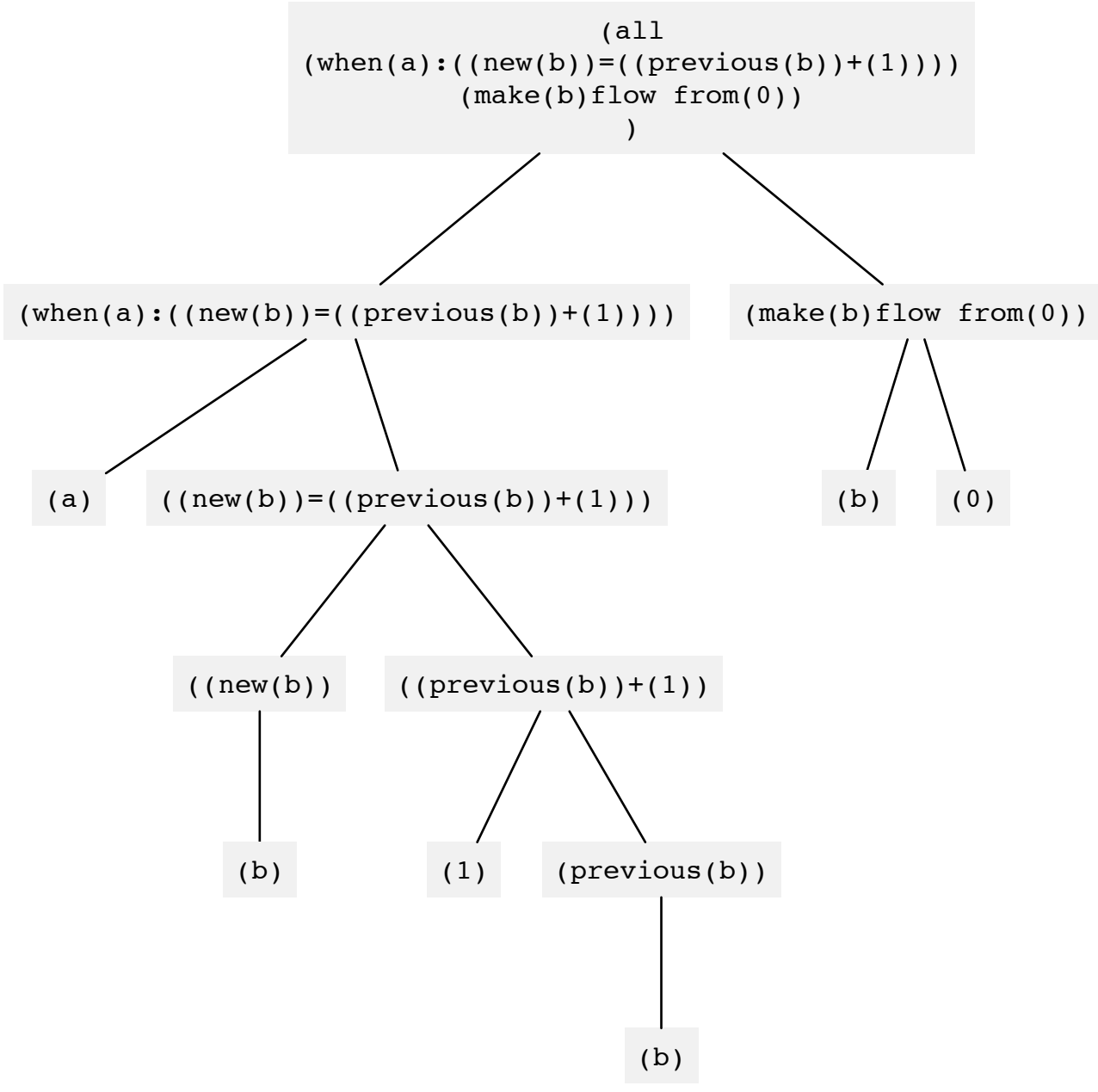
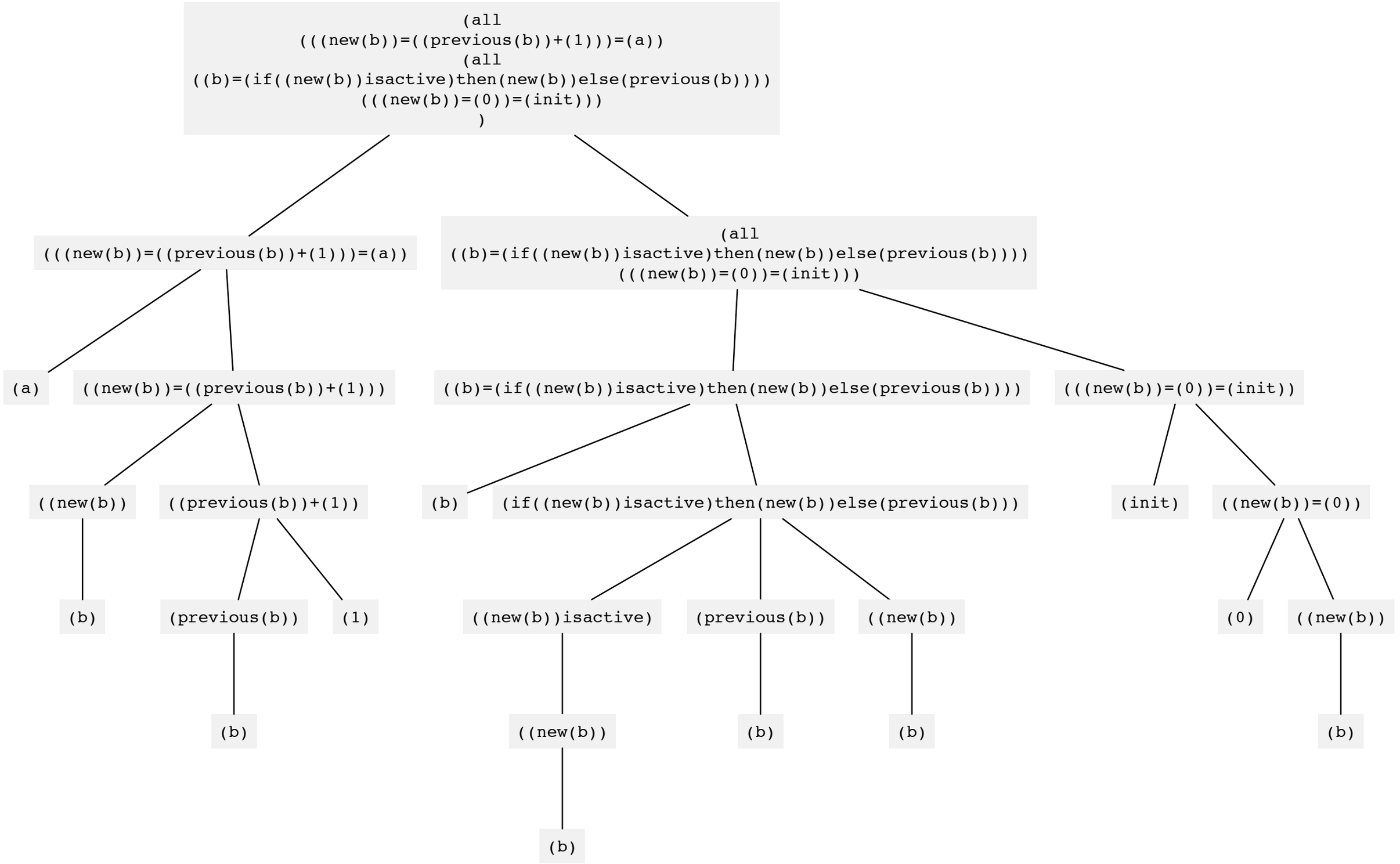
Parse

Here we parse the code and made the expression tree



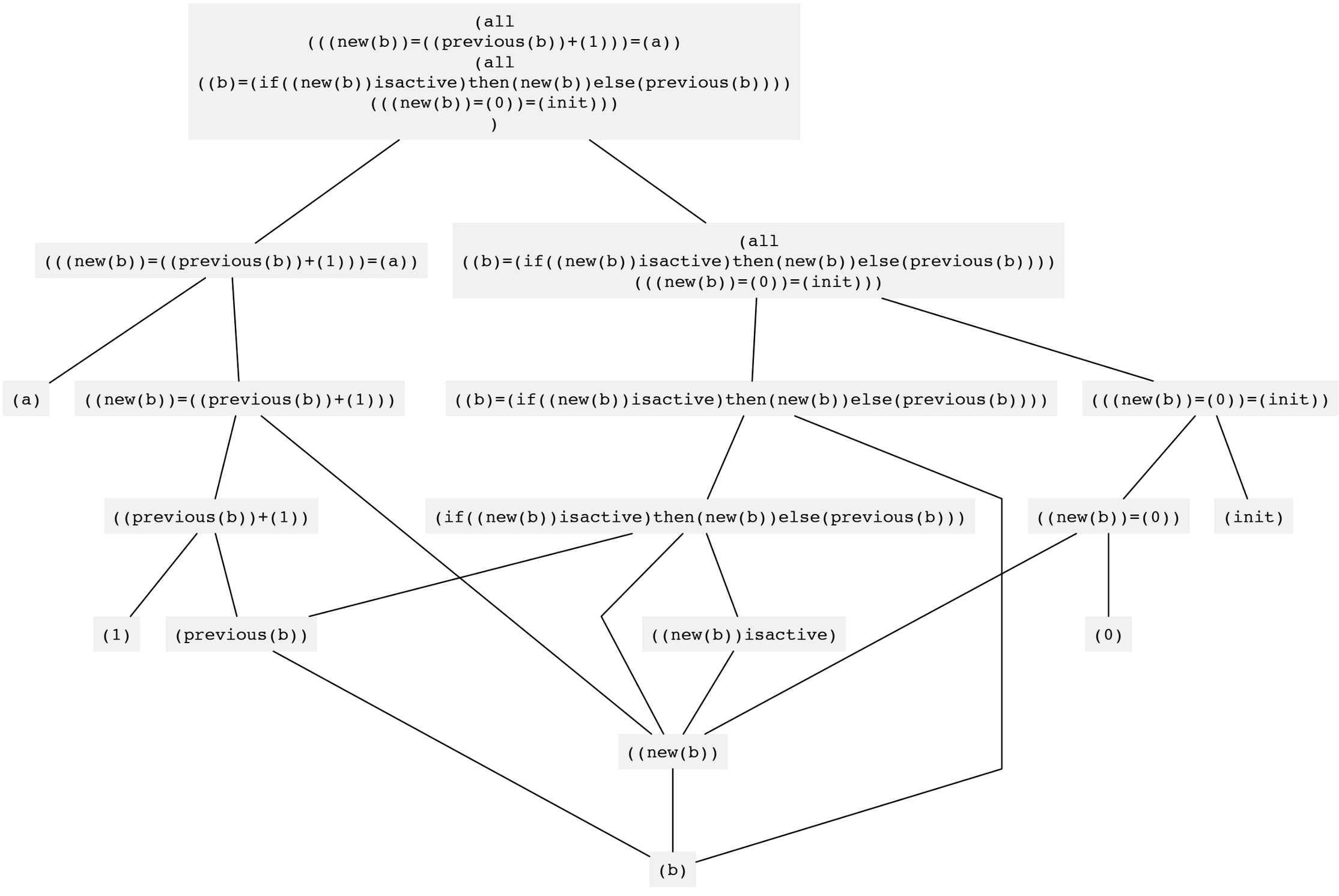
Expand

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



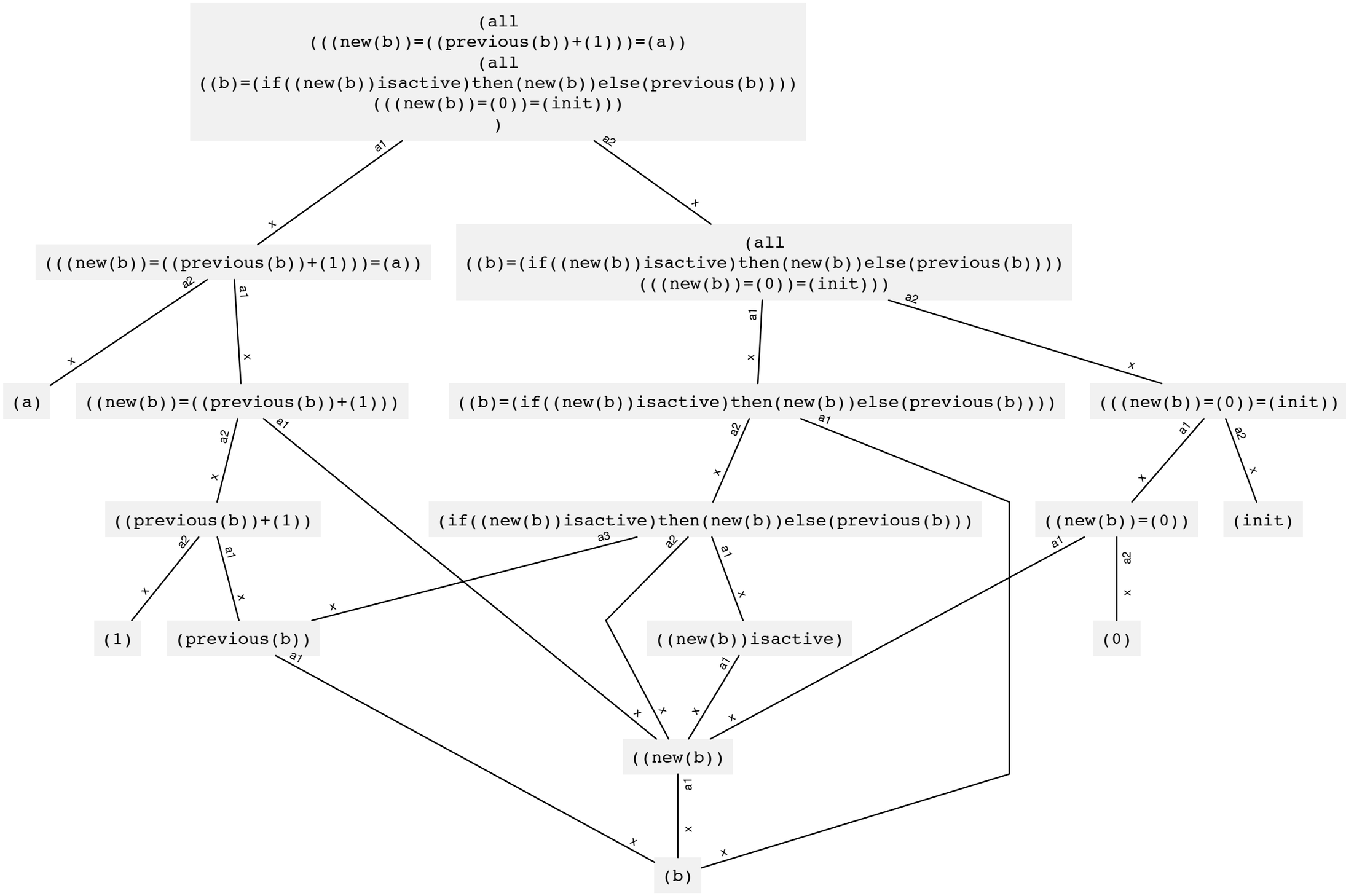
Link

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



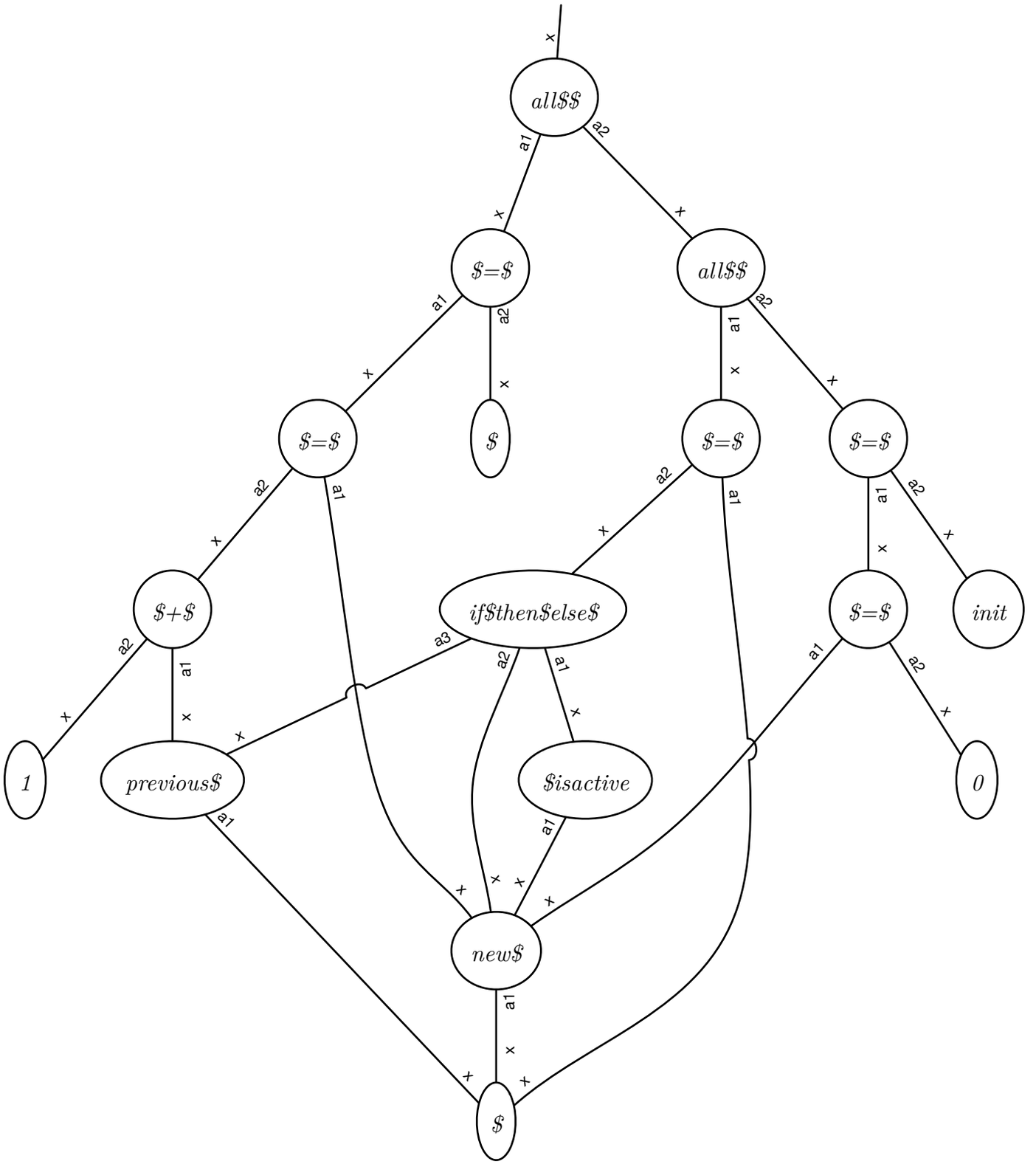
Annotate

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.



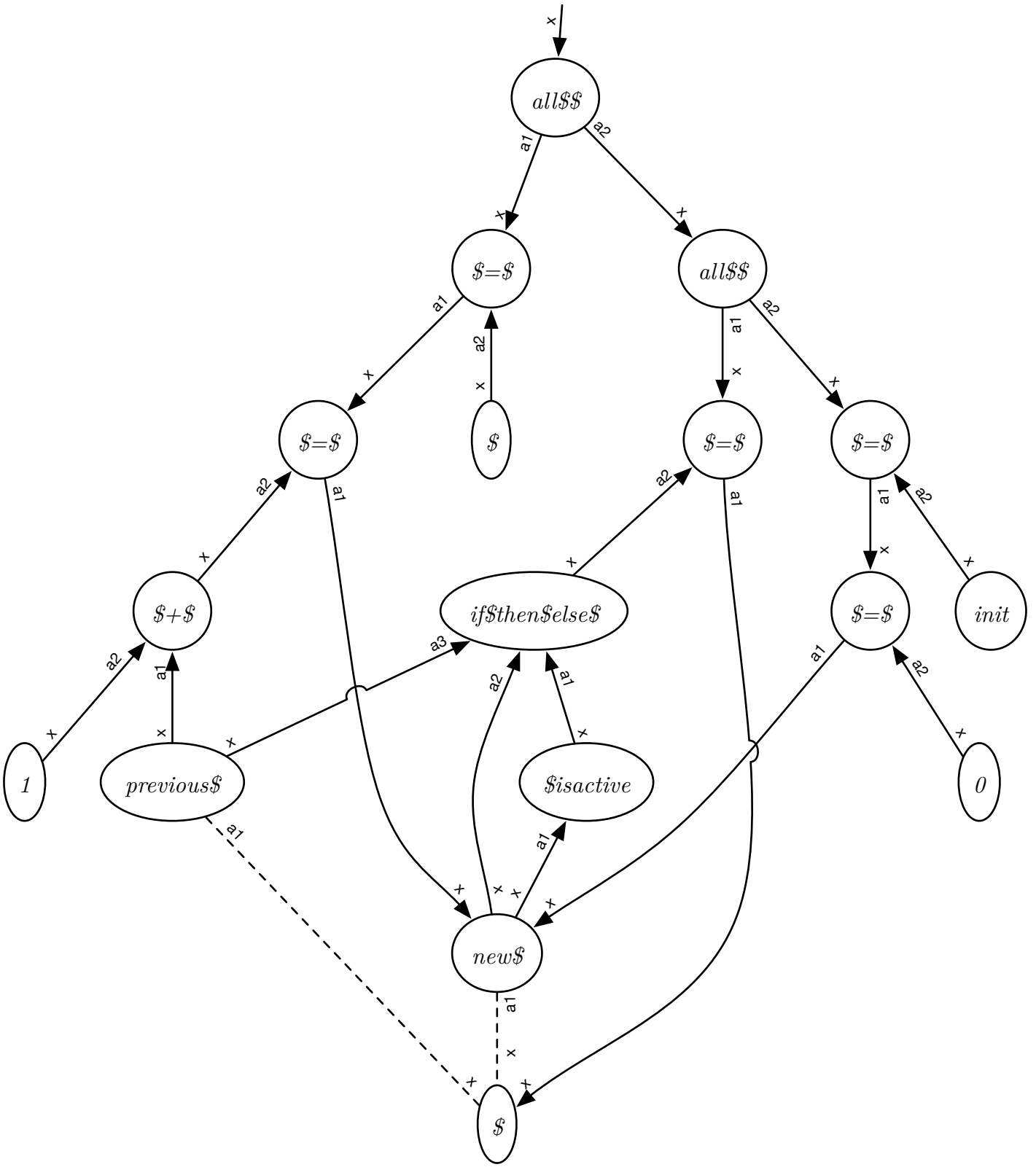
Instantiate

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



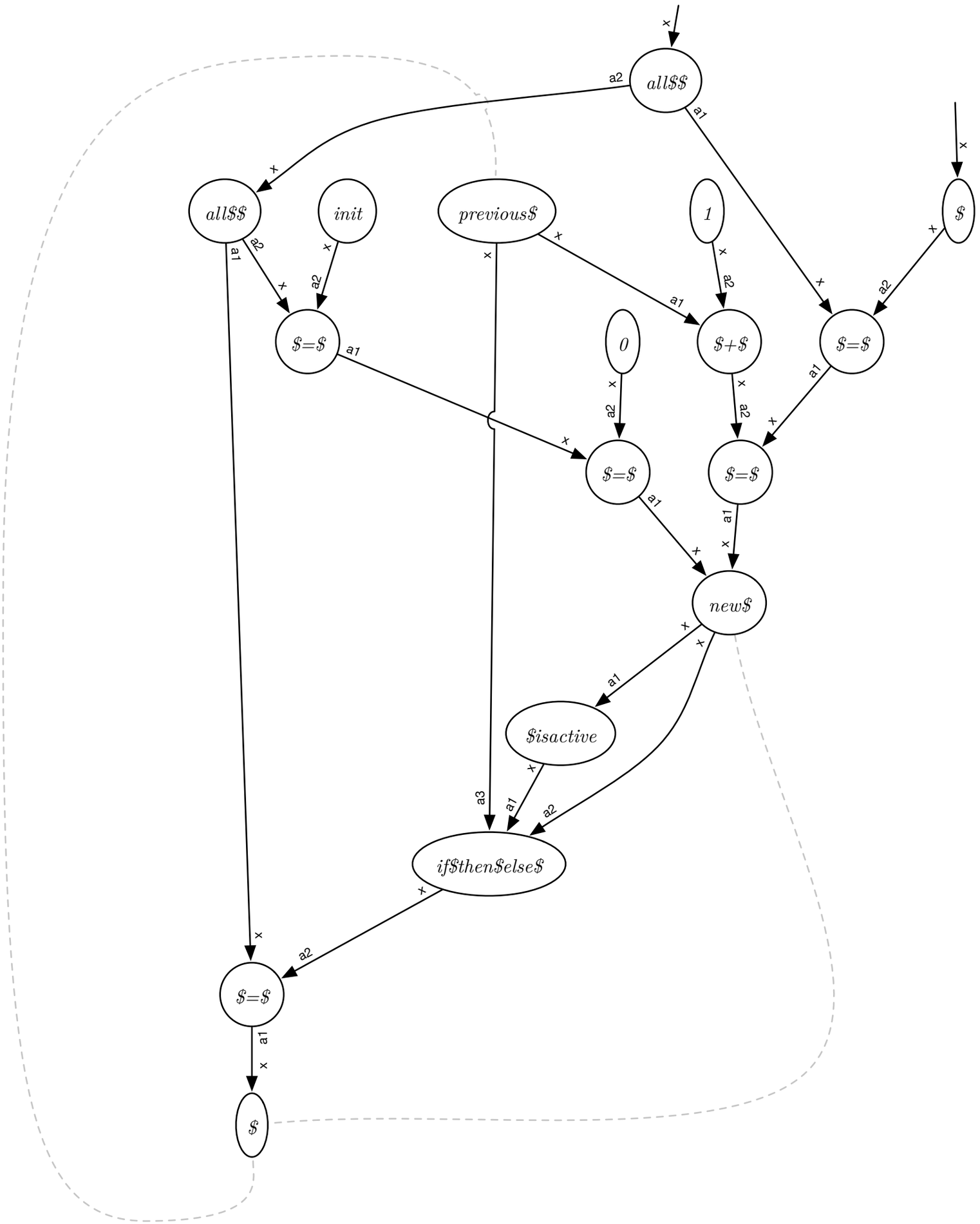
Direct

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



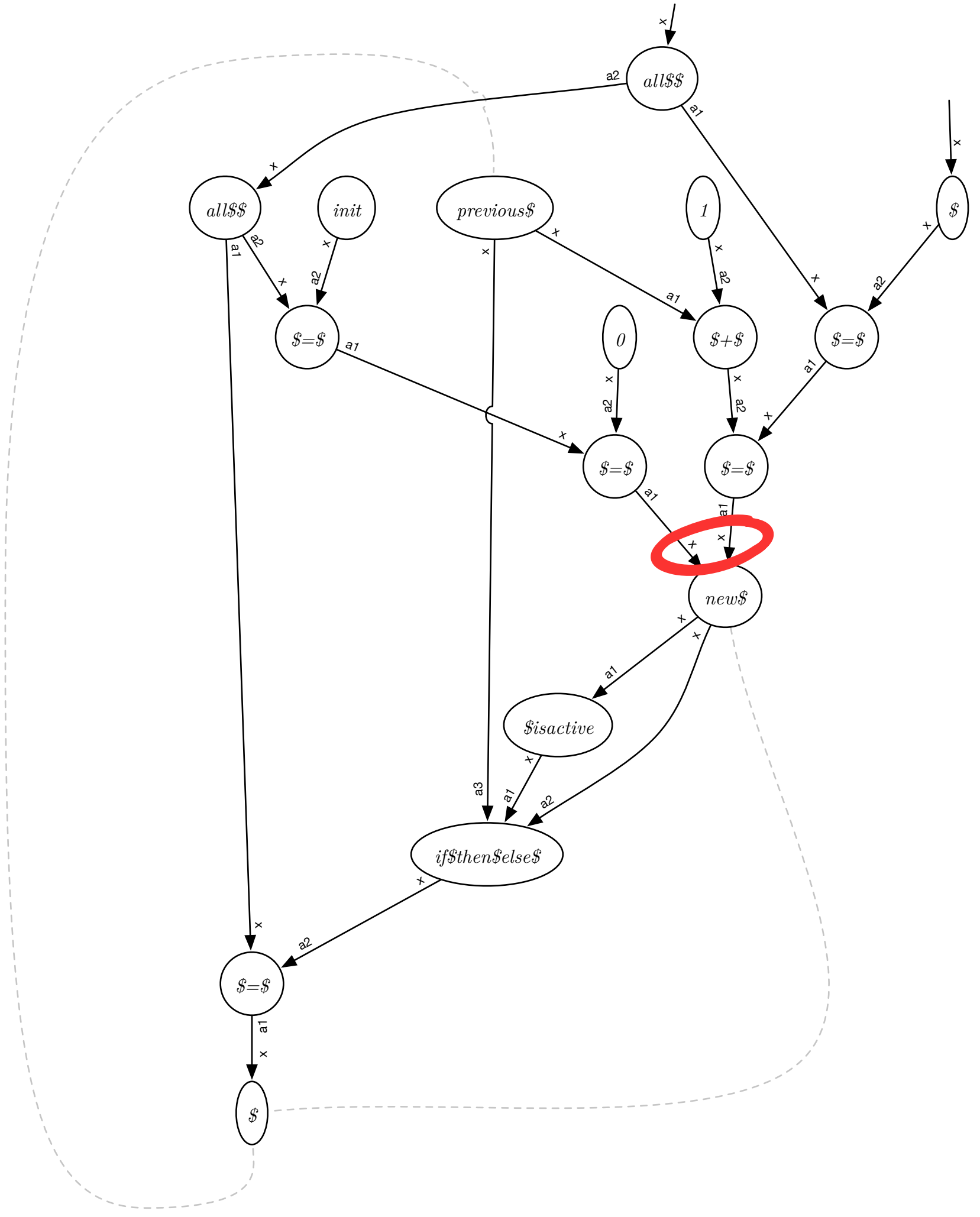
Order

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



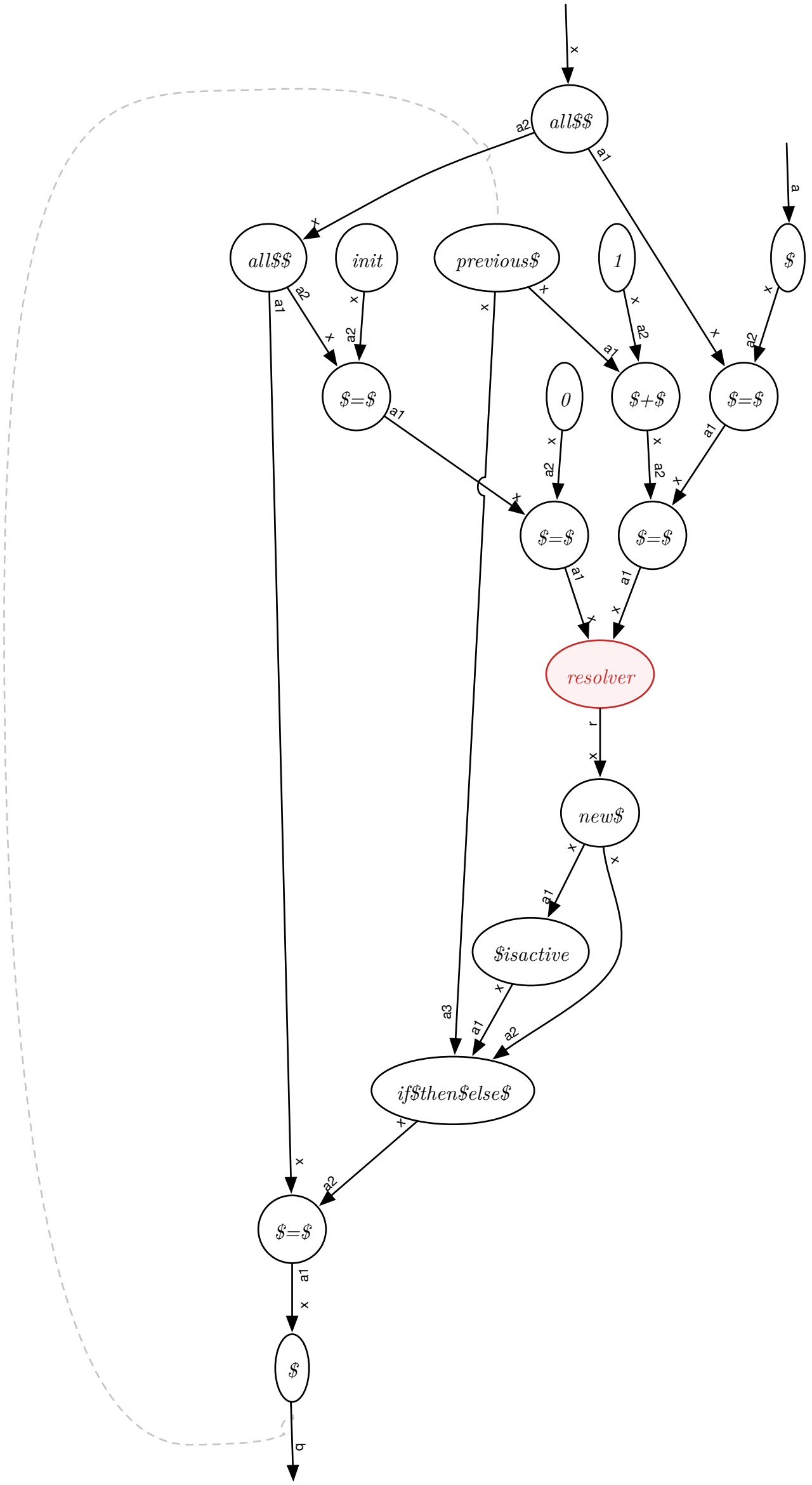
Detect conflicts

We detect nodes where two incoming edges have the same annotations



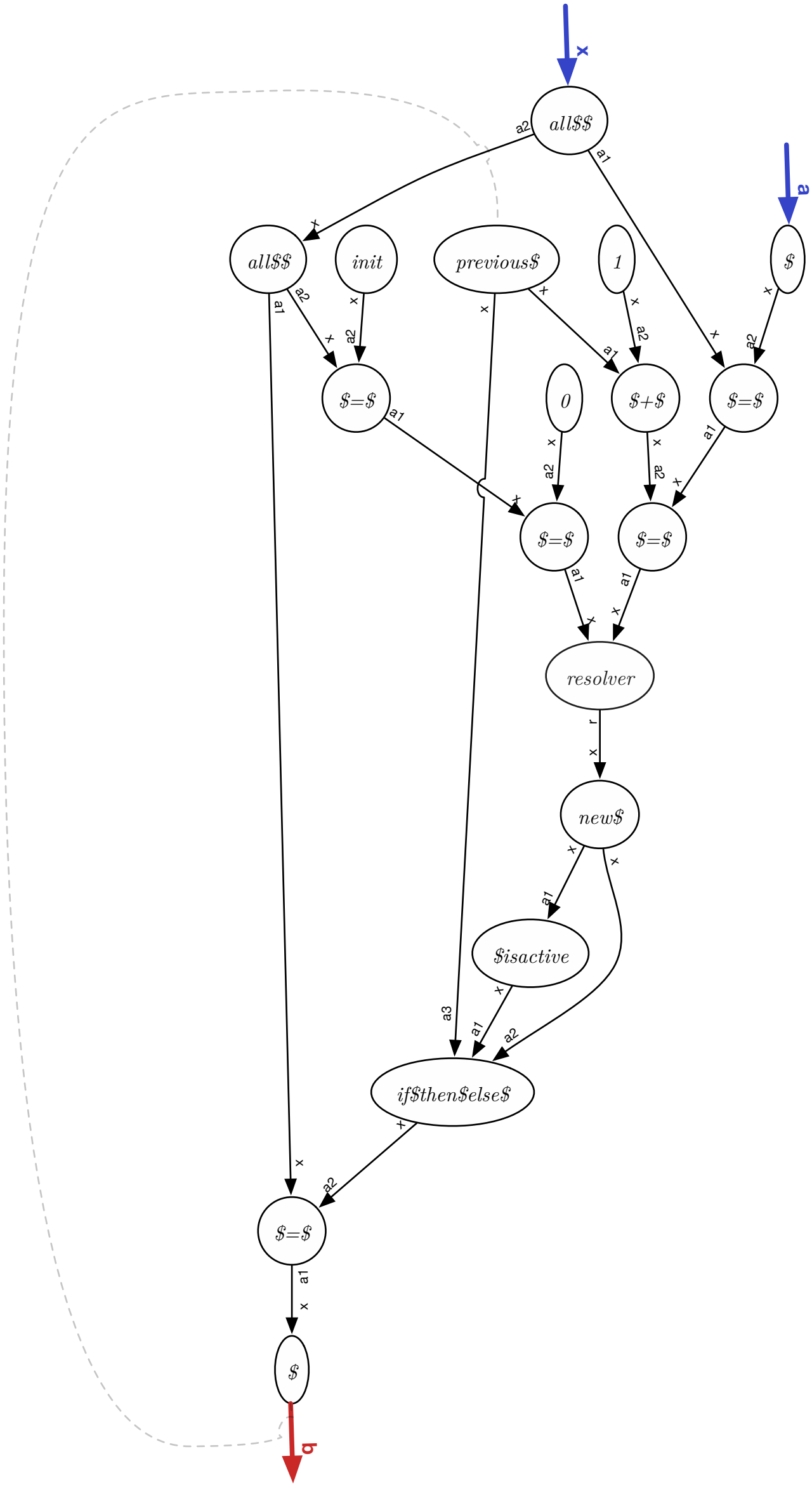
Resolve conflicts

Add a resolver where conflicts exist



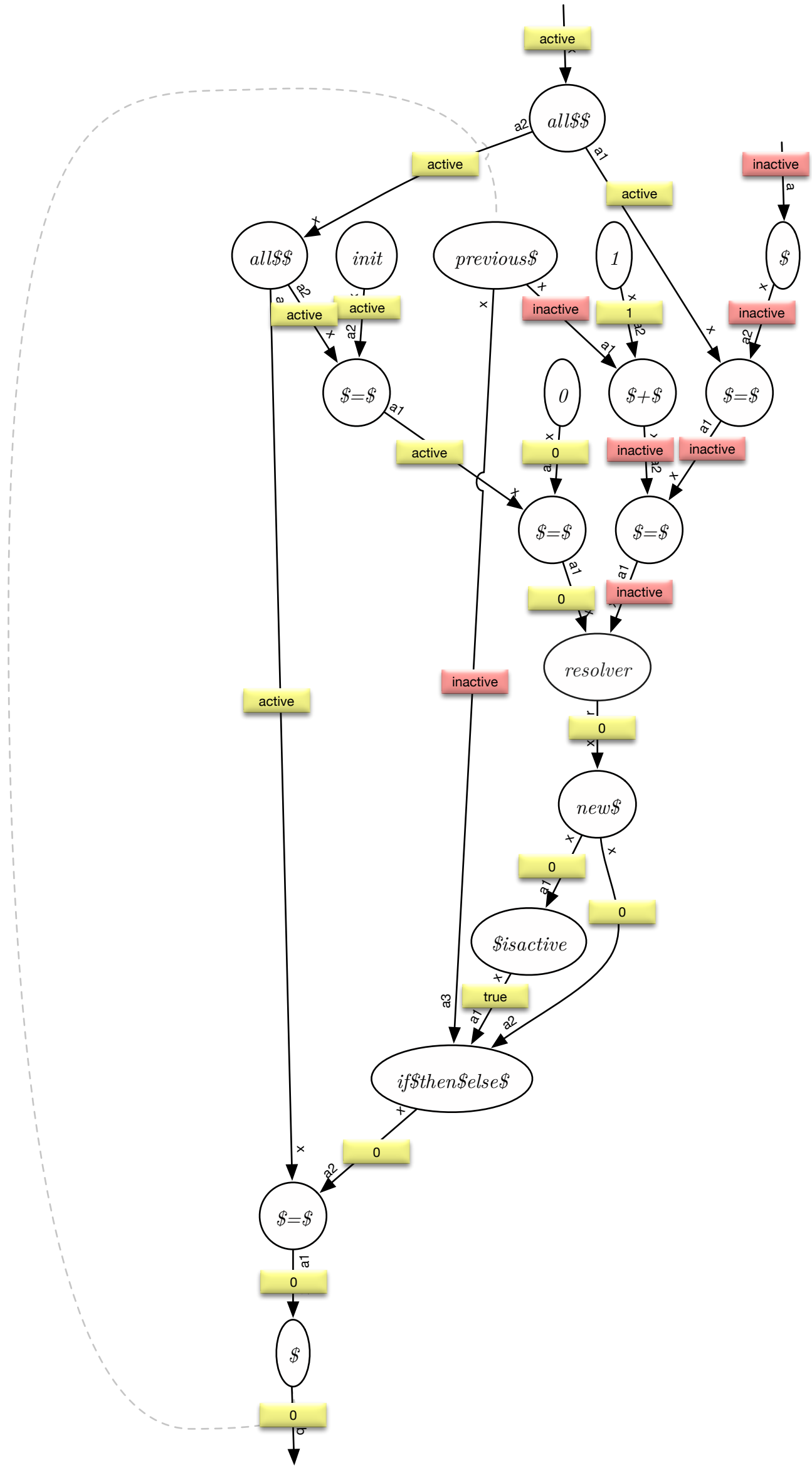
Execution

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.



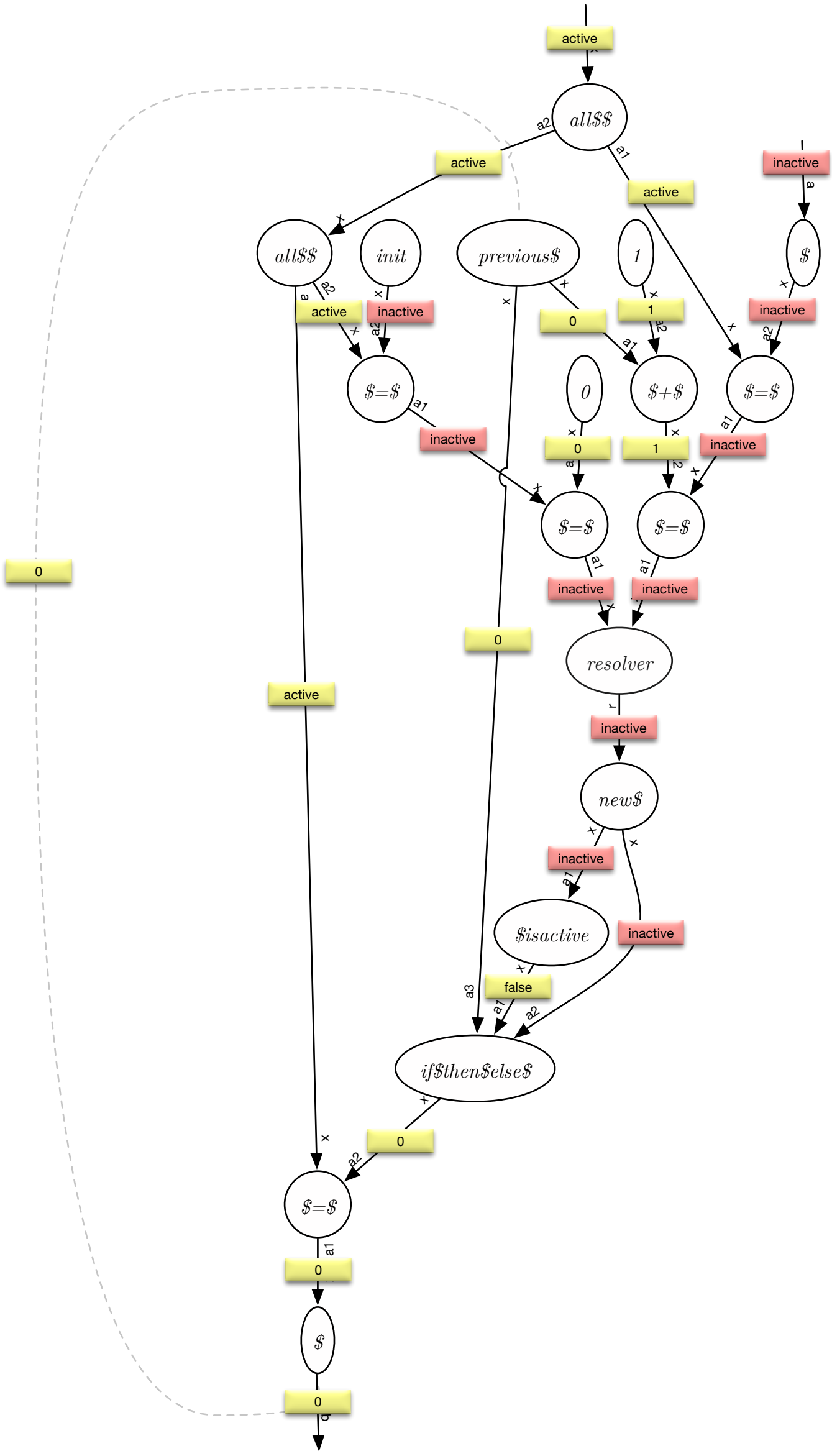
Execute 0

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



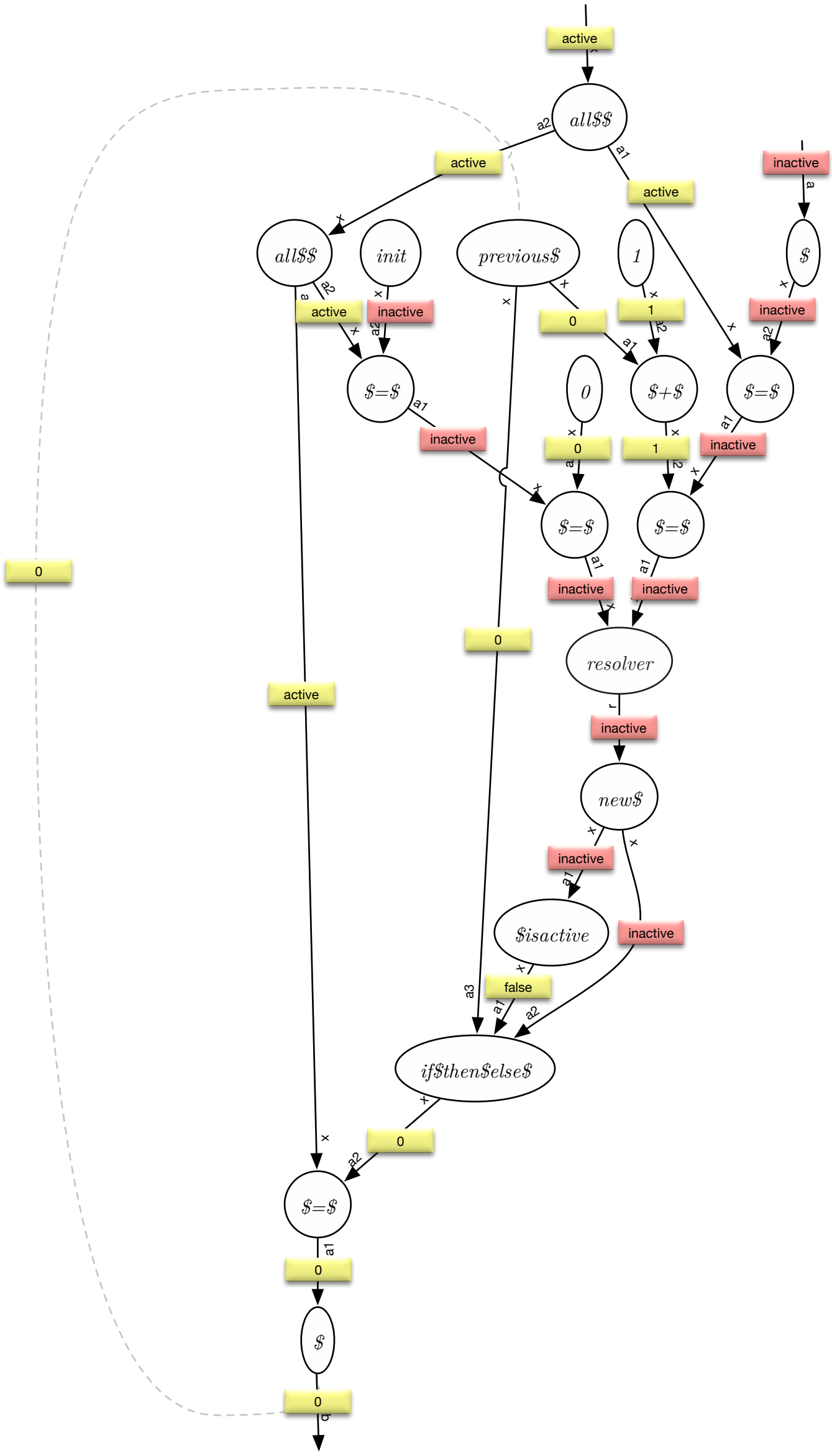
Execute 1

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



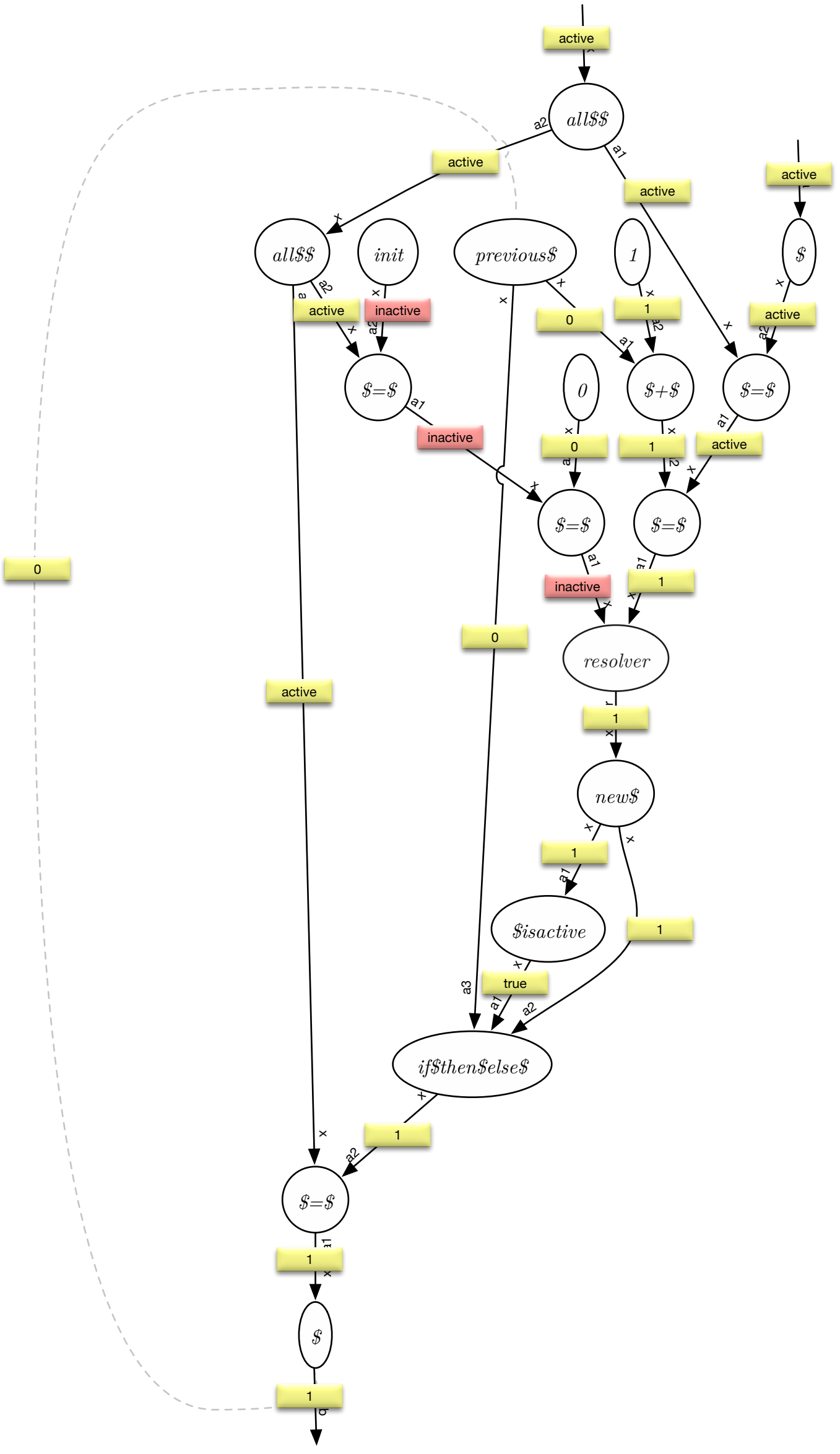
Execute 2

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



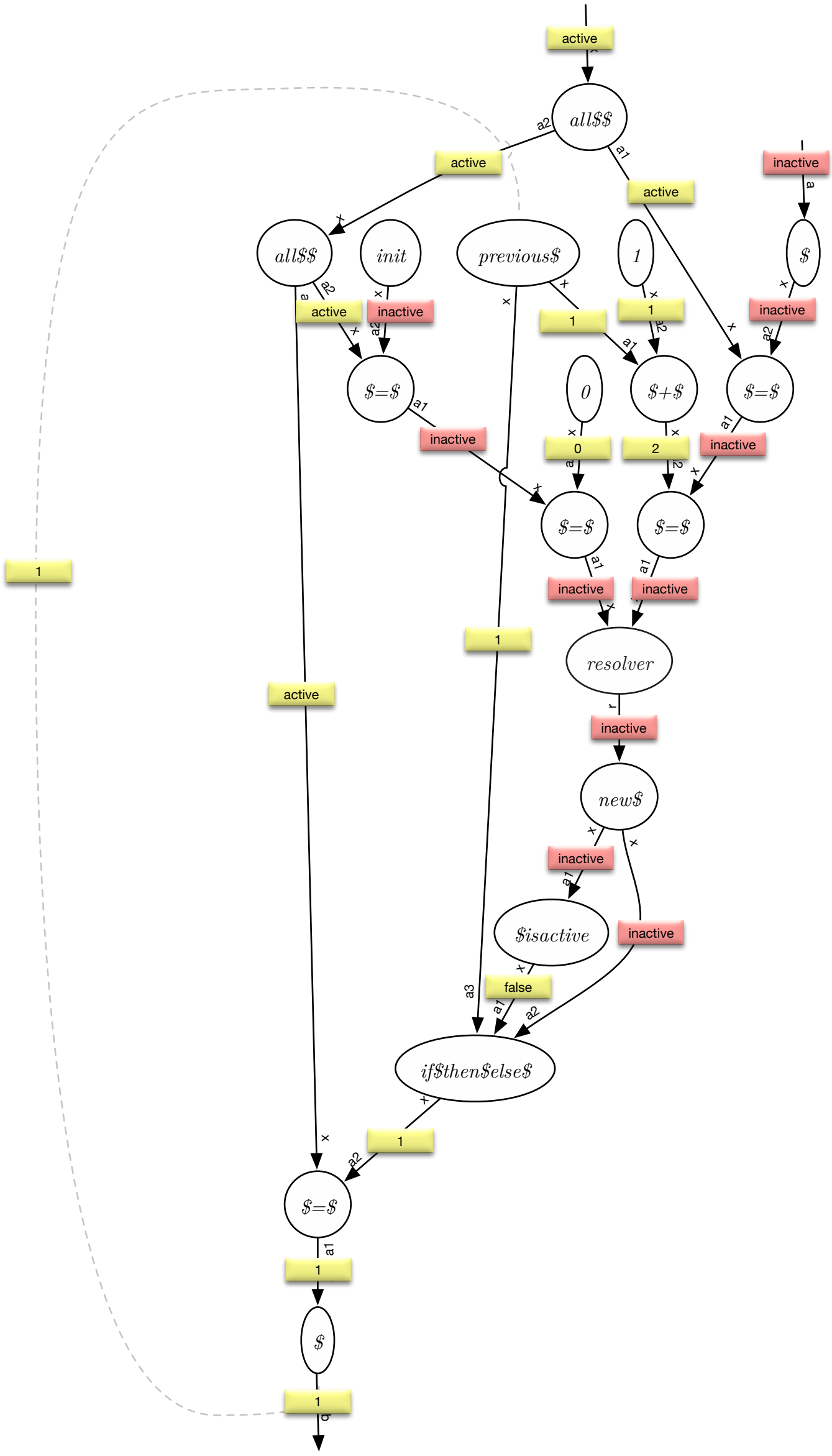
Execute 3

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