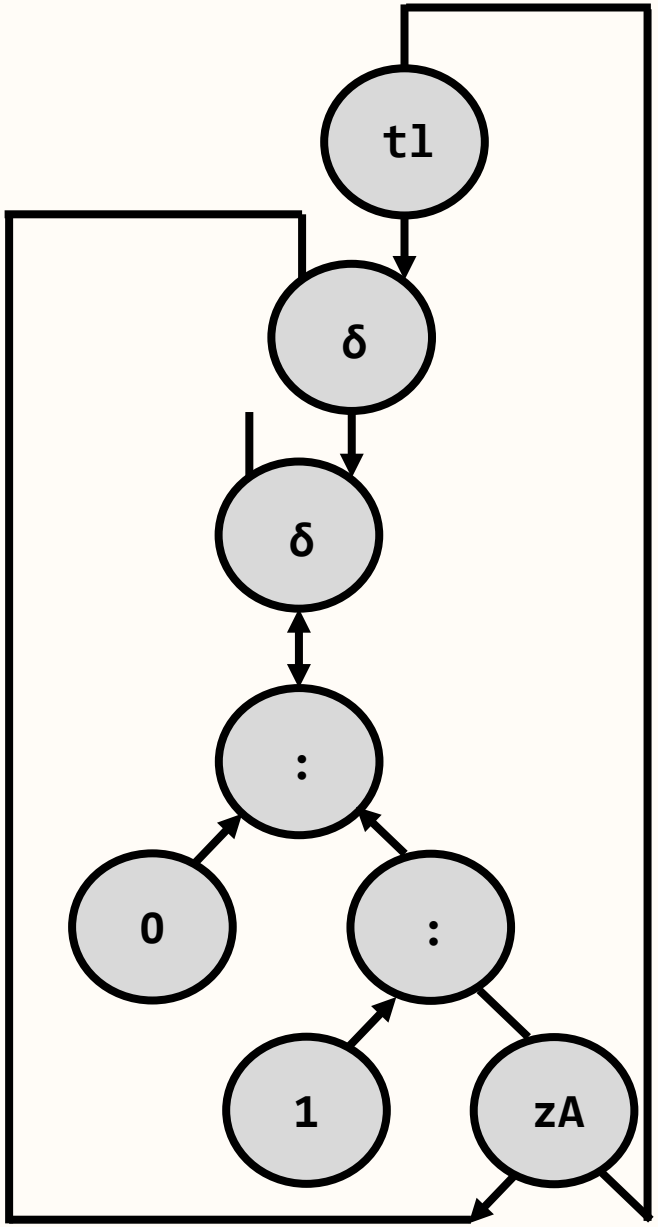
Productivity of Interaction Nets

FOSS Presentation

11th June 2025

UNIVERSITY
OF SUSSEX

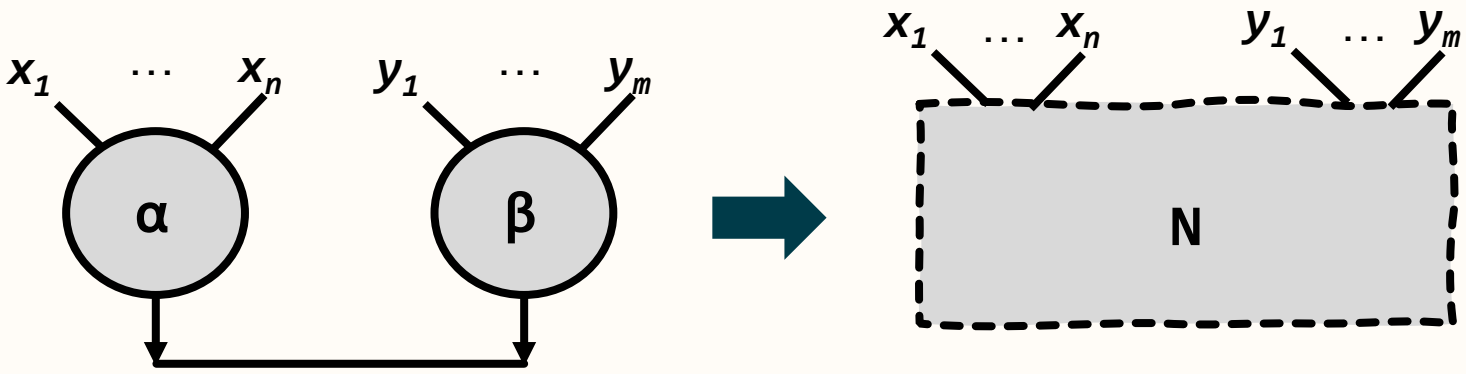
Interaction nets (Lafont)



Interaction rules

At most one rule per agent pair.

Interface preserved.



Cf. proof nets; actually proof structures.

Evaluation = repeated reduction to normal form.

Springtime for interaction nets!

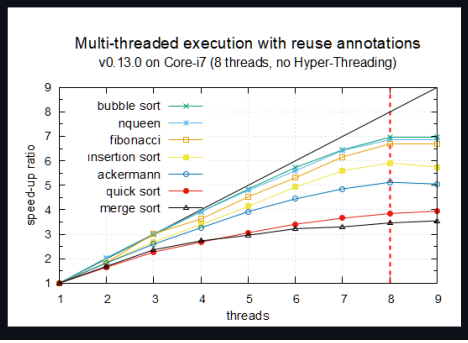
README MIT license

Inpla: Interaction nets as a programming language

What is Inpla

Inpla is a multi-threaded parallel interpreter of interaction nets. Once you write programs for sequential execution, it works also in multi-threaded parallel execution. Each thread is managed on each CPU-core with POSIX-thread library.

- The current version is 0.13.0-2, released on **12 September 2024**. (See Changelog.md for details.)
- The below graph shows speed-up ratio to threads numbers for programs in the following benchmark table.



github.com/ inpla/inpla

Paper GitHub

Higher Order Company

WELCOME TO THE PARALLEL FUTURE OF COMPUTATION

BEND

A PARALLEL LANGUAGE

With **Bend** you can write parallel code for multi-core CPUs/GPUs without being a C/CUDA expert with 10 years of experience. It feels just like Python!

No need to deal with the complexity of concurrent programming: locks, mutexes, atomics... **any** work that can be done in parallel **will** be done in parallel.

```
# Sums all numbers from 0 til 2^depth:
def sum(depth, x):
  switch depth:
    case 0:
      return x
    case _:
      fst = sum(depth-1, x*2+0) # odds the first half
      snd = sum(depth-1, x*2+1) # odds the second half
      return fst + snd

def main:
  return sum(30, 0)
```

Example Program
Simple Parallel Sum

Try it Now →

higherorderco.com

The Vine Programming Language

Vine is an experimental new programming language based on interaction nets.

Vine is a multi-paradigm language, featuring seamless interop between functional and imperative patterns.

See `vine/examples/` for examples of Vine.

```
cargo run -r --bin vine run vine/examples/$NAME.vi
```

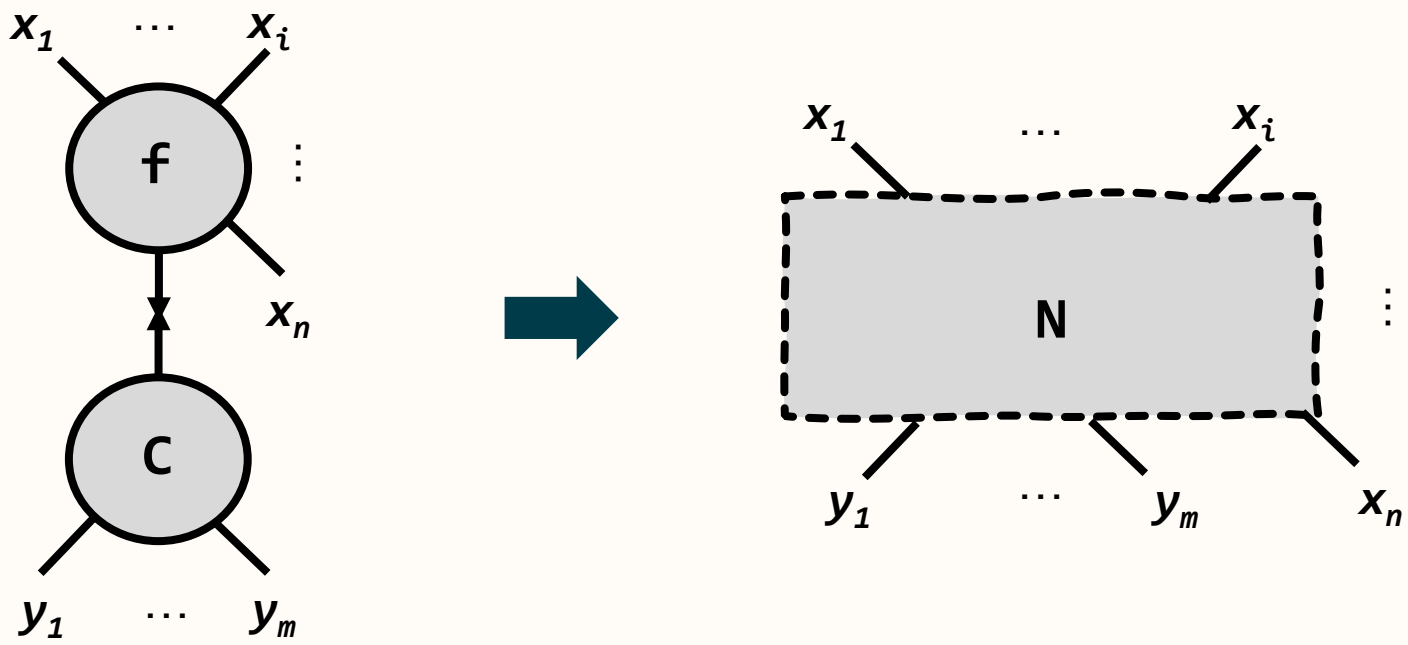
If you're curious to learn more, join the Vine Discord server.

(Vine is still under heavy development; many things will change.)

vine.dev

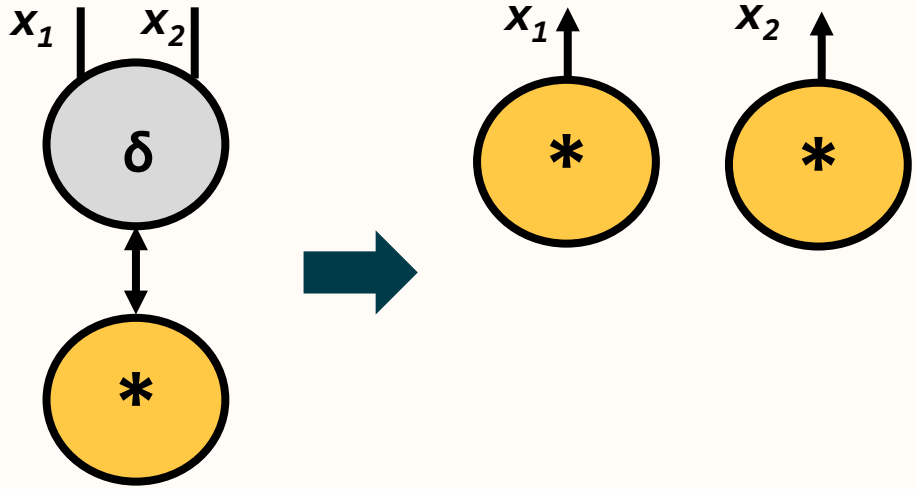
Function-Constructor nets

$f(C(y_1, \ldots y_m), x_{i\leq 1}, \ldots, x_n) = N(y_1, \ldots, y_m, x_{i\leq 1}, \ldots, x_n)$

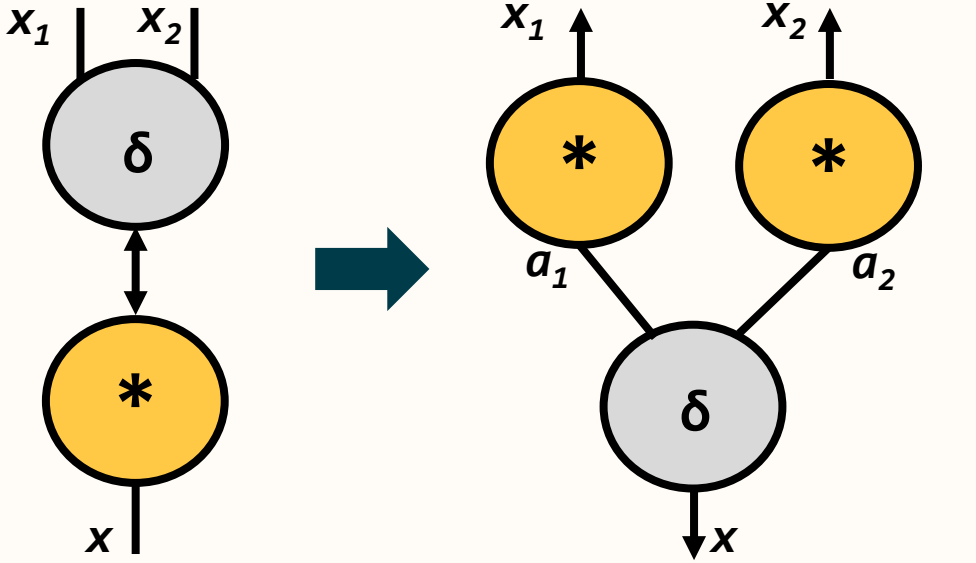


Example rule - Duplication

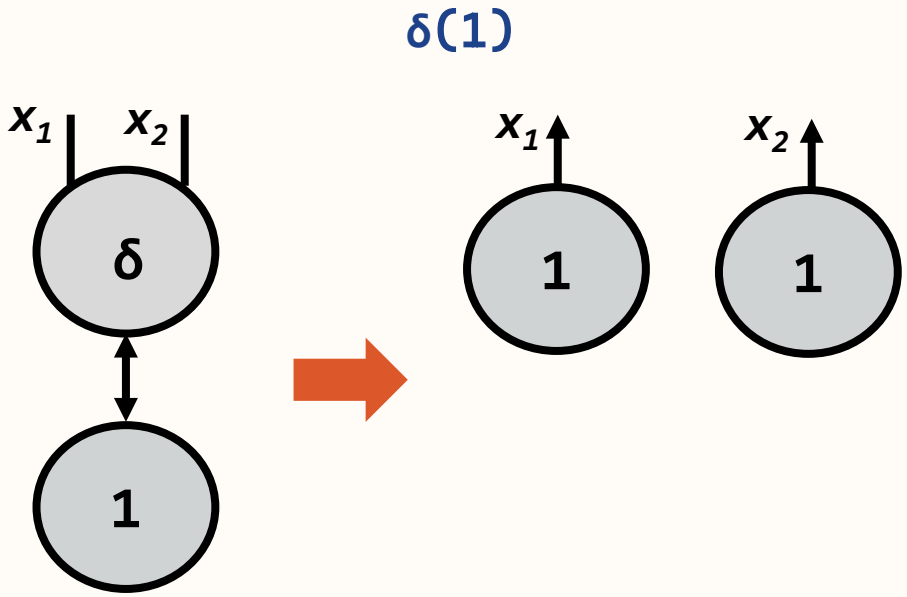
$\delta(*) = * \mid *$



$\delta(*(x)) = [a_1, a_2]\sim\delta(x)\ \texttt{in}\ *(a_1)\mid *(a_2)$

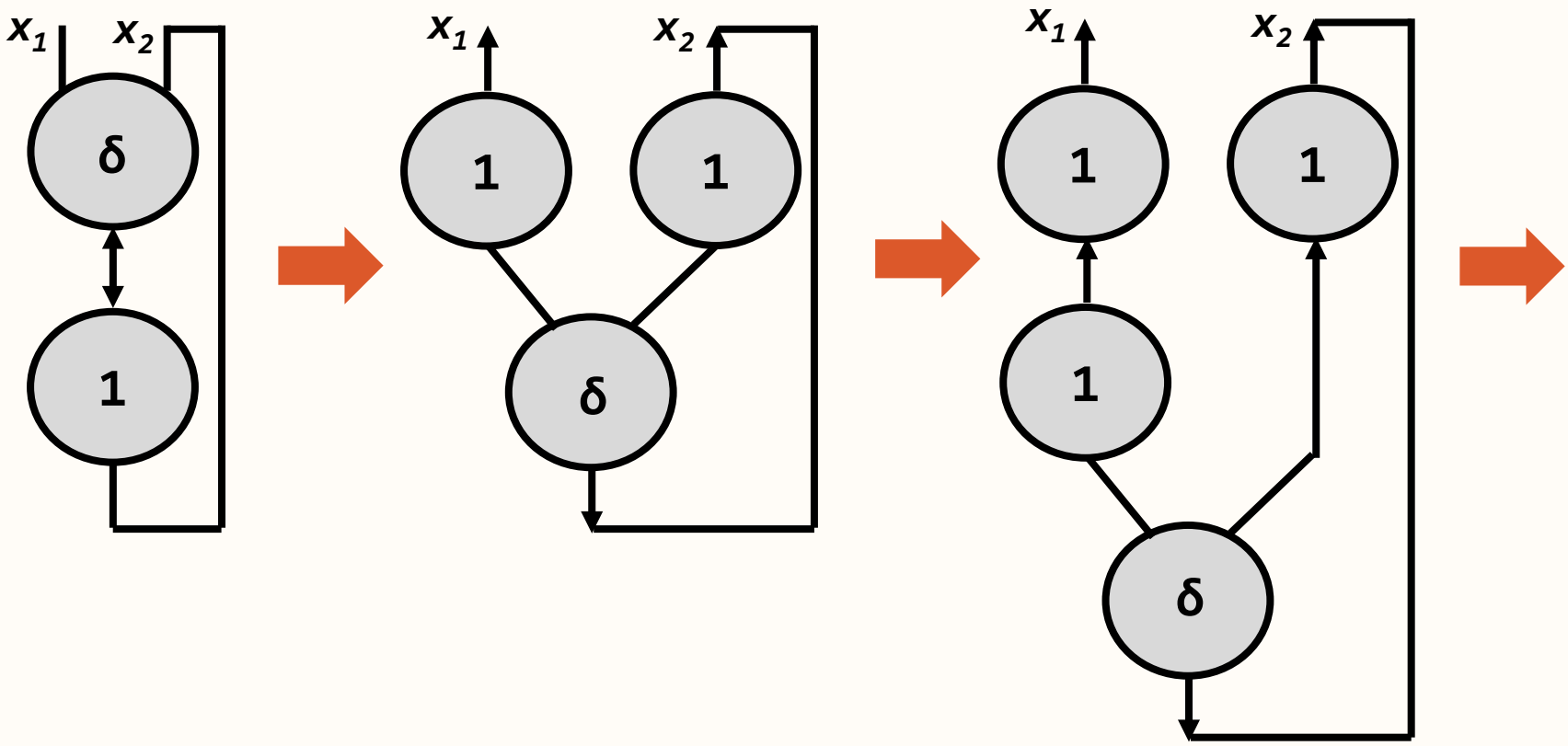


Productive normal form



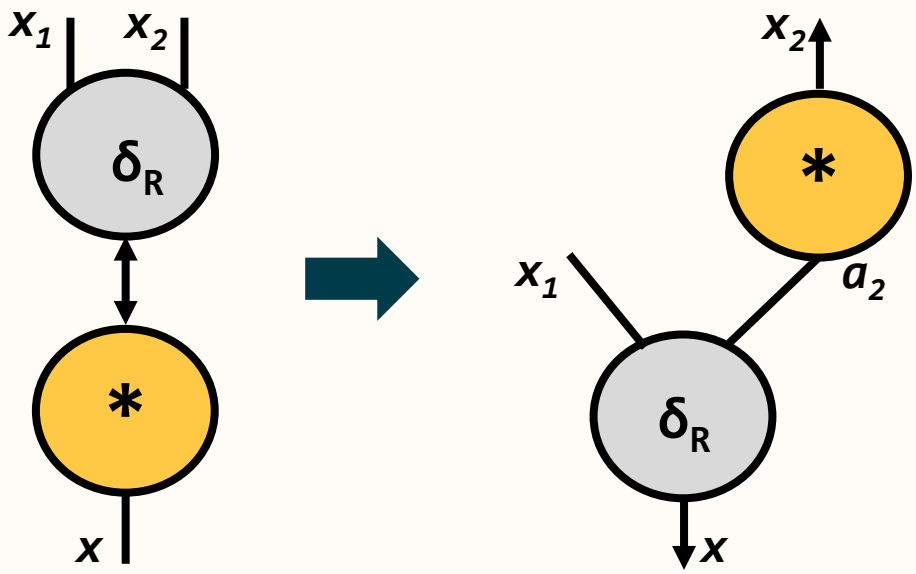
Productive stream

$[x_t|x_t]\sim\delta(1(x_t))$ in x_1



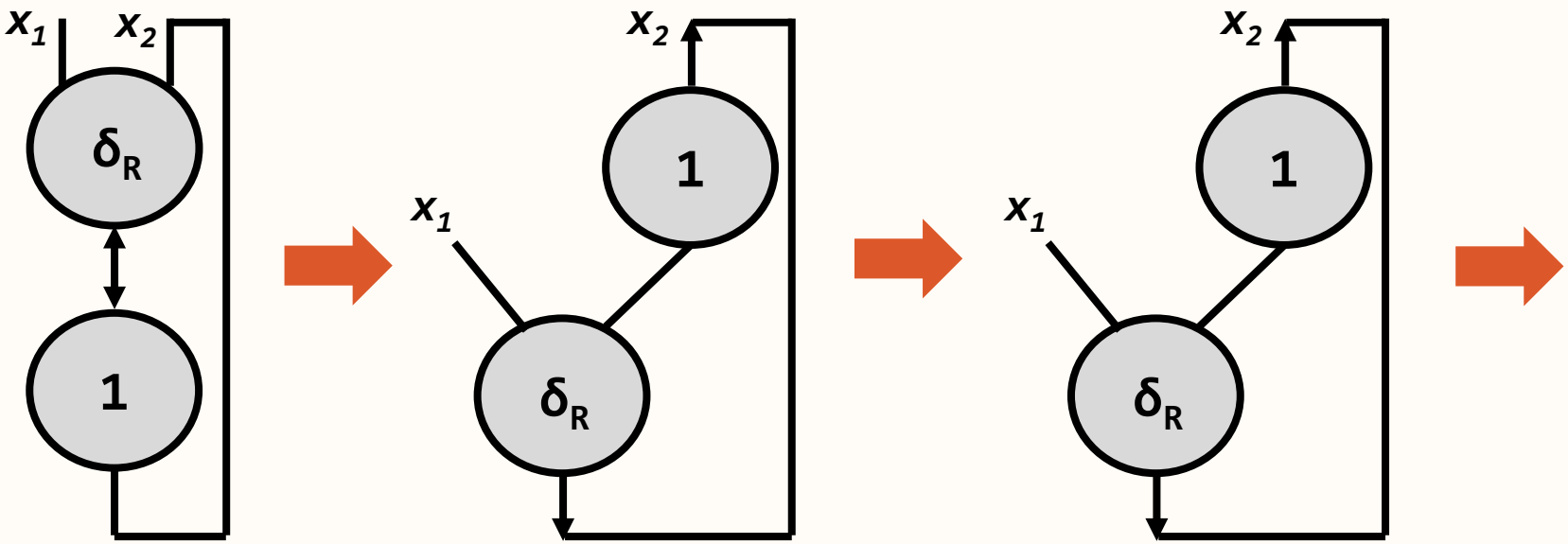
Non-productive non-terminating

$\delta_R(*(x)) = [x_L, x_l] \sim \delta_R(x)$ in ~~$*(a_1)$~~ $*(a_2)$



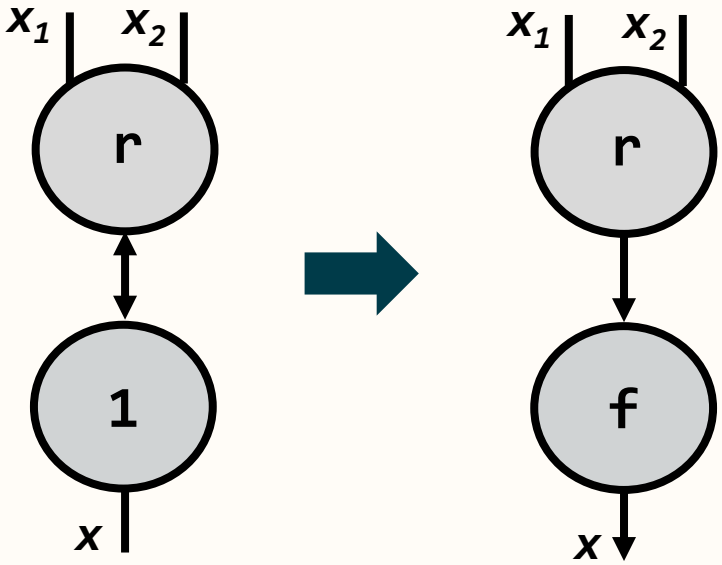
Non-productive non-terminating

$[x_L, x_l] \sim \delta_R(1(x_l))$ in x_1

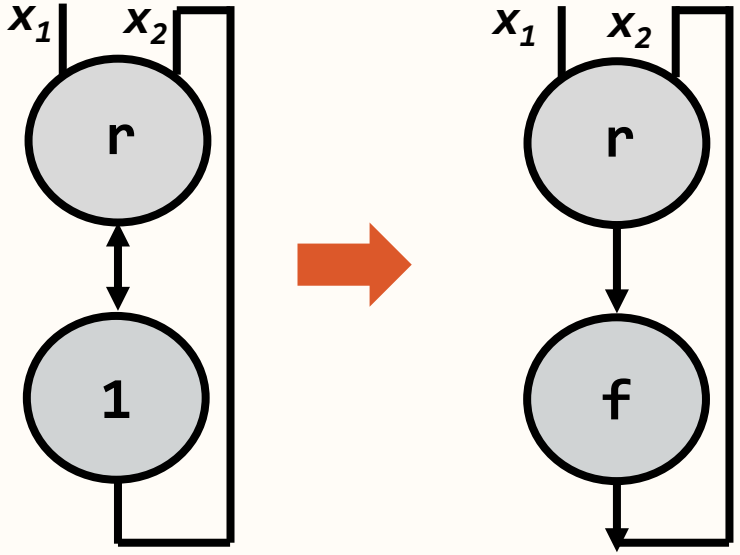


Deadlock (non-productive nf)

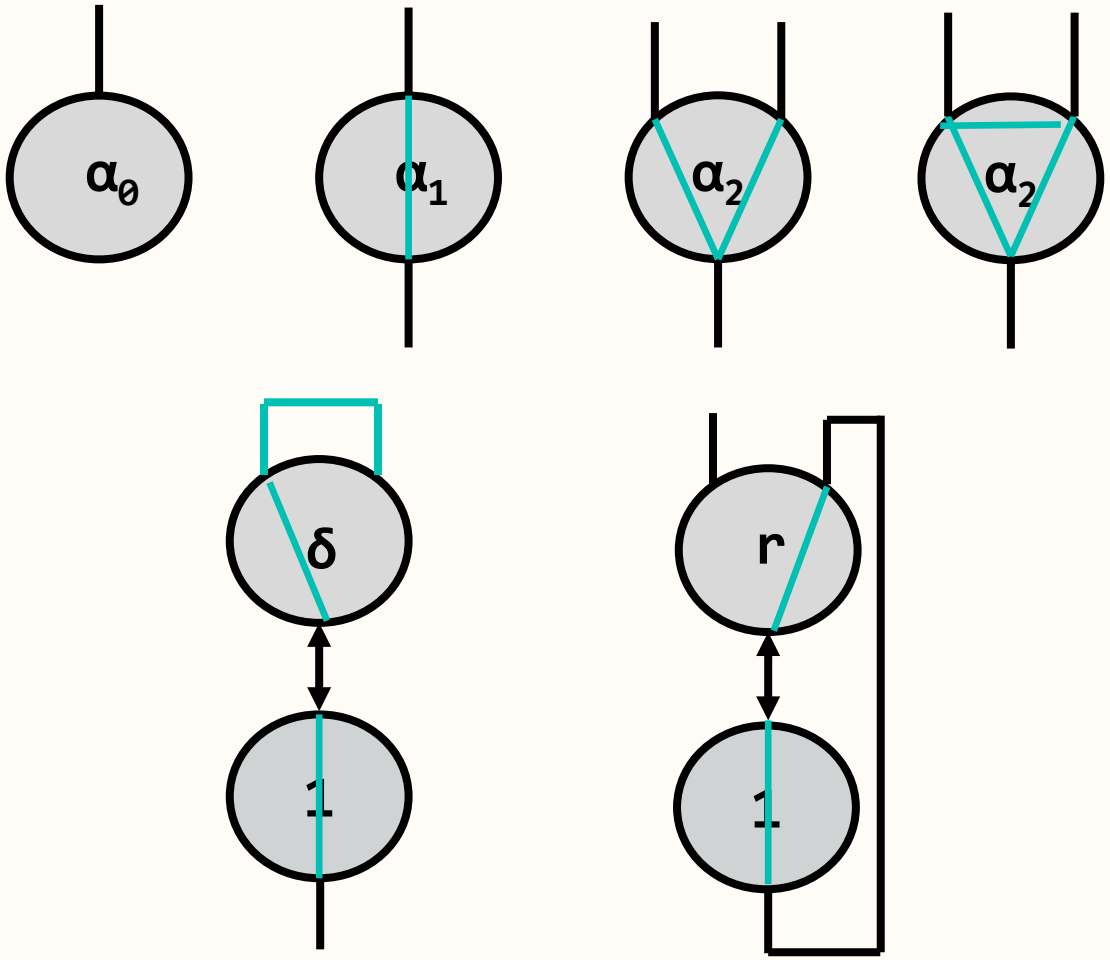
r(1(x)) = r(f(x))



[x_1,x_1]~r(1(x_1)) in x_1

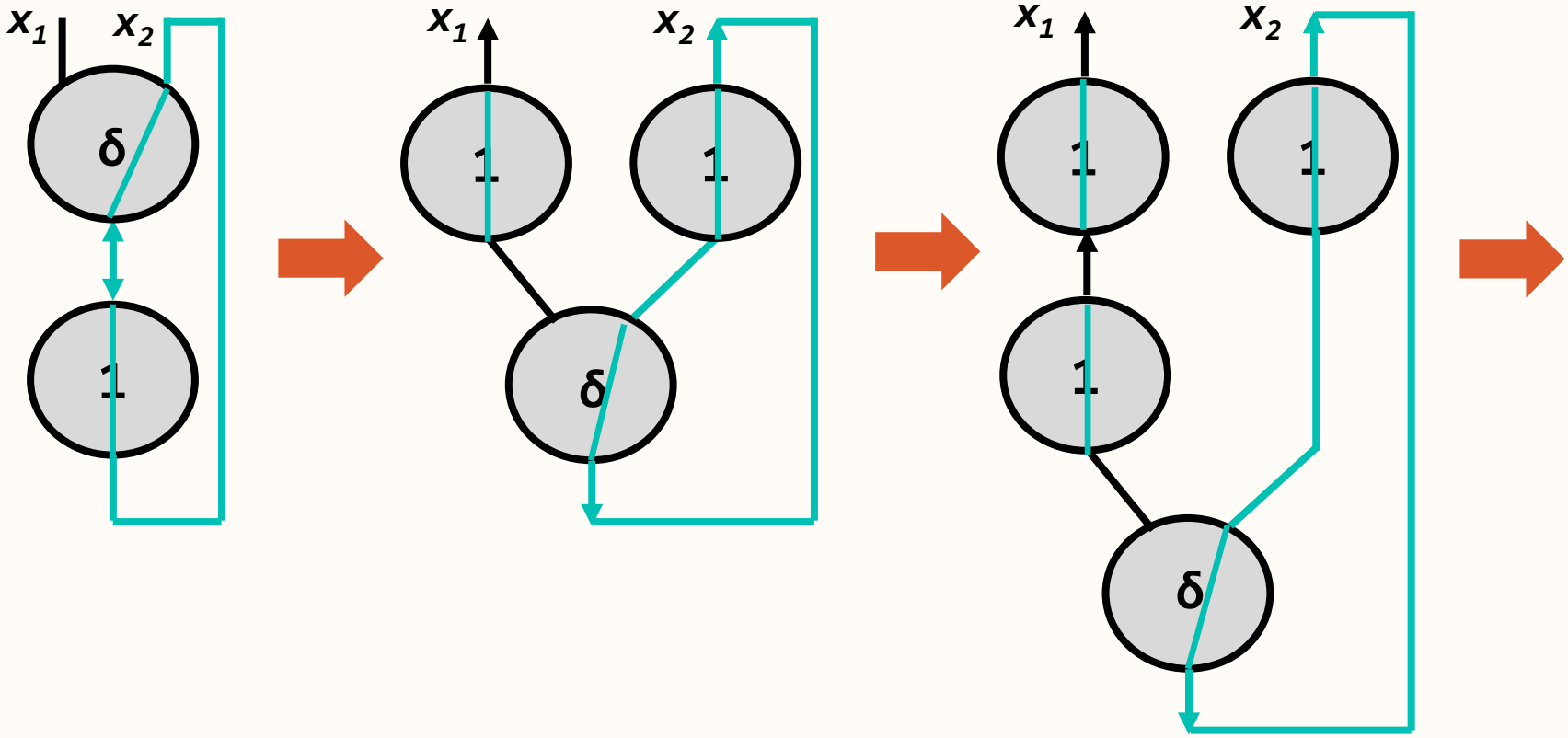


How to avoid deadlock



“Switching” – The Paradigm of Interaction, Lafont, 1991

But…



Defn: Productivity

Net:

	Terminating	Non-terminating
Output	Normal form	Stream
No output	Deadlock	Infinite loop

Productivity:

	Terminating	Non-terminating
Output	*n*-productive	Stream-productive
No output	Unproductive	Unproductive

Hybrids exist – e.g. produces *n* outputs then loops.

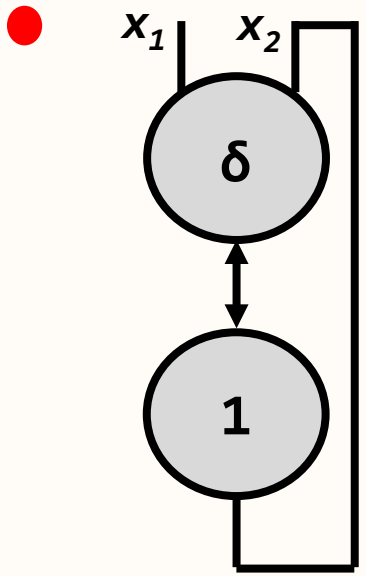
Classify as *n*-productive.

How quantify? *n* = number interactions with erase agent.

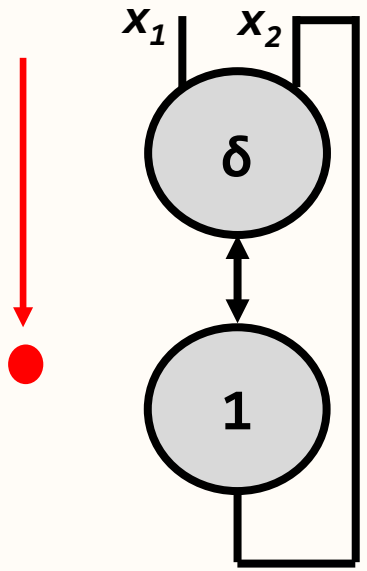
Question

Can we statically determine which nets are productive and which are not?

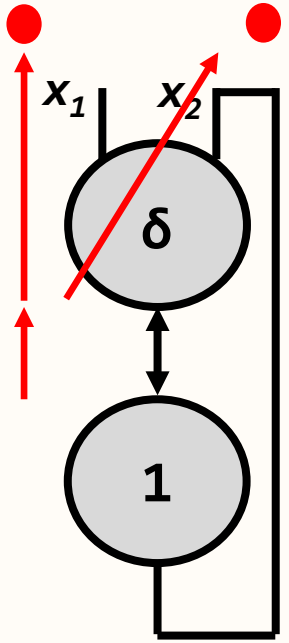
Enter the pebble!



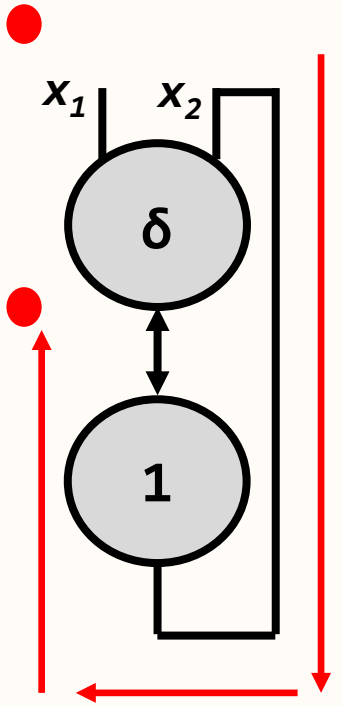
Enter the pebble!



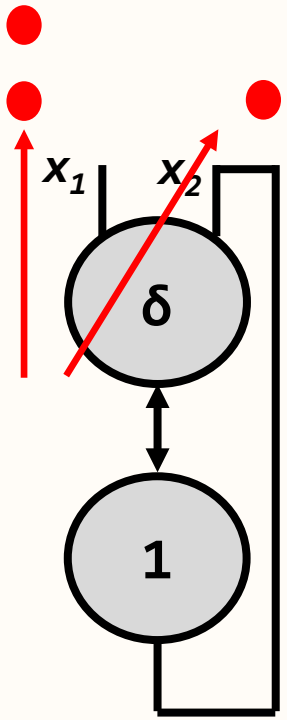
Enter the pebble!



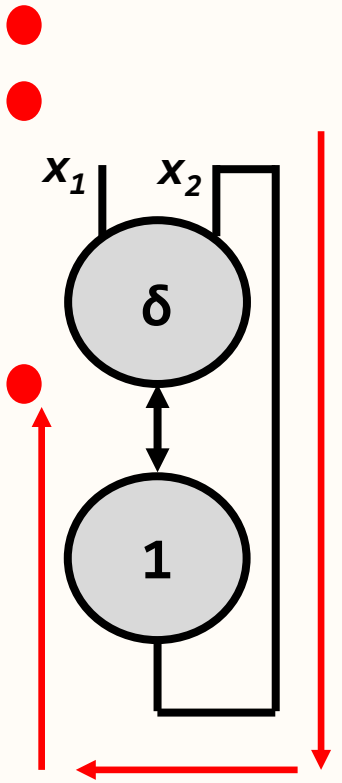
Enter the pebble!



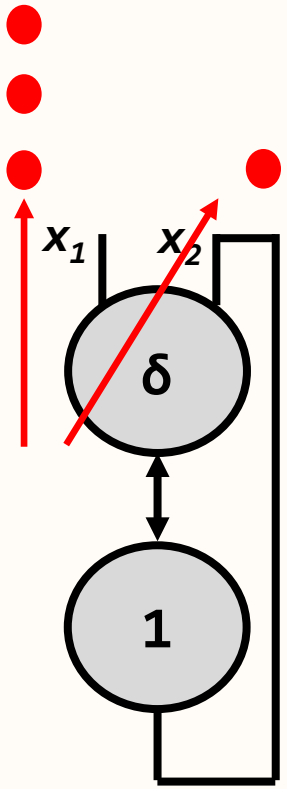
Enter the pebble!



Enter the pebble!

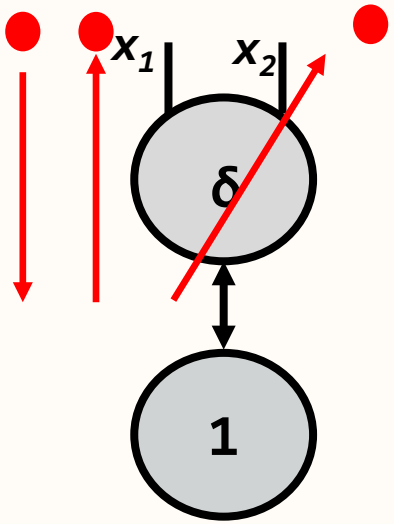


Enter the pebble!

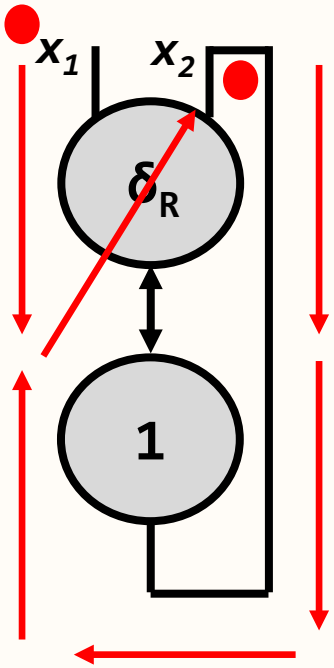


Other pebble analyses

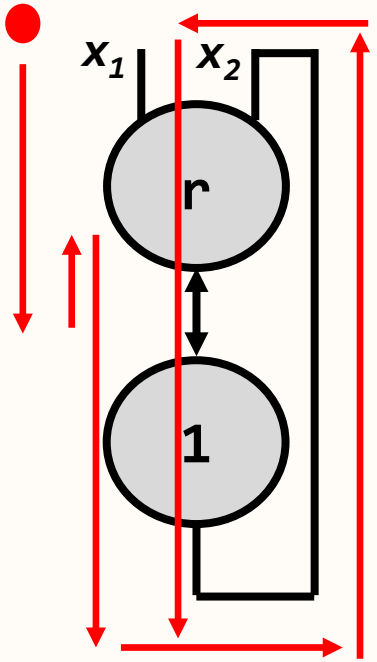
Productive normal form



Infinite loop



Deadlock



Pebble analysis

Pebble value at port *x* = limit of number pebbles at port *x*

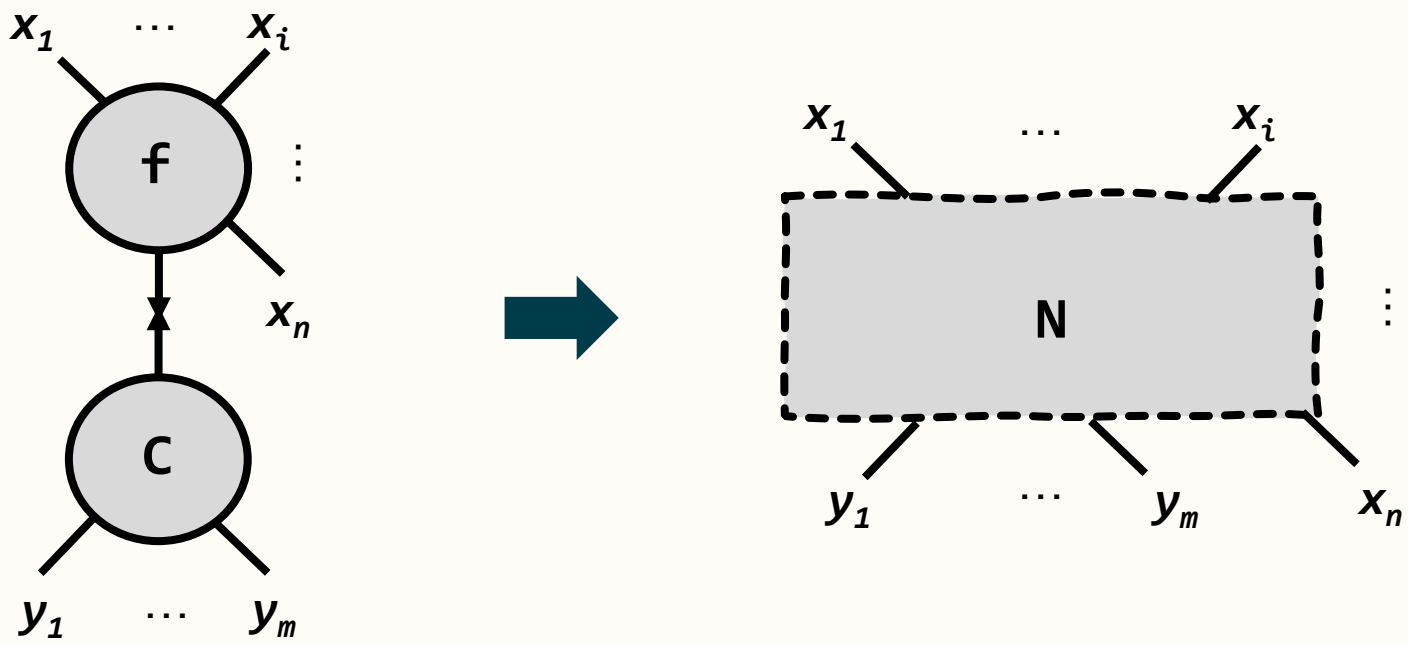
Q1. How define pebble rule?

Q2. Does value always exist?

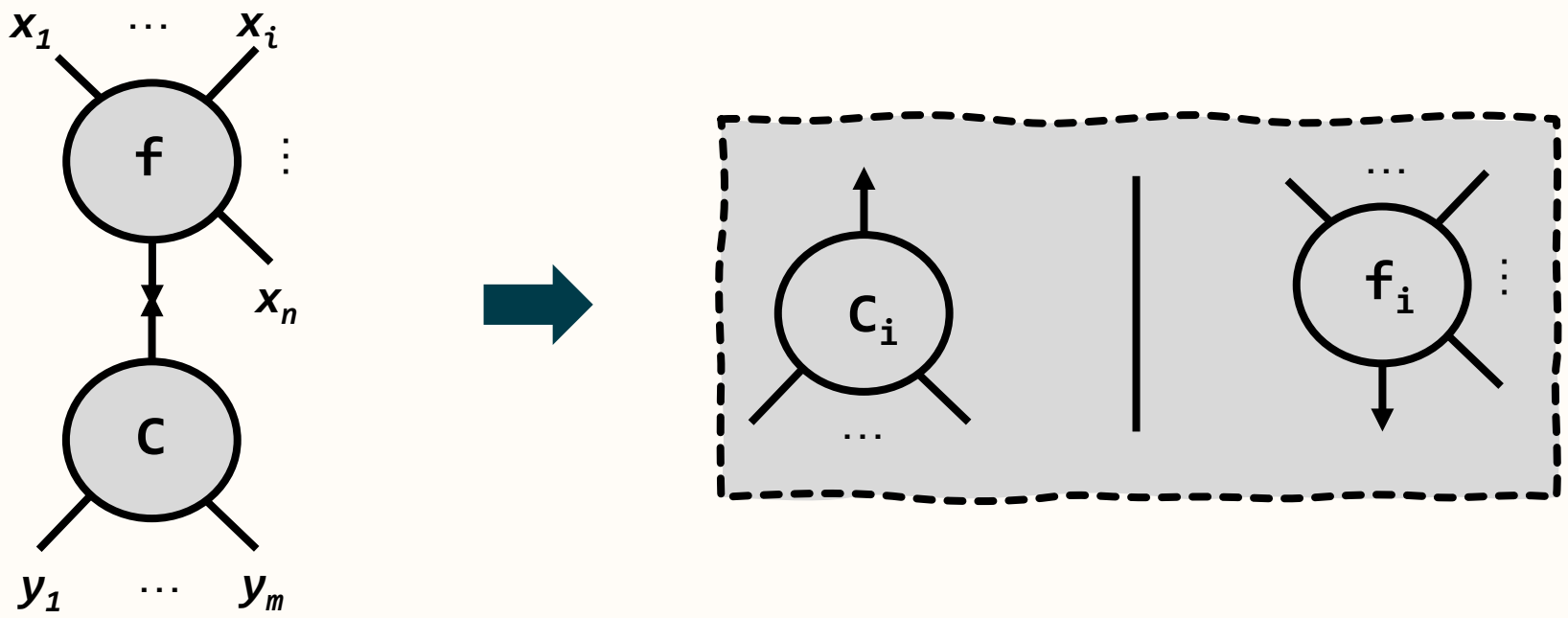
Q3. How interpret it?

Q4. Can we compute it?

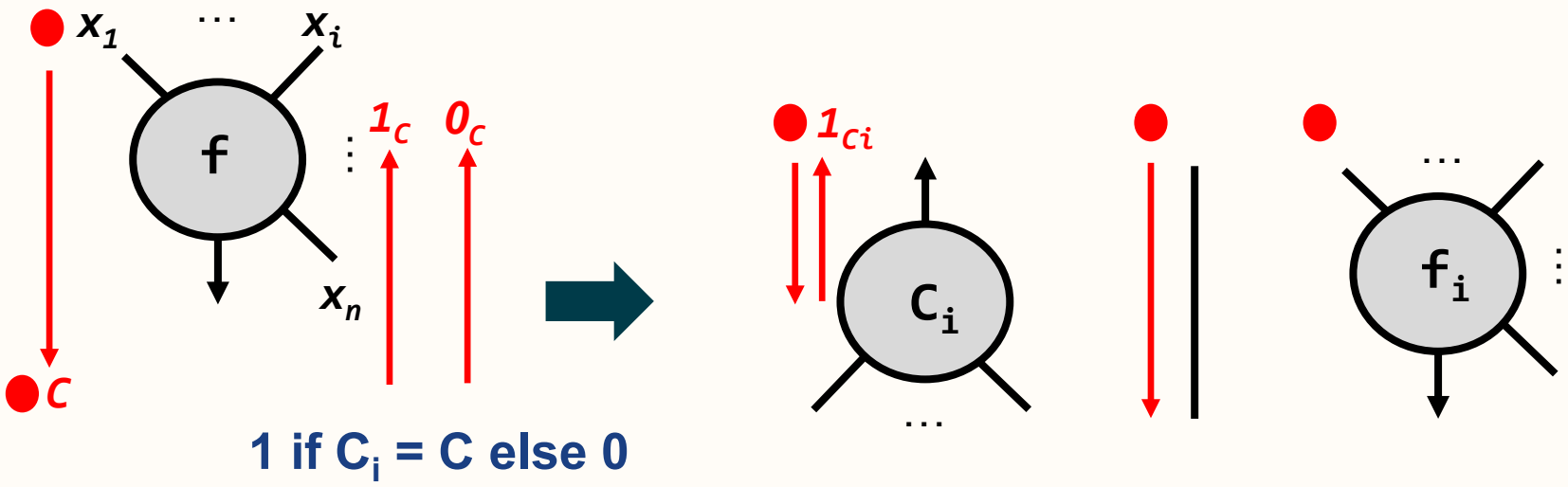
Interaction rules → pebble rules



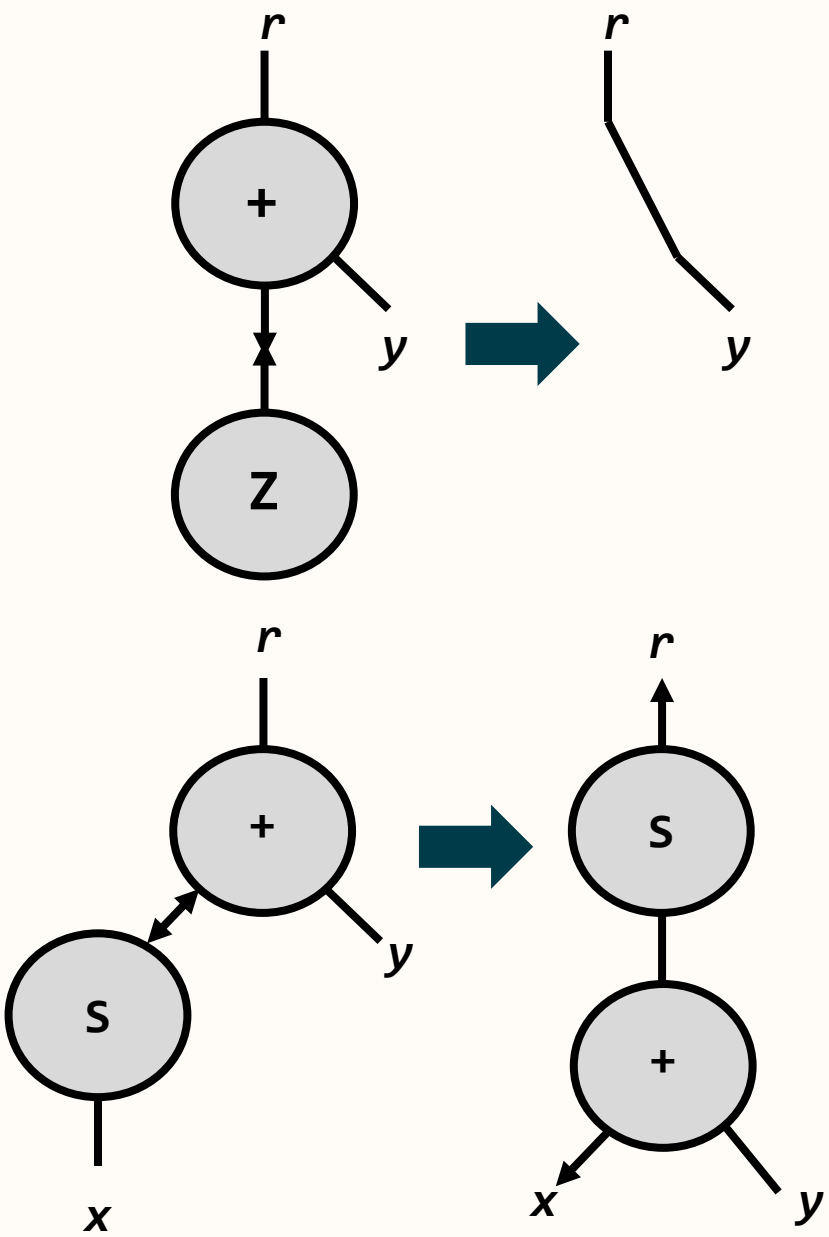
Interaction rules → pebble rules

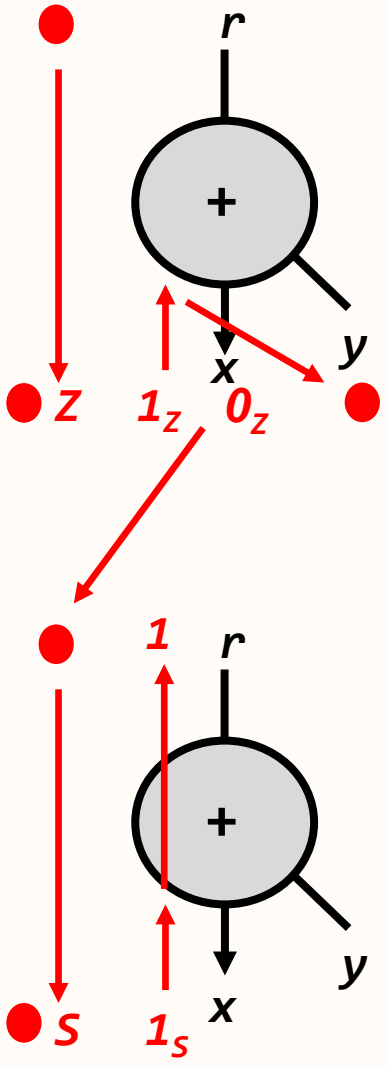


Interaction rules → pebble rules



Example: unary addition





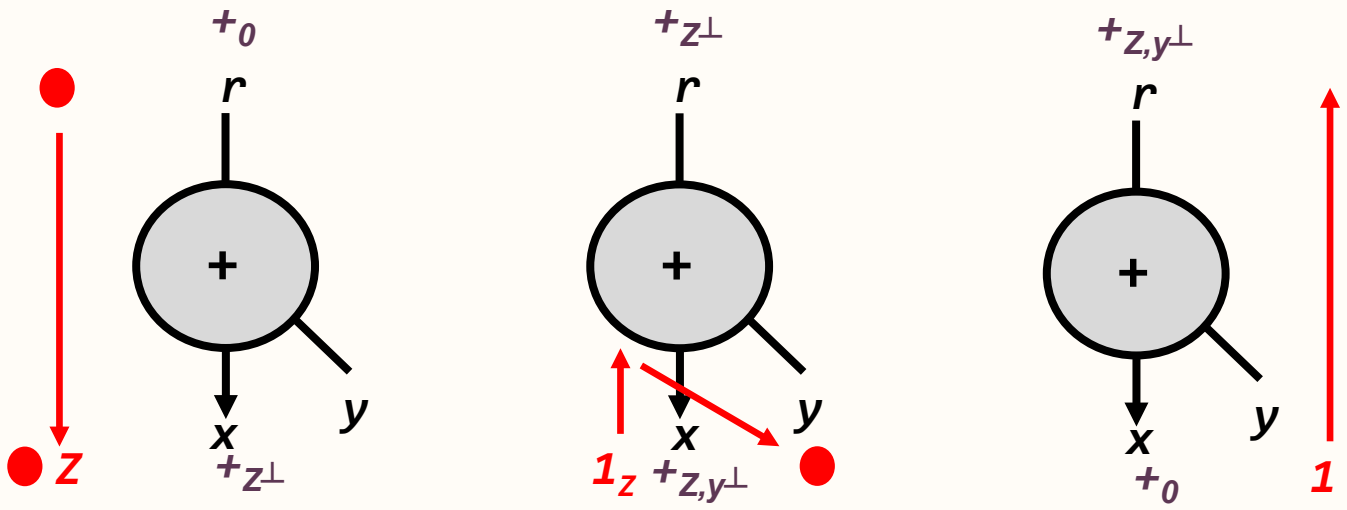
Need to consider cyclicality

Note: functions don't respond to questions.

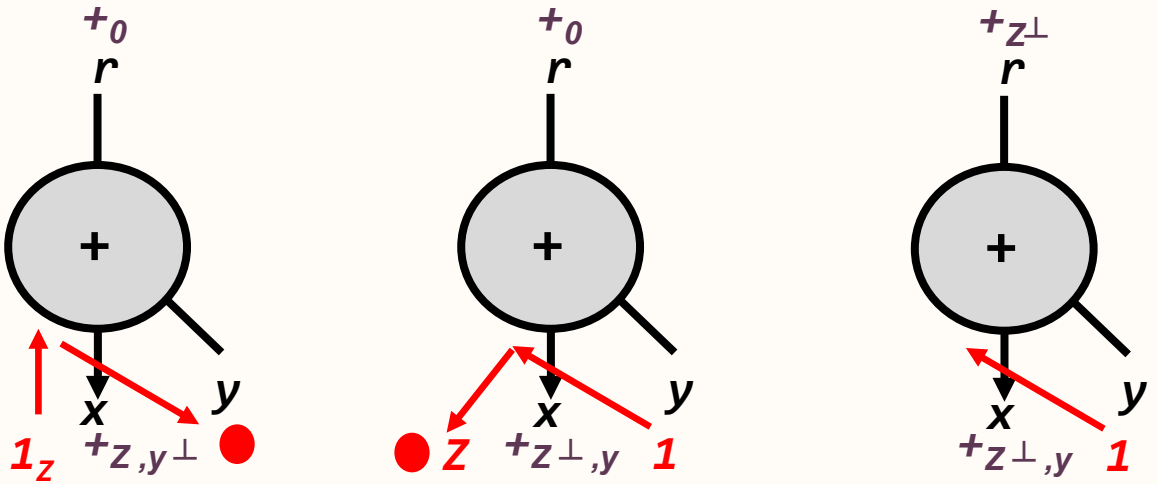
(Constructors don't respond to answers).

From primary to derived rules

Primary

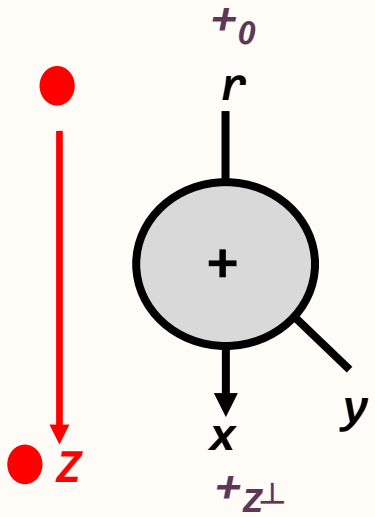


Derived

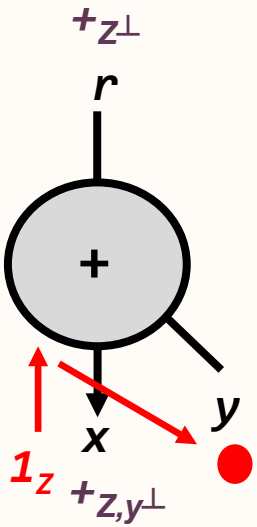


State transition system

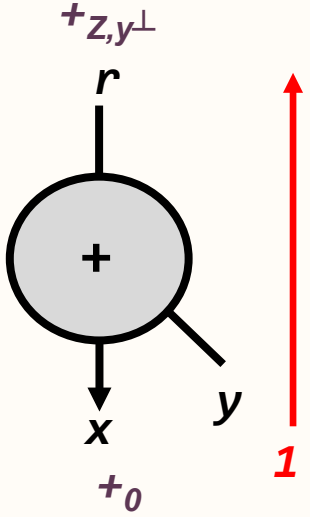
Primary



$(+_0, r, \cdot) \rightarrow (+_{Z'}, x, \cdot\, Z)$

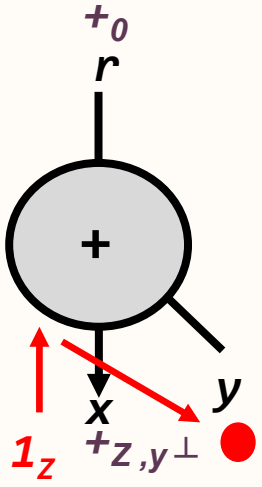


$(+_{Z'}, x, 1) \rightarrow (+_{Z,\,y'}, y, \cdot)$

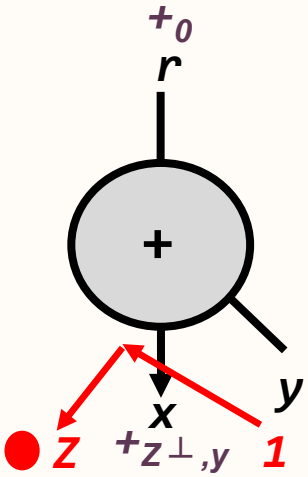


$(+_{Z,\,y'}, y, 1) \rightarrow (+_0, r, 1)$

Derived

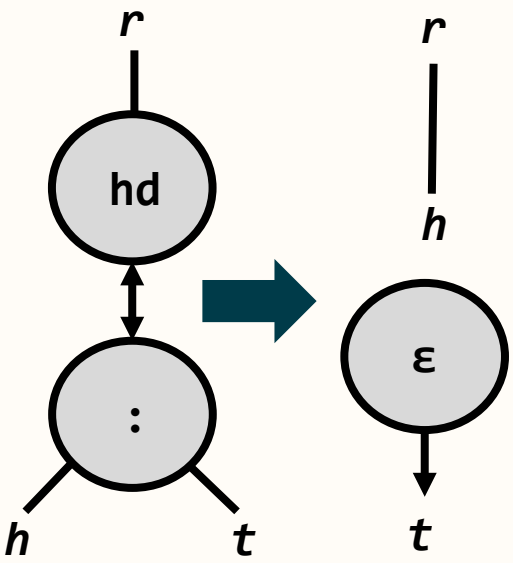


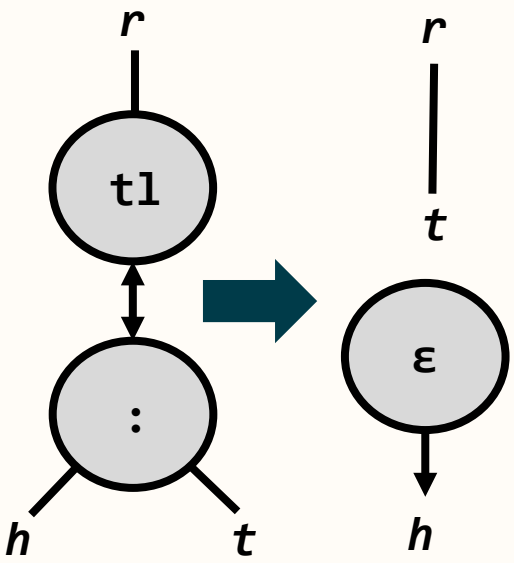
$(+_0, x, 1) \rightarrow (+_{Z,\,y'}, y, \cdot)$

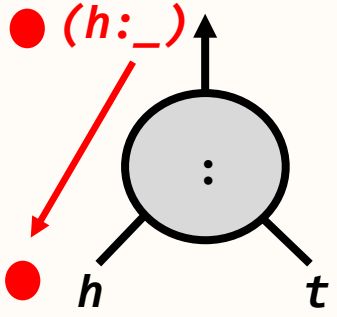


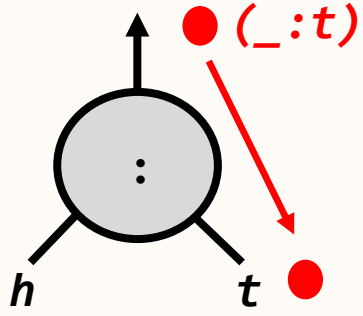
$(+_0, y, 1) \rightarrow (+_{Z',\,y}, x, \cdot\, Z)$

Constructor rules

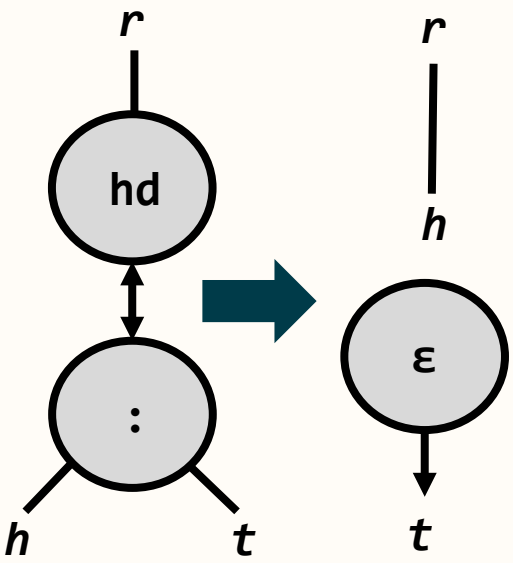


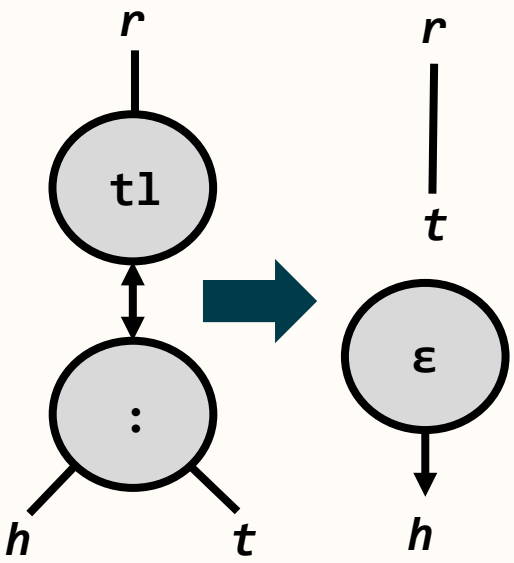


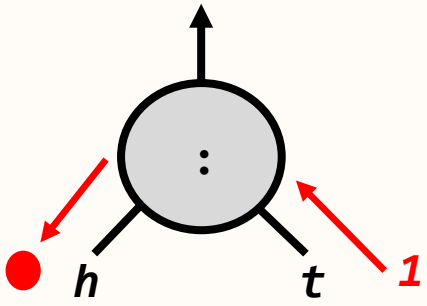


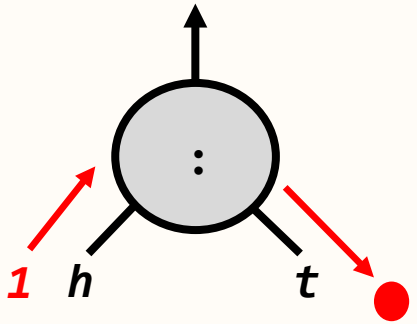


Constructor rules

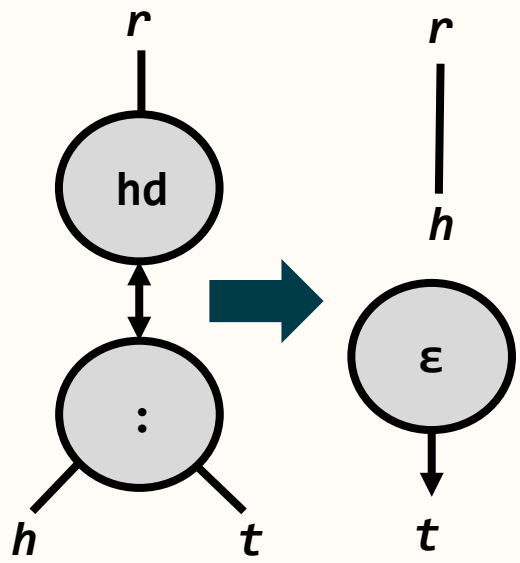


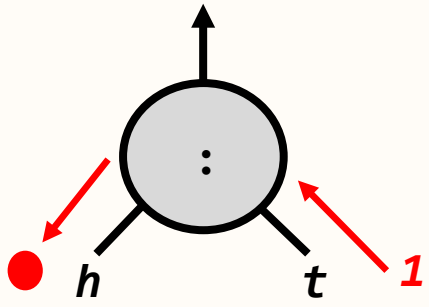


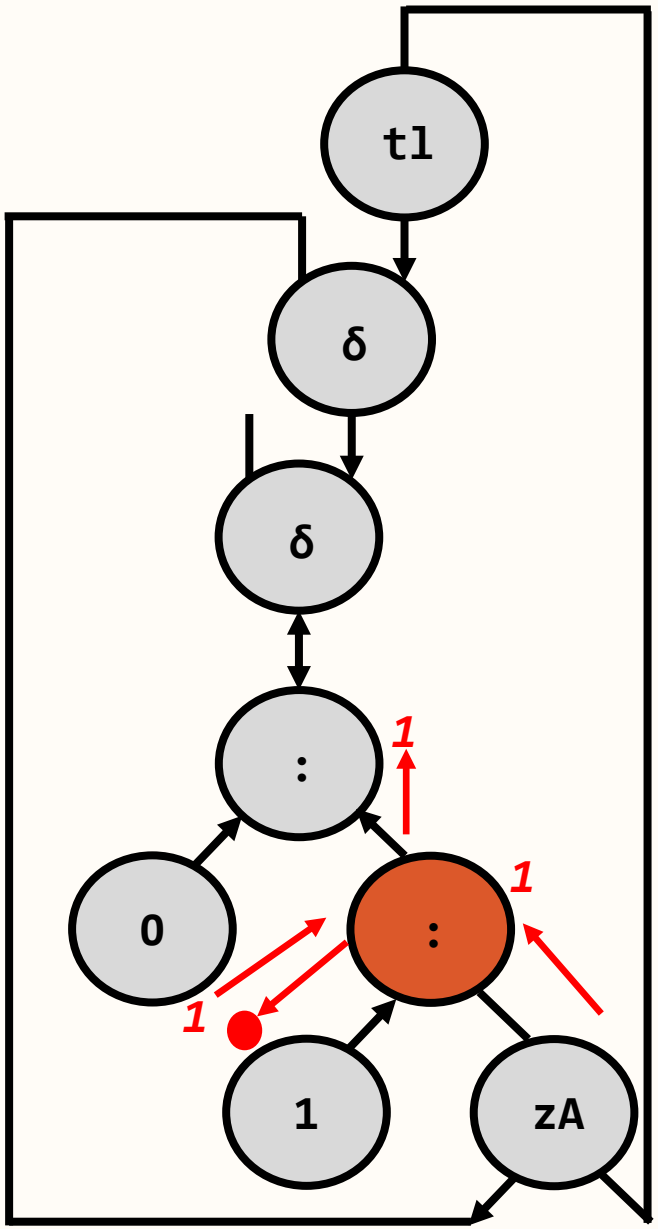


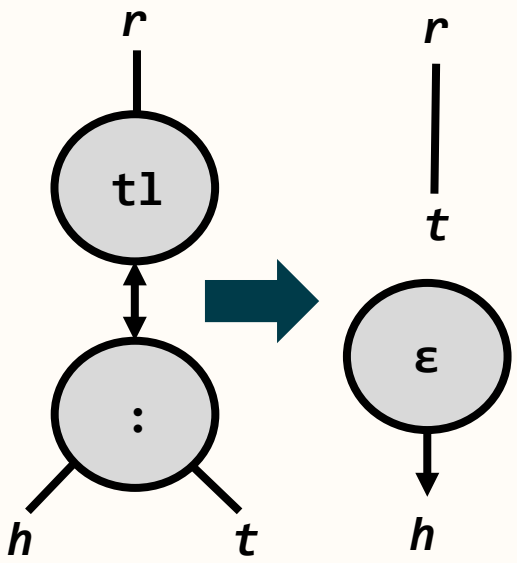


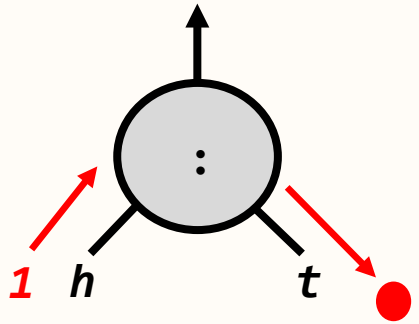
Constructor rules



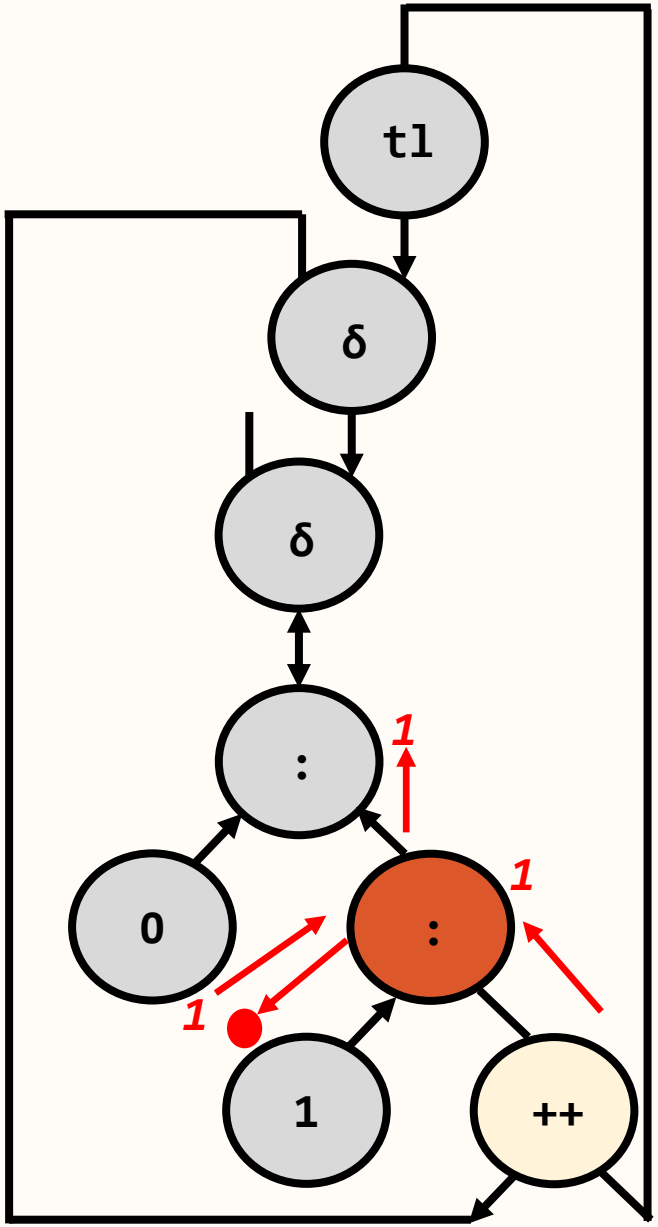




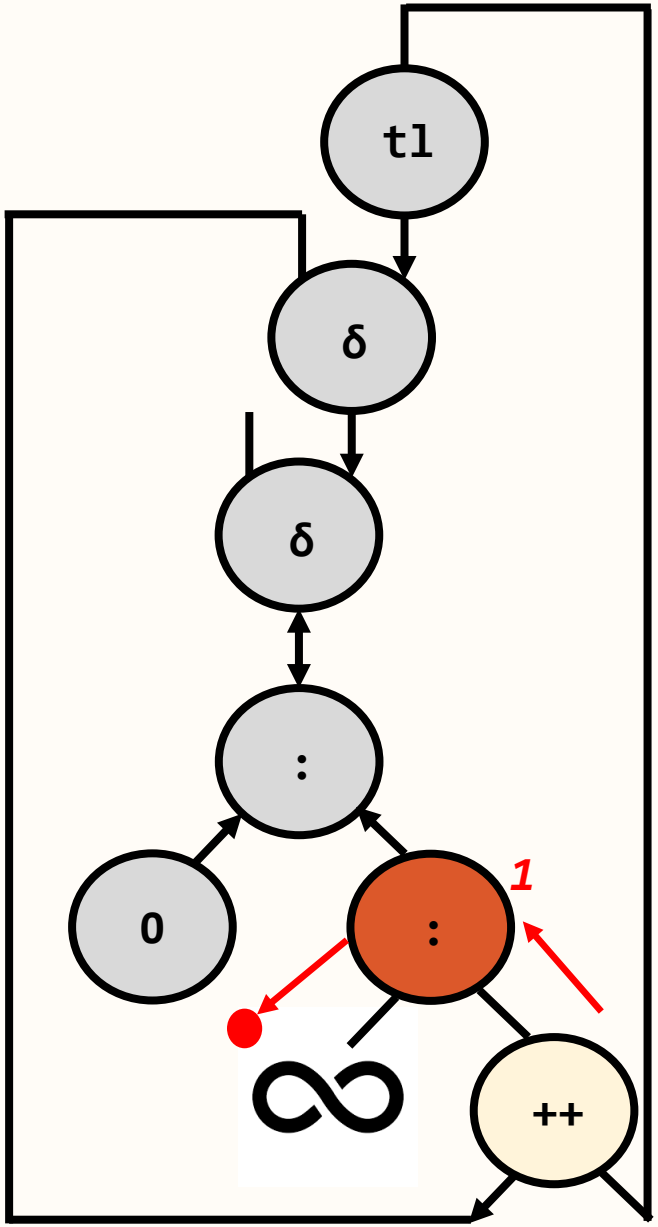




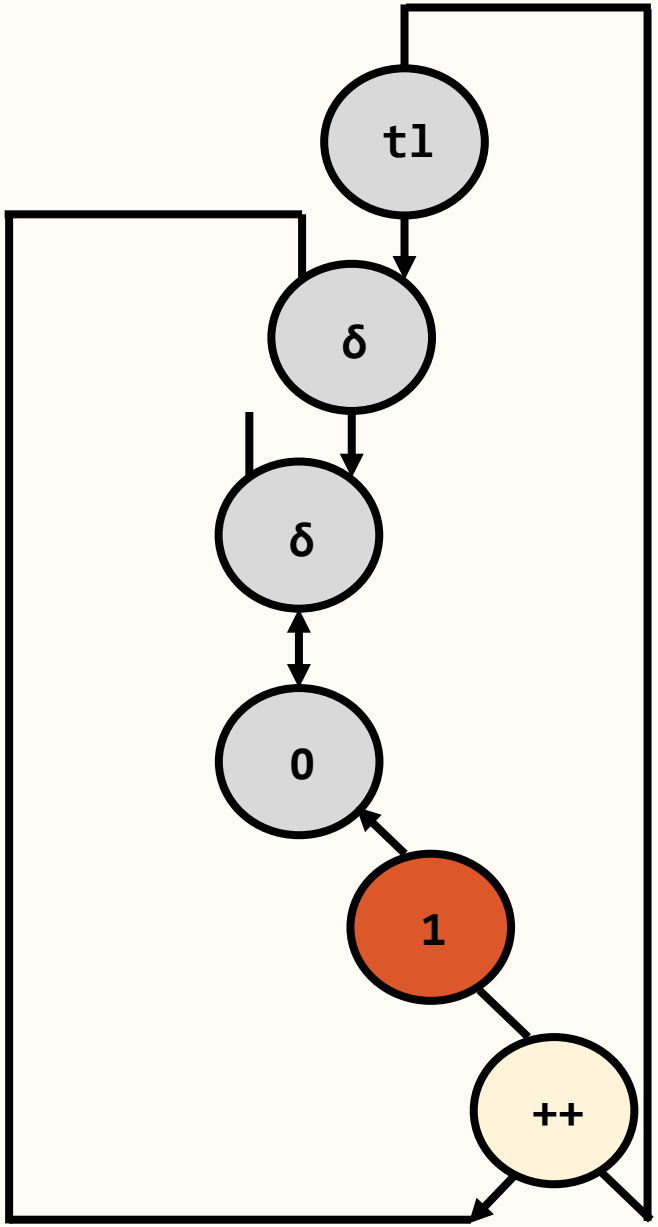
The “tail trap”



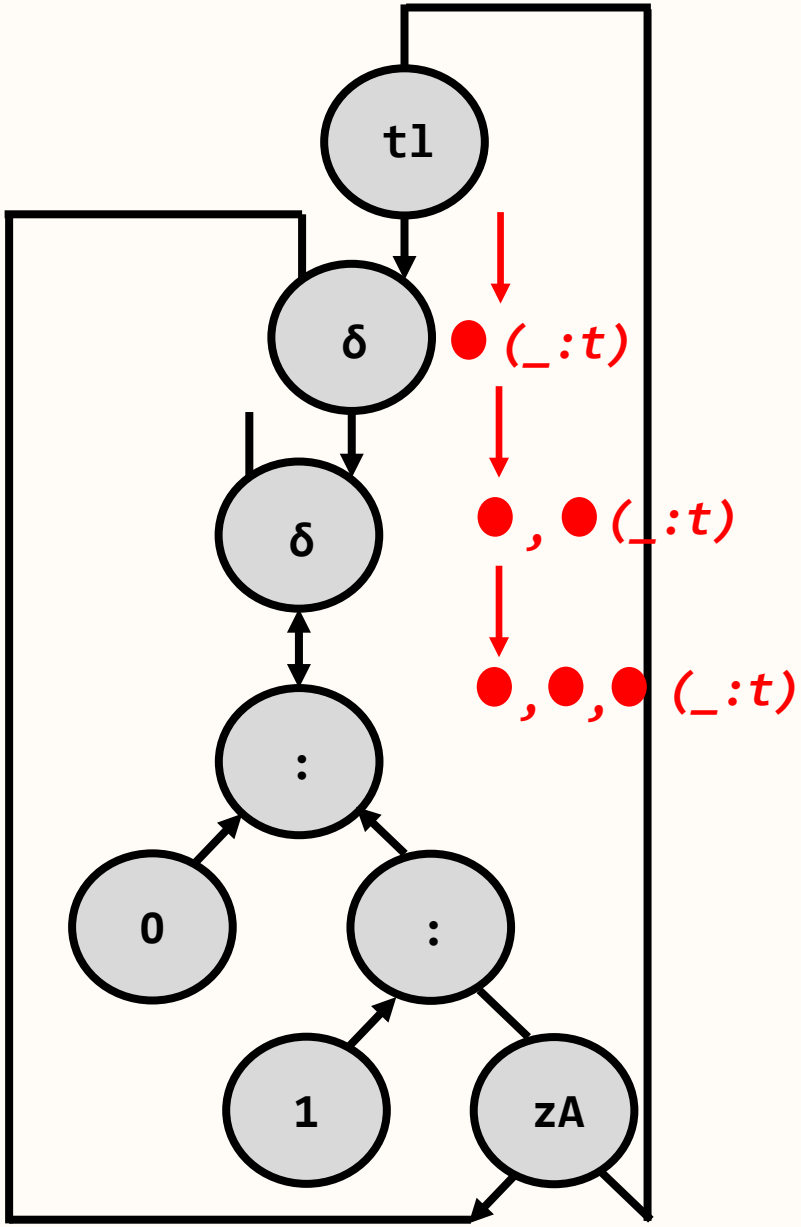
The “tail trap”



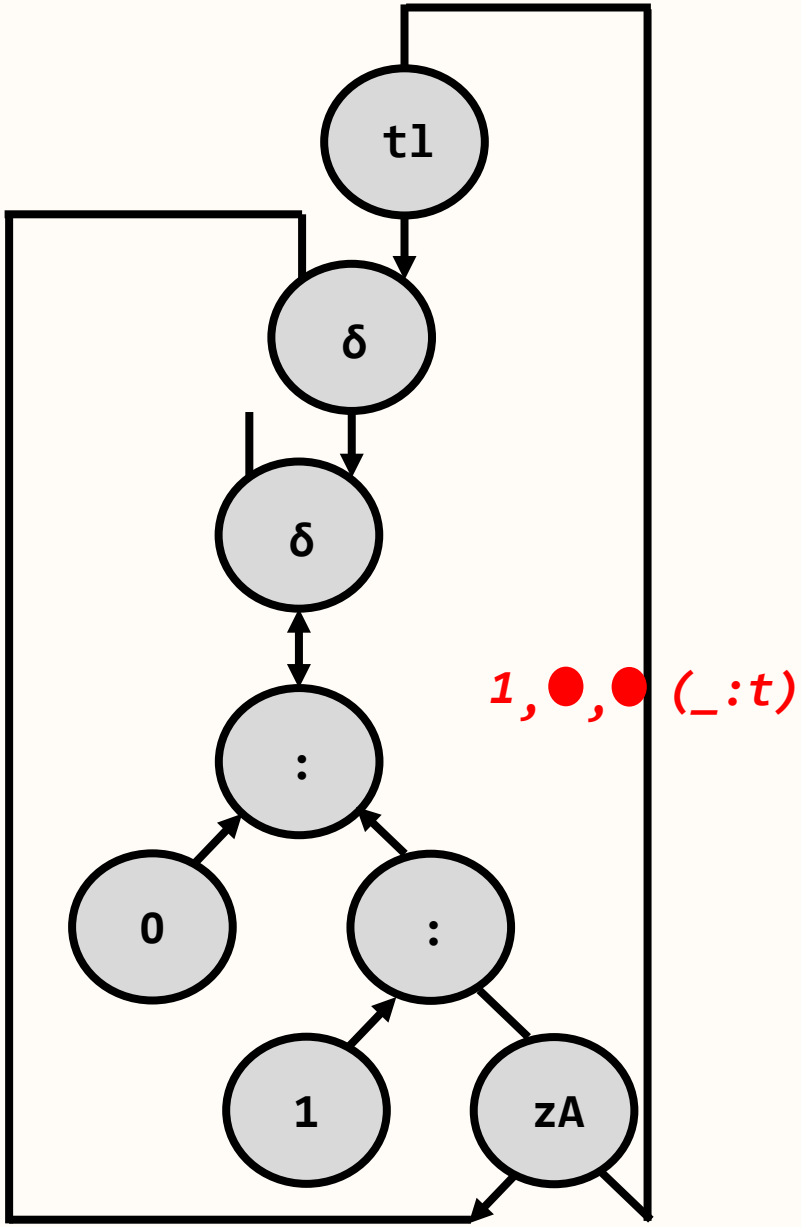
Avoid the “tail trap”



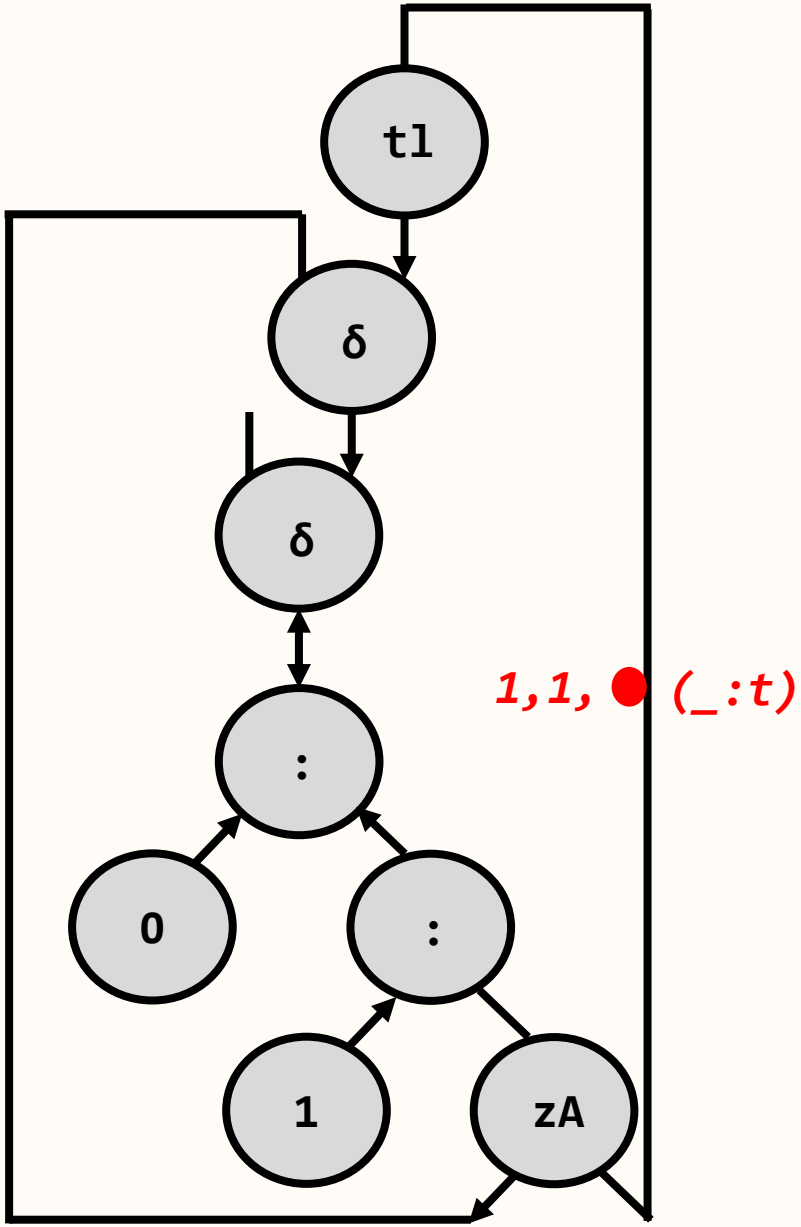
Stacking questions



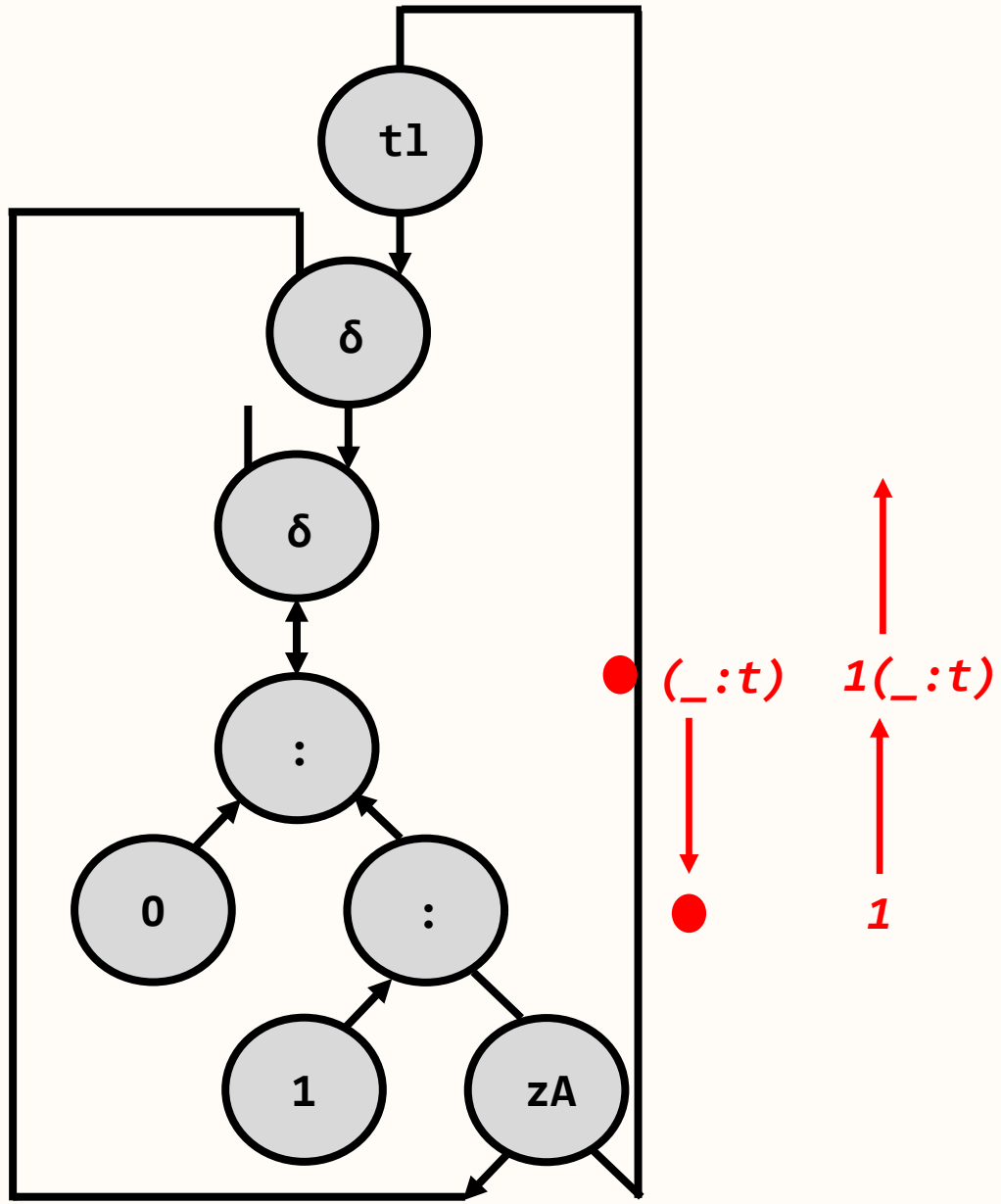
Stacking questions & answering



Stacking questions & answering



Stacking questions & answering



Does pebble value exist?

Recall: Pebble value at port *x* = limit of number pebbles at port *x*.

Given an output port *x*, define set of paths through net & path length function.

- **Every path has a length (tuple => max).**
- **Looping paths have length = ω.**

Then order relation on path lengths is a poset with infimum (0) and supremum (ω).

Therefore is complete lattice => least fixed point.

How interpret it?

Pebble value	Output size
0	At least zero!*
$n \in \mathbb{N}^+$	At least *n* (maybe stream)
ω	Stream

*** The "tail trap".**

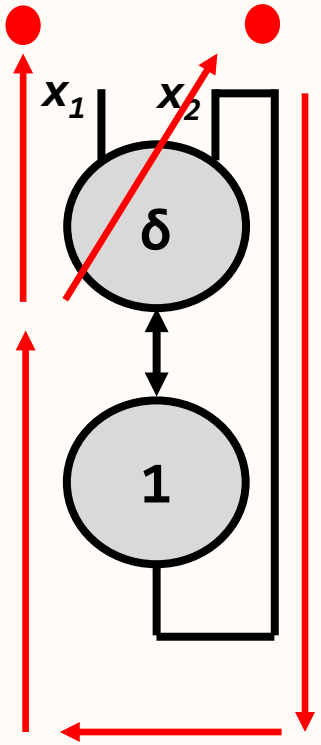
Interaction nets are Turing Complete; therefore any non-trivial non-trivial semantic property is in general undecidable (Rice's Thm, 1953).

So is this useless?

Can we calculate it?

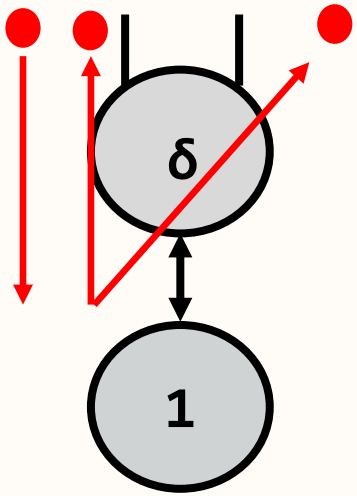
In some cases yes - using paths.

Example:



If path repeats with all agent states returning to 0 and pebble ends in same state, then determinism => non-termination.

If process terminates, result is finite.



Some questions

Can we iterate? 1-step productive ; 2-step etc

Which nets beget paths we can analyse?

Can we put some bounds on path lengths?

Better ways to avoid the “tail trap”?

Can we enhance pebble to get static evaluation à la GoIM?

Some questions

Can we iterate? 1-step productive ; 2-step etc

Which nets beget paths we can analyse?

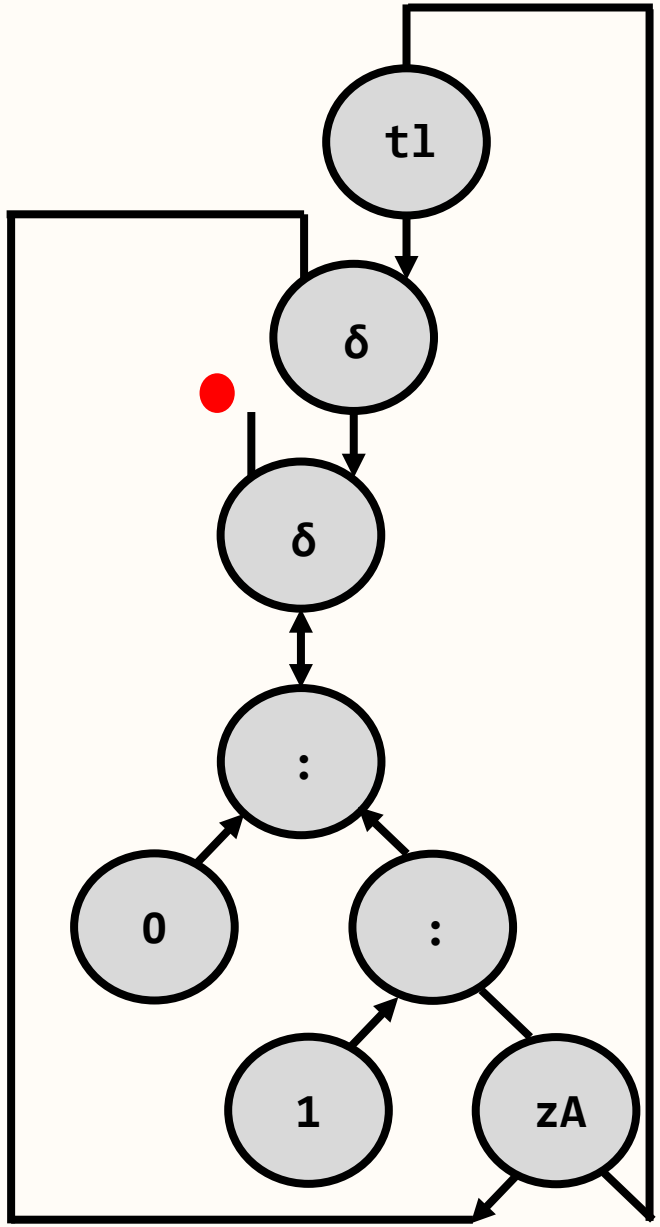
Can we put some bounds on path lengths?

Better ways to avoid the “tail trap”?

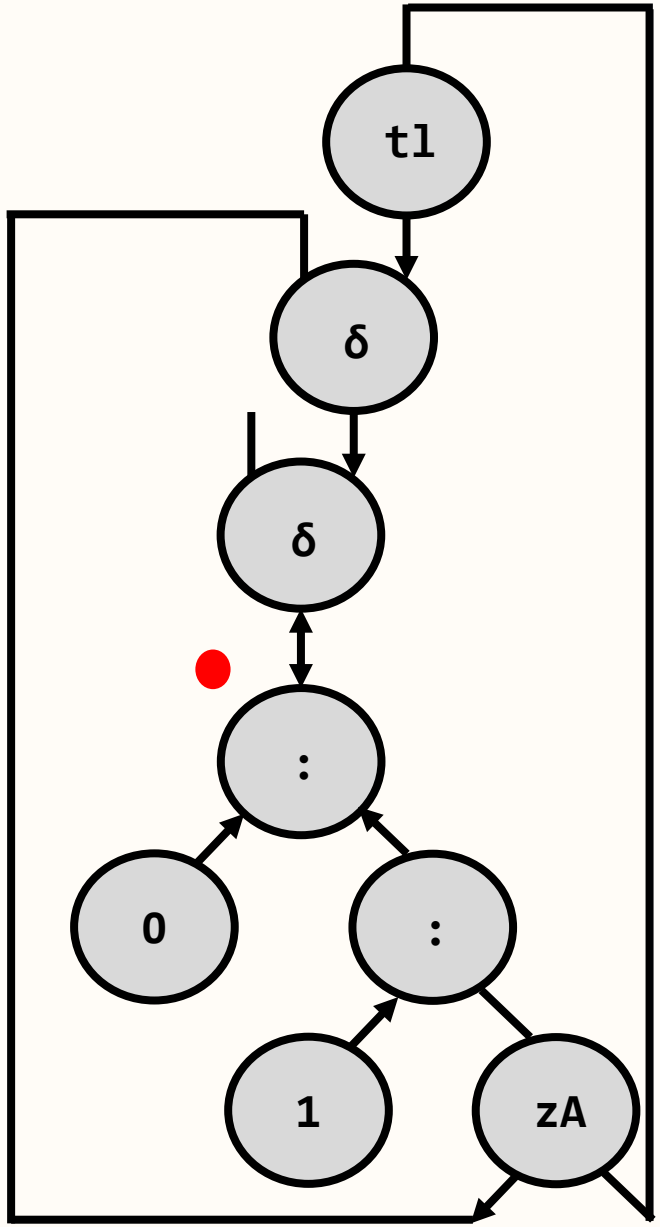
Can we enhance pebble to get static evaluation à la GoIM?

Grand finale....

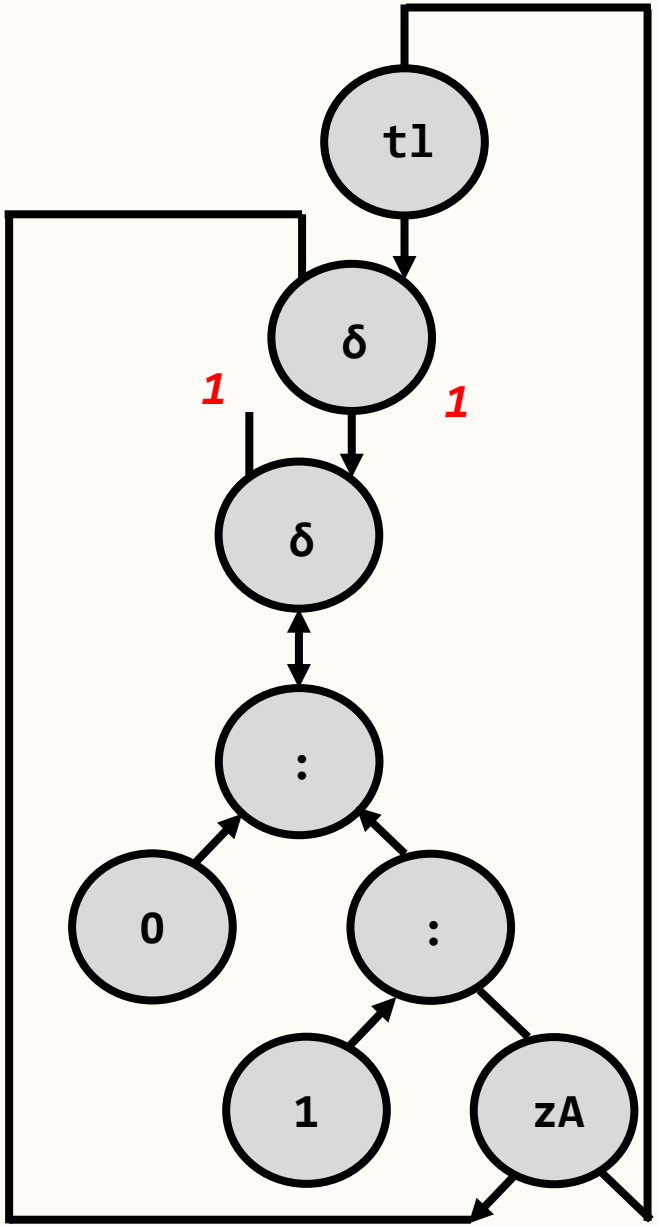
Pebble analysis on fib net



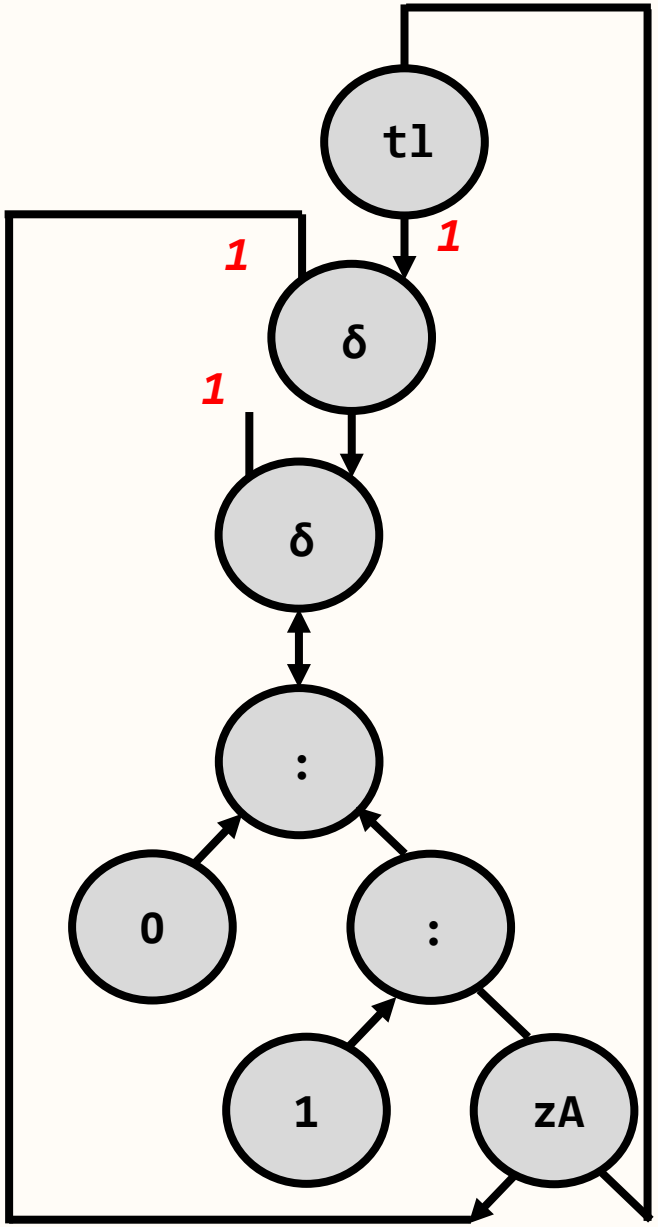
Pebble analysis on fib net



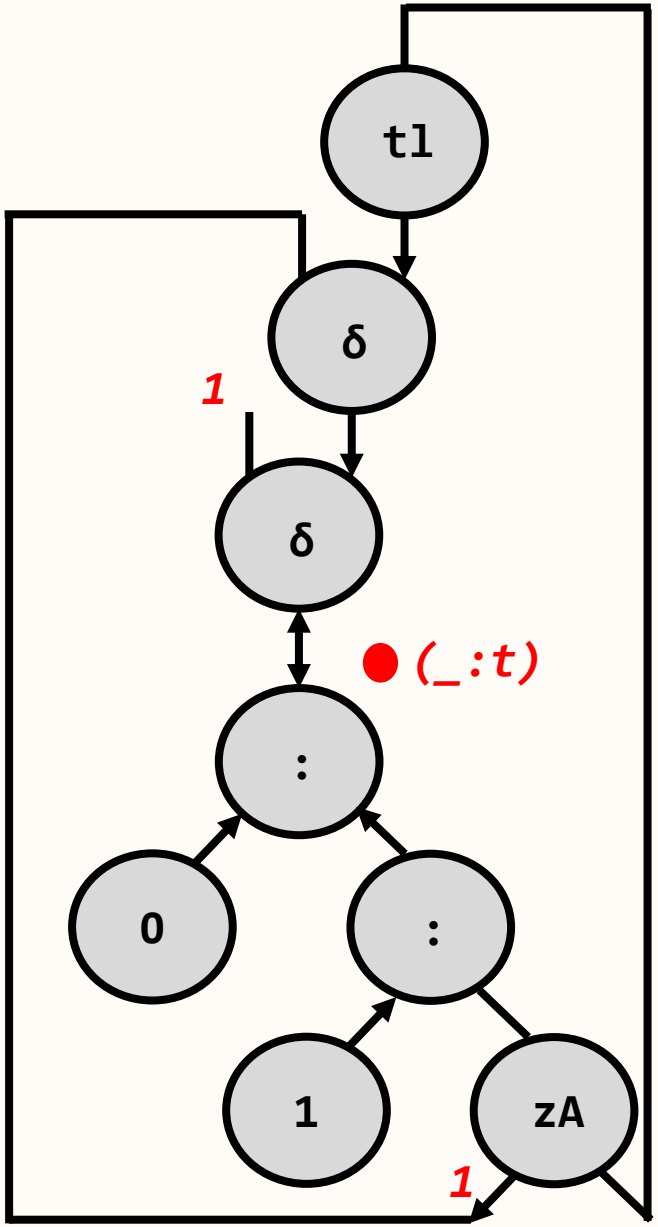
Pebble analysis on fib net



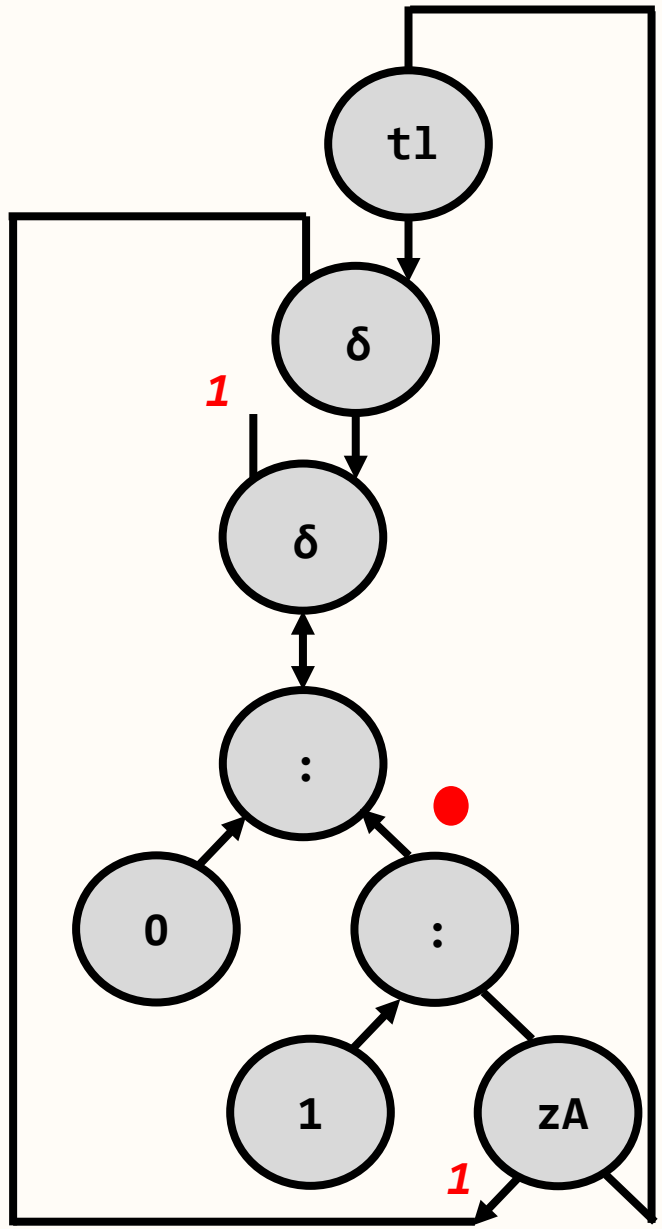
Pebble analysis on fib net



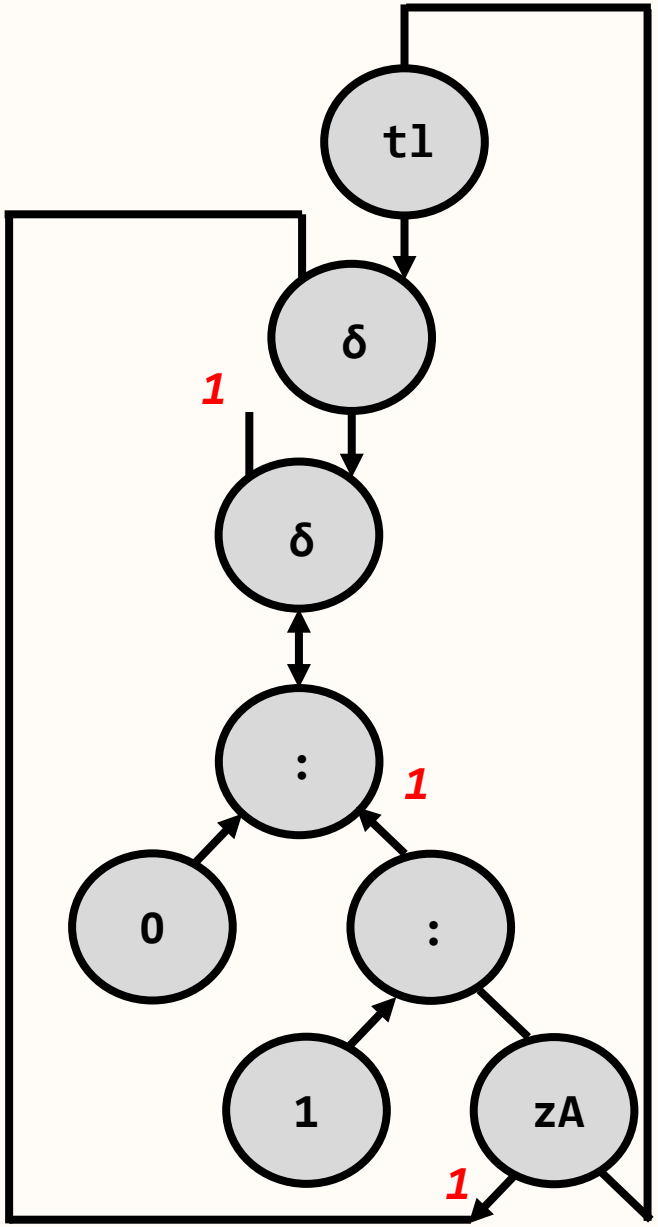
Pebble analysis on fib net



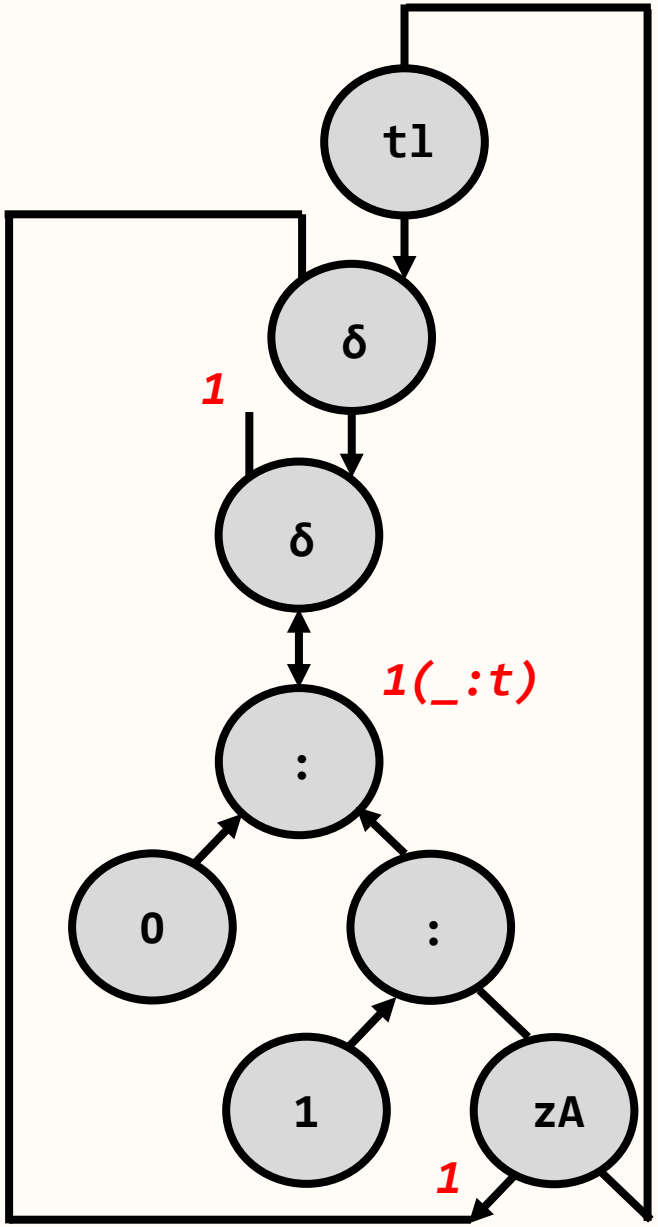
Pebble analysis on fib net



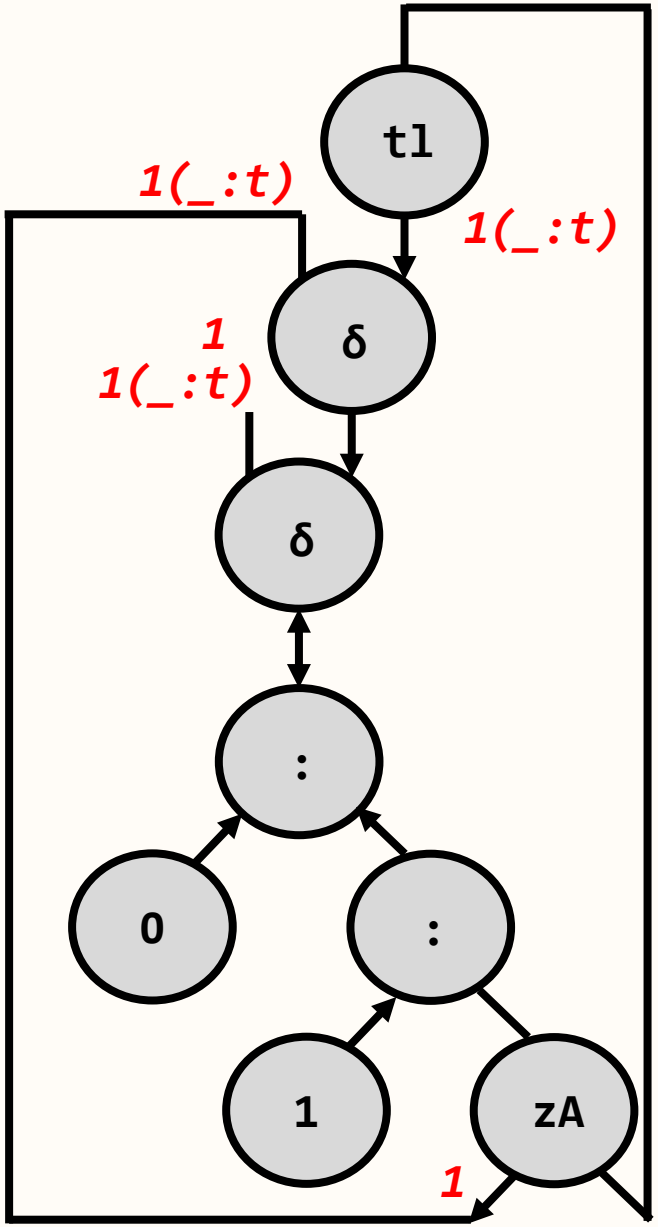
Pebble analysis on fib net



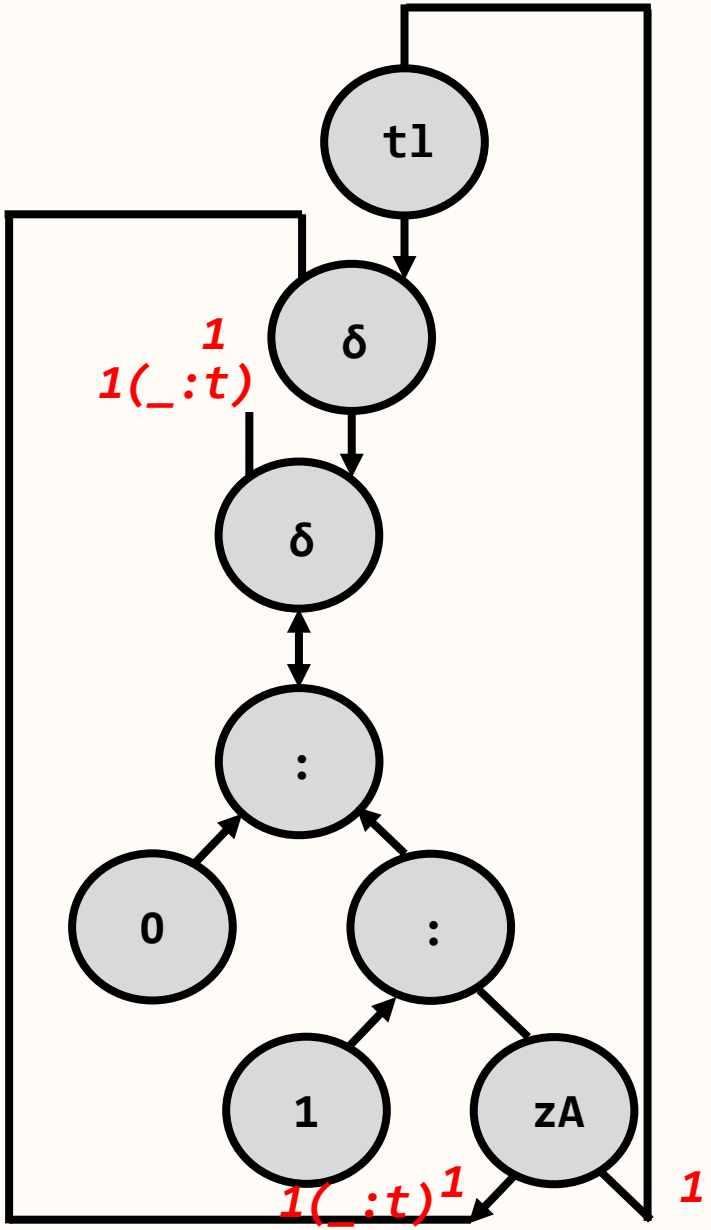
Pebble analysis on fib net



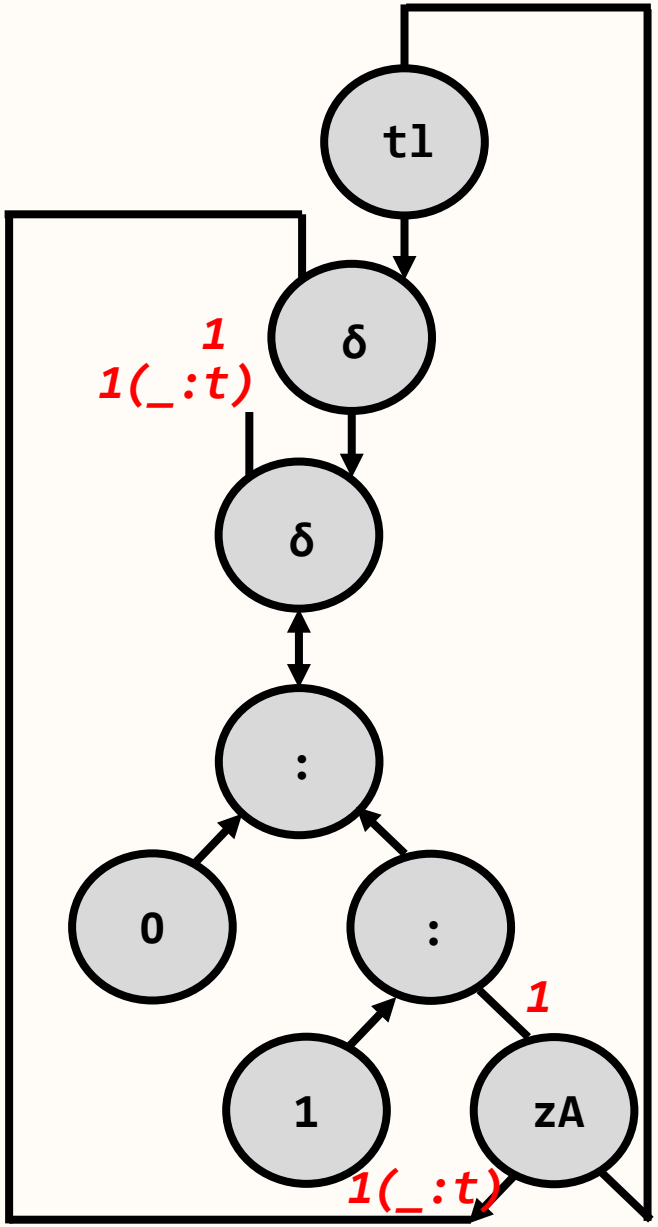
Pebble analysis on fib net



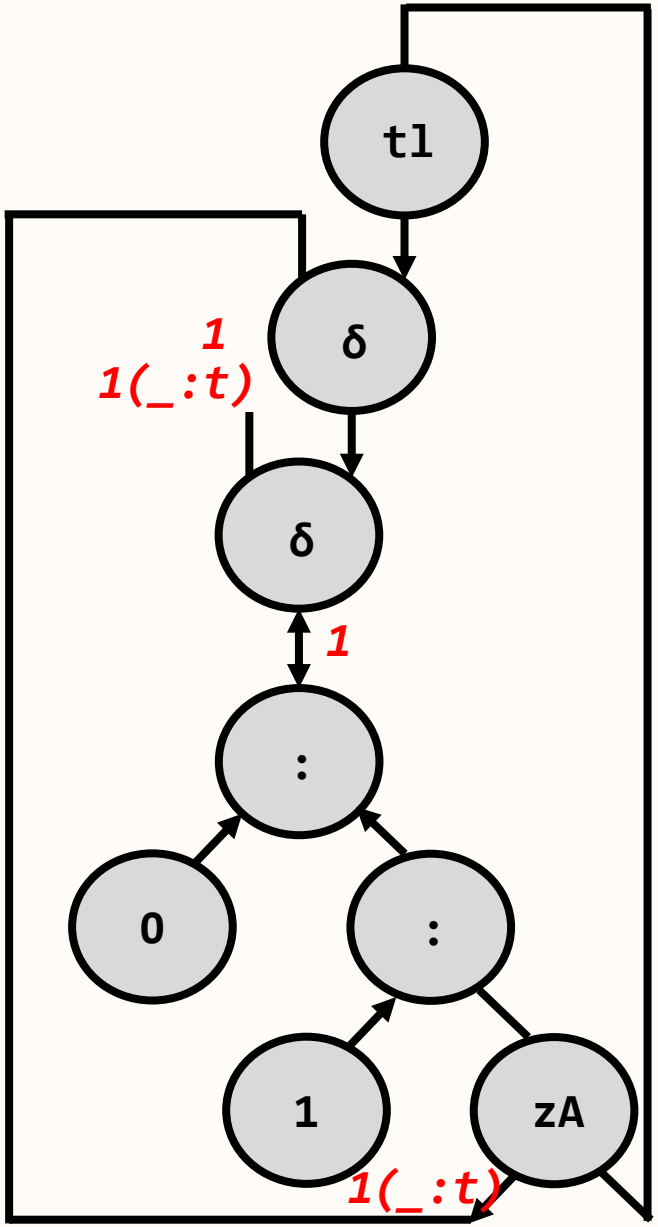
Pebble analysis on fib net



Pebble analysis on fib net



Pebble analysis on fib net



Thanks for listening!

Questions?

Comments?