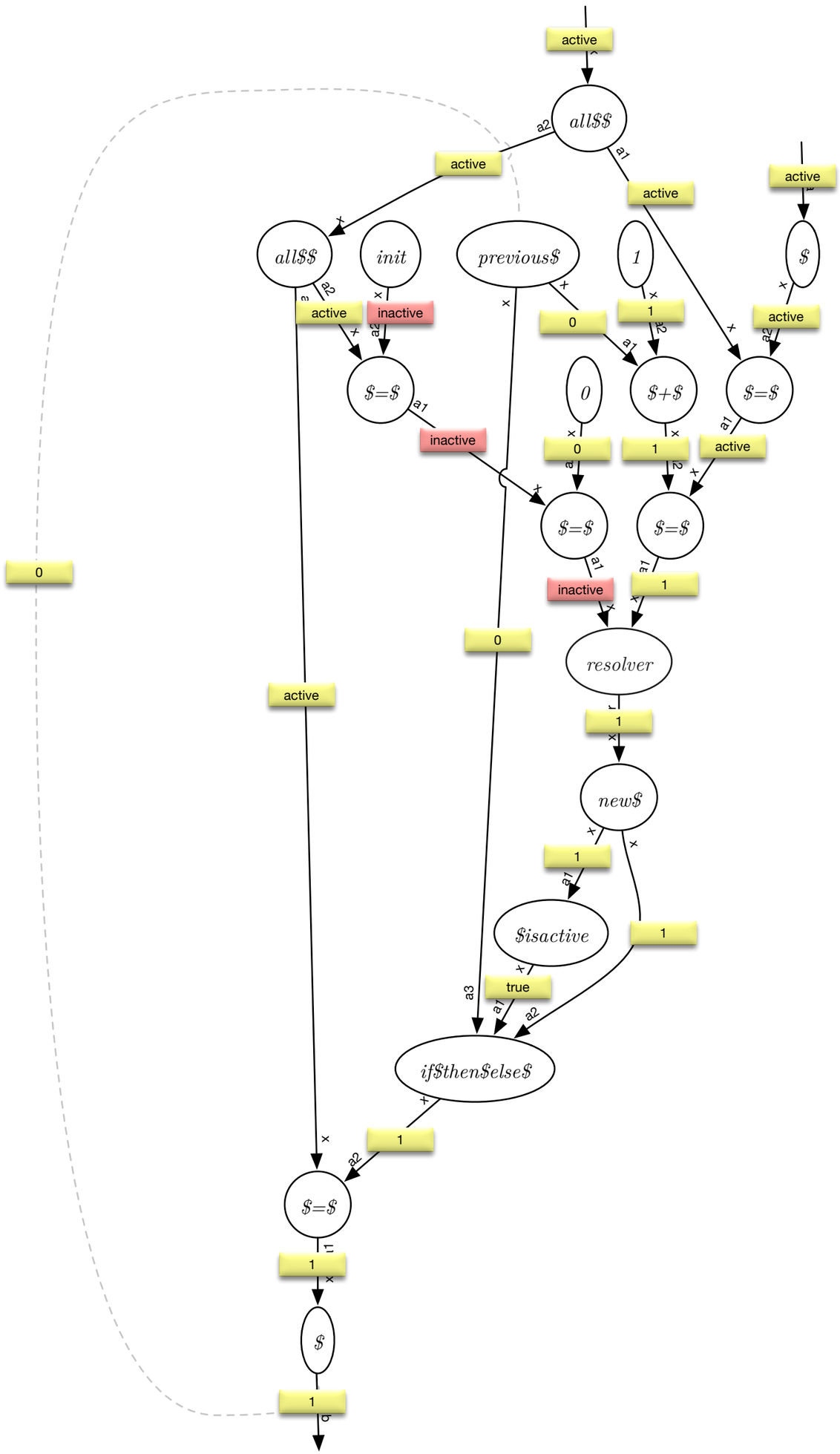
Execute 2

Third execution step, still no input on a, nothing changes



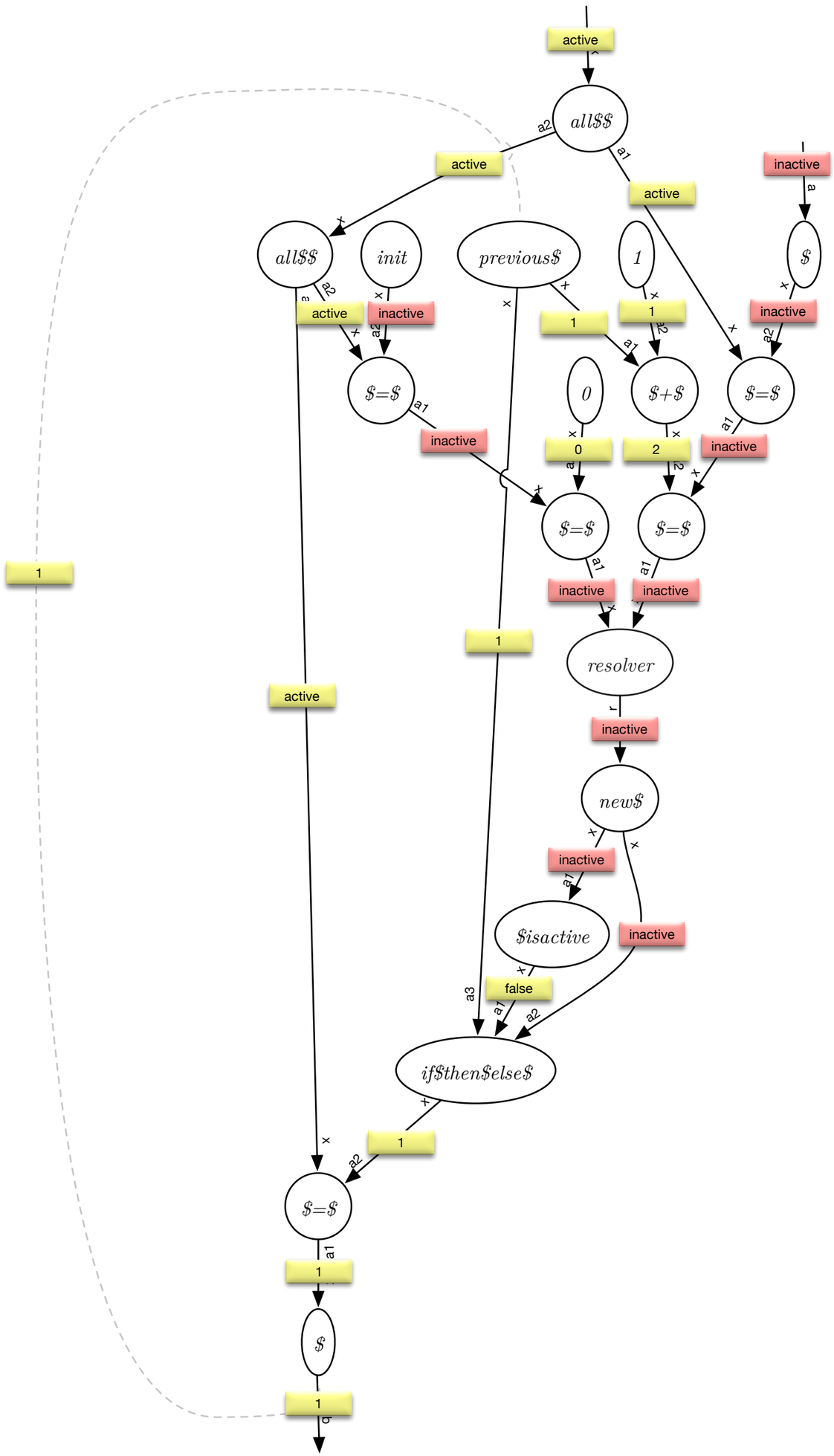
Execute 3

Here the input a receives the value “active”,
the output value b is incremented



Execute 4

No input on a, b stays to 1



Execute 5

Another input “active” on a, b is now 2