Function-Constructor Nets and their Semantics

DIALOCO - 1st Workshop on DIAgrams in LOgic and COmputation

Marc Thatcher

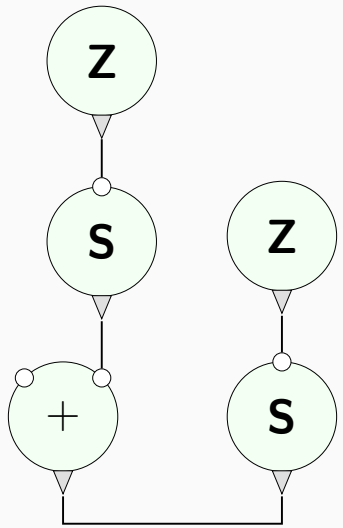
University of Sussex

Lisbon, 19th July 2026

Interaction nets

Interaction nets (Lafont, POPL90)

MLL+units+mix proof structures + definable node labels and cut rules = interaction nets.



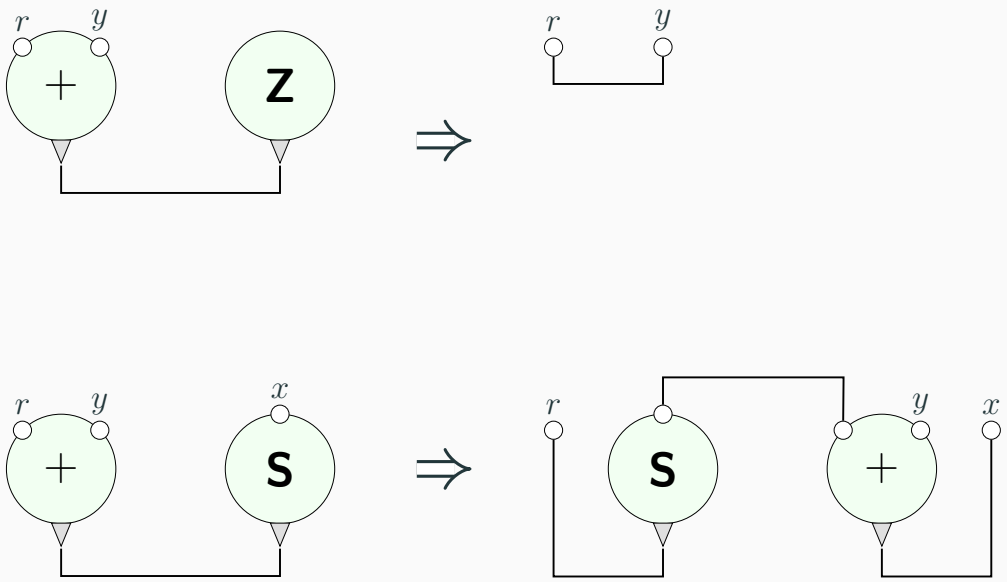
Interaction net = finite port graph ; nodes (*agent*) + edges (*wires*).

Agent label $\in \Sigma$; $ar : \Sigma \to \mathbb{N}$; $ar(\sigma)$ *auxiliary* ports .

May be cyclic, empty, disjoint.

Interaction (rewrite) rules defined between agents linked via *principal* ports.

Interaction rules

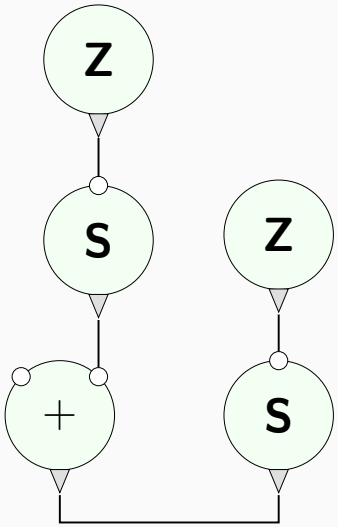


LHS (*active pair*) = two agents connected only by principal port.

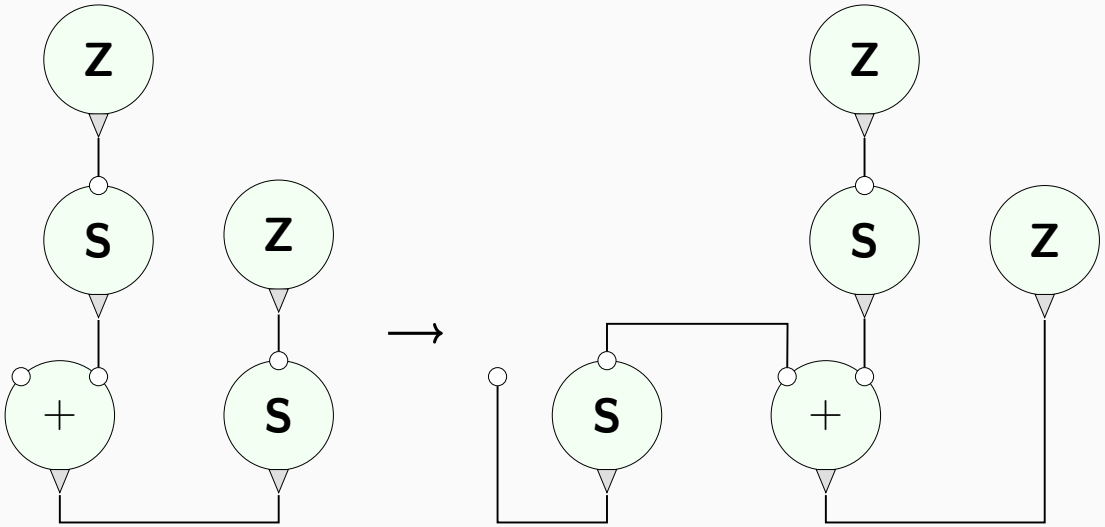
RHS = net that preserves LHS's *interface* (set of free ports).

LHSs are unique.

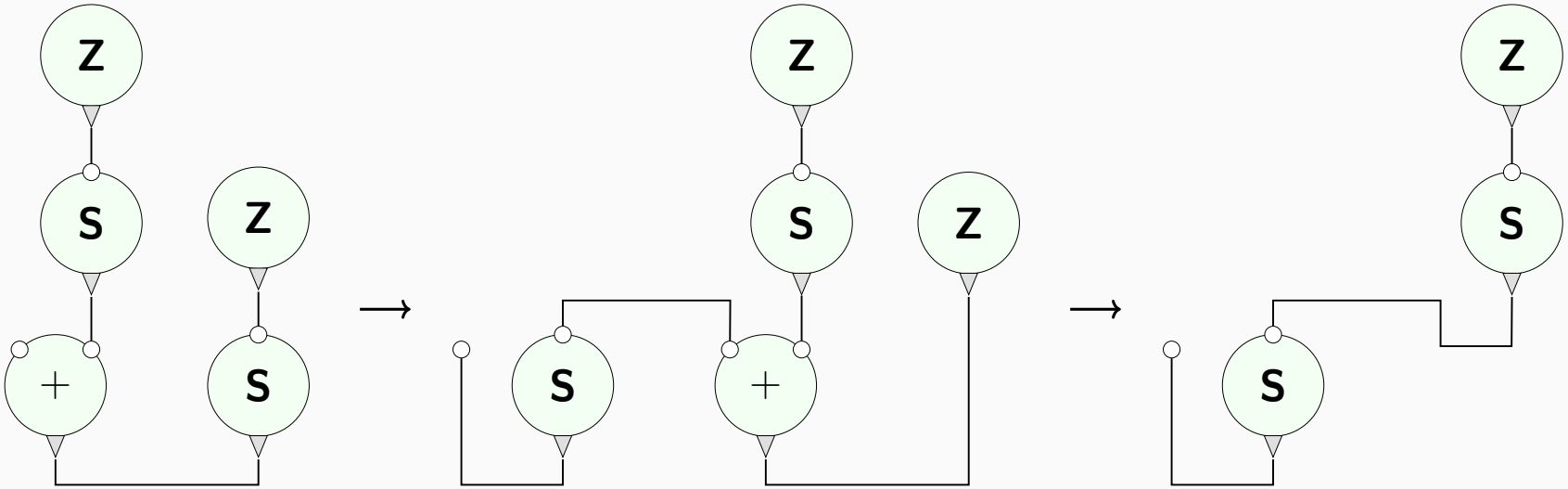
Interaction net evaluation



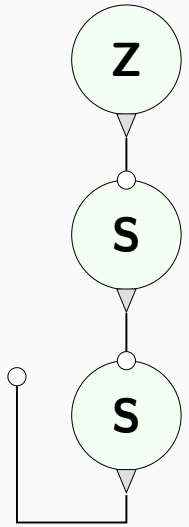
Interaction net evaluation



Interaction net evaluation



Result = normal form



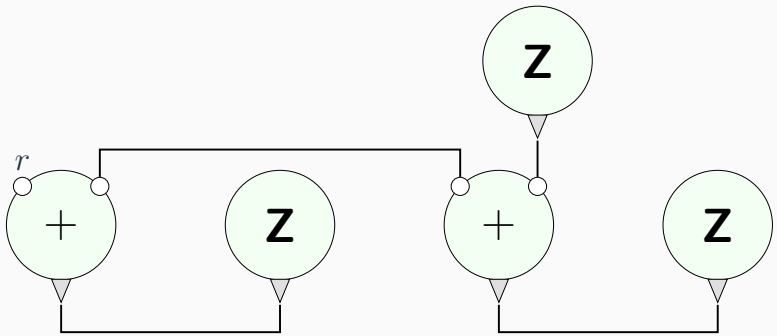
$$N \Downarrow M \iff N \to^* M \text{ and } \nexists M' \text{ s.t. } M \to M'$$

Interaction nets as programs

"We propose a new kind of programming language. . . "

— Yves Lafont (POPL90)

- Turing complete.
- Explicit resource management.
- Permutation equivalent reductions $\Longrightarrow$ parallel evaluation, unique normal forms.



Hence

- `INPLA` : `github.com/inpla/inpla`
- `Higher Order Co`: `higherorderco.com`
- `tendrils`: `tendrils.co`

Interaction nets as programs

Questions:

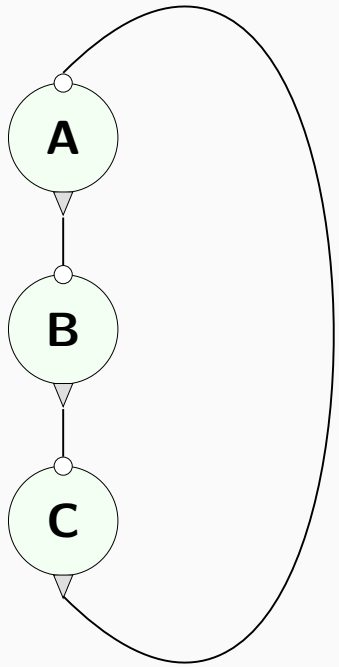
- What is input? Output?
- What are data?
- What are computations?
- What are results?
- How compose programs?
- ...

Recall:

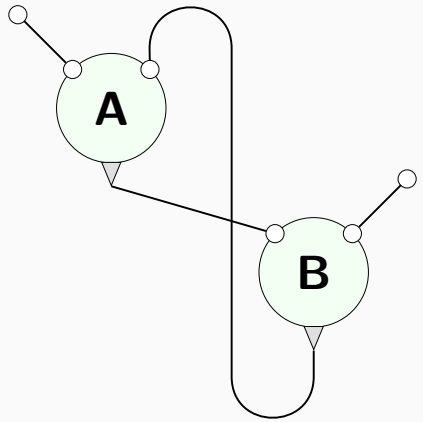
1. We can only reduce (compute) when there is an active pair.
2. Result of evaluation is normal form.

Problematic result #1 – Empty net

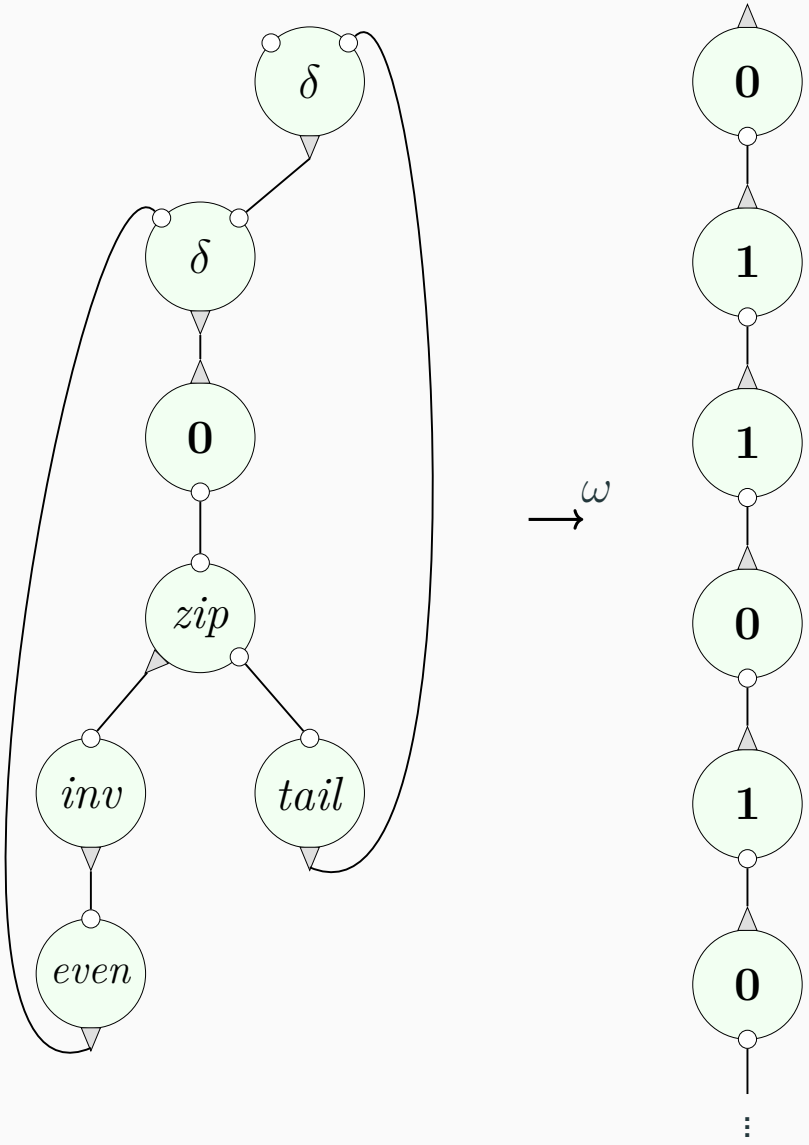
Problematic result #2 – Closed nets



Problematic result #3 – Vicious circles



Problematic non-result – Stream output



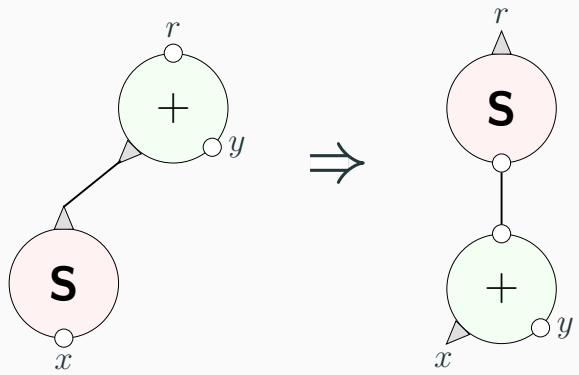
Function-constructor nets

Function-constructor nets

$\Sigma = \Sigma_{\mathcal{F}} \cup \Sigma_{\mathcal{C}}$; $\Sigma_{\mathcal{F}} \cap \Sigma_{\mathcal{C}} = \varnothing$

$\Sigma_{\mathcal{F}} = \{\epsilon/0,\ \mathsf{id}/1,\ \delta/2,\ \ldots,\ F/n_F\}$ — principal port is *input*.

$\Sigma_{\mathcal{C}} = \{\mathsf{Z}/0,\ \mathsf{S}/1,\ :/2,\ \ldots,\ C/n_C\}$ — principal port is (only) *output*.



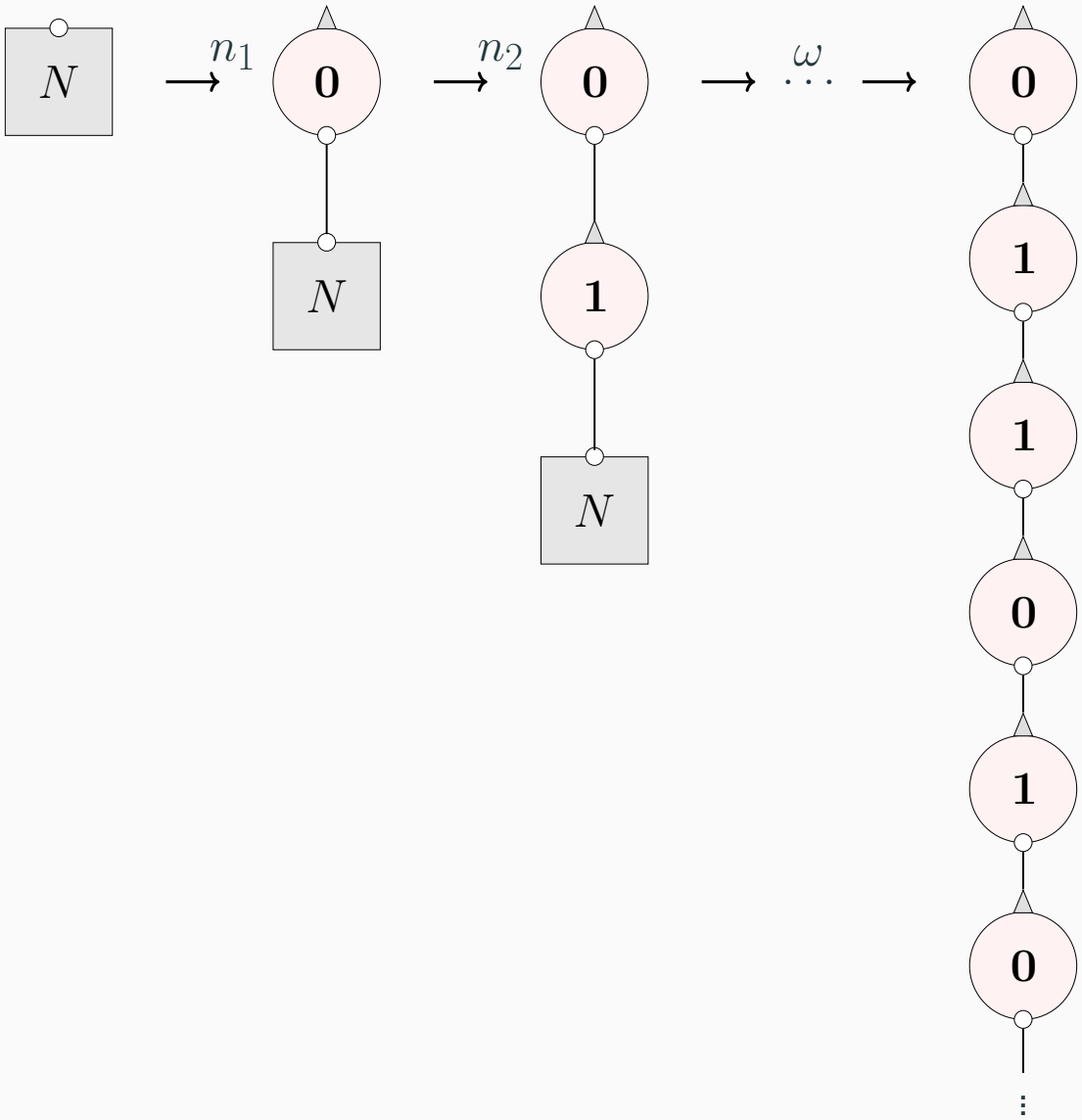
First-order system: constructors are function inputs and program outputs.

Still Turing complete.

$\implies$ interface, $I = I_{in} \cup I_{out}$; $I_{in} \cap I_{out} = \varnothing$.

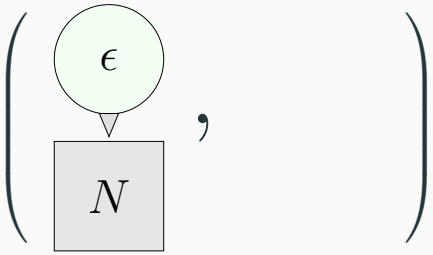
Consider nets with $I_{in} = \varnothing$.

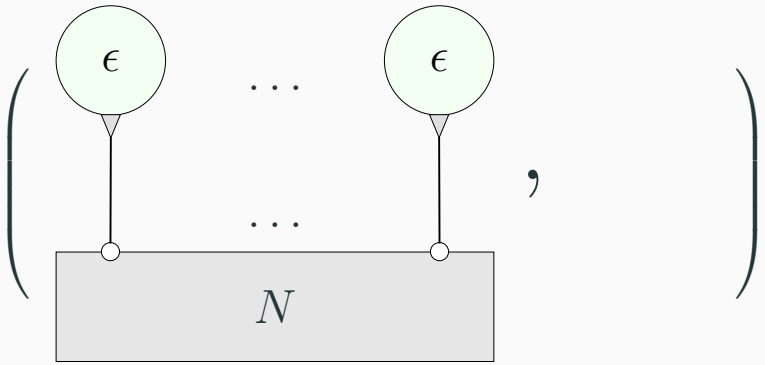
Result = observable output



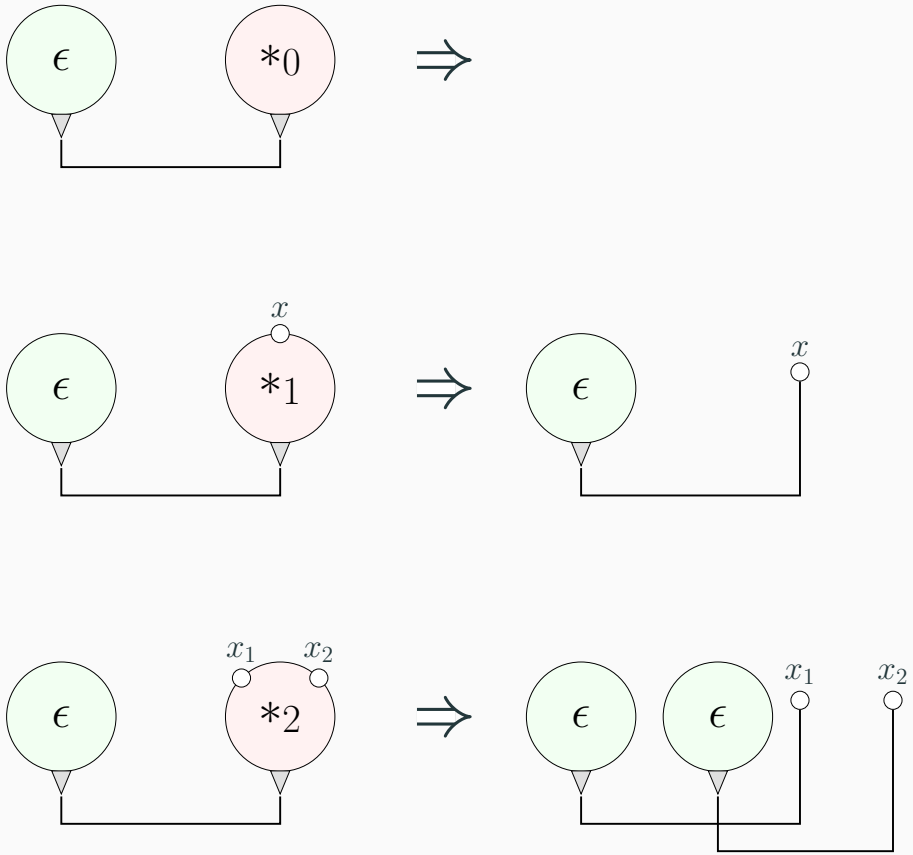
Operational semantics

Function-constructor net operational semantics—set up

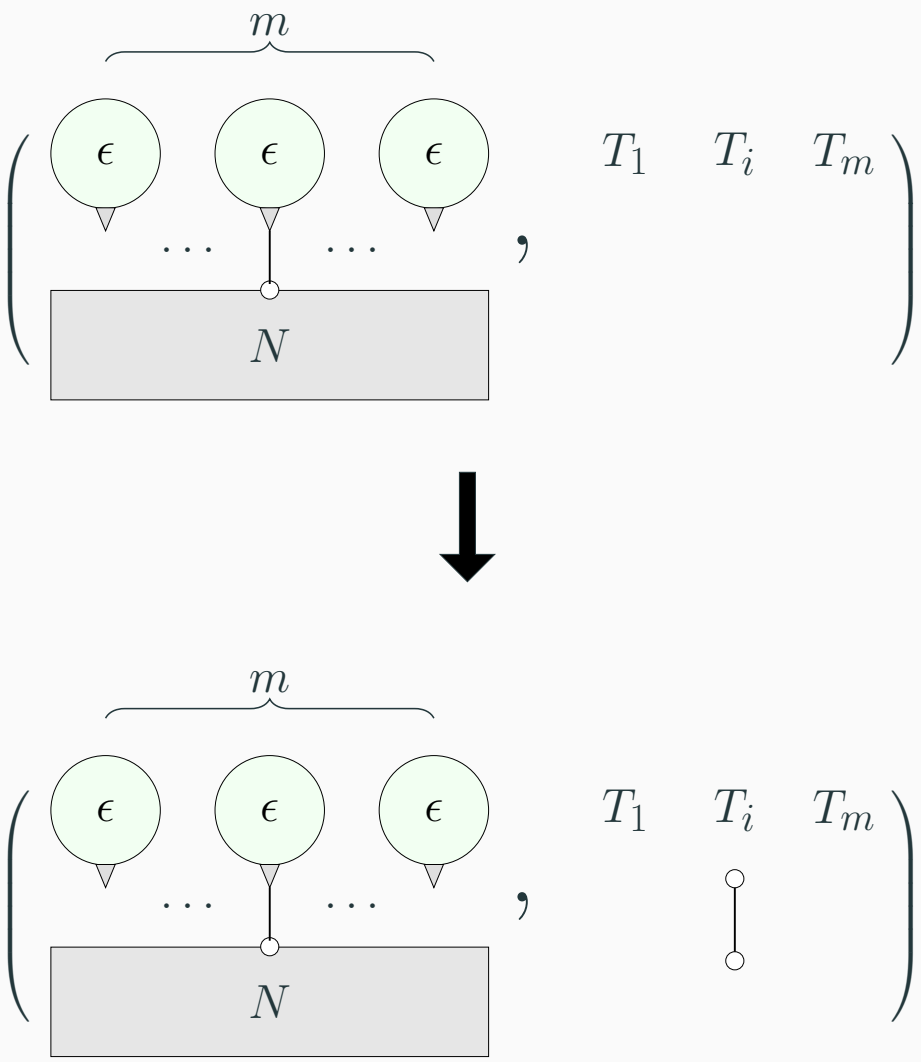




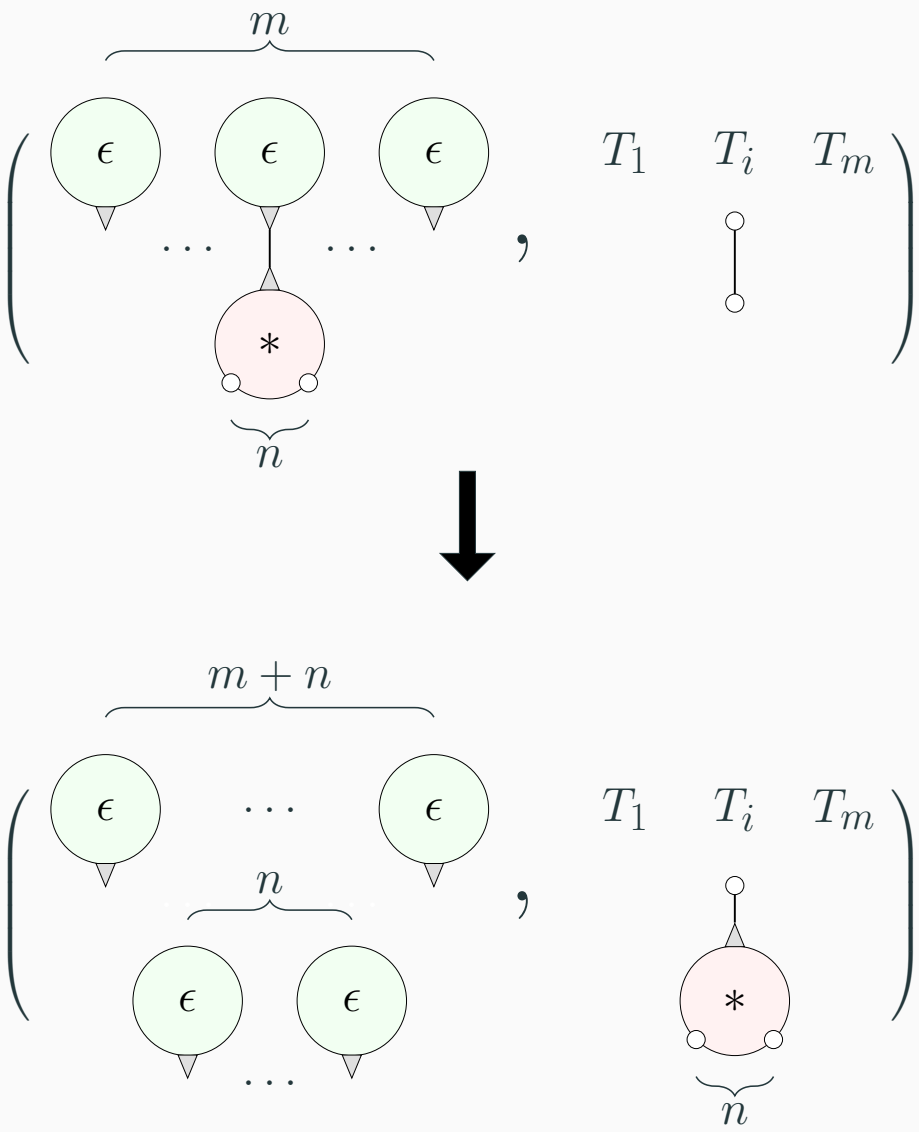
Interaction rules—erasure



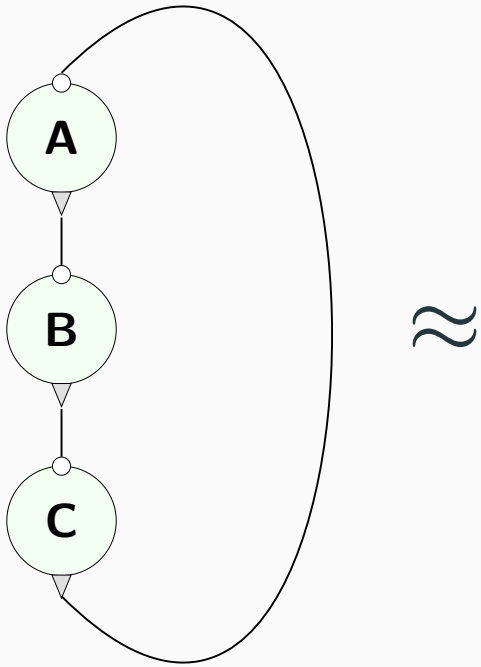
Function-constructor net operational semantics—dynamics #1



Function-constructor net operational semantics—dynamics #2

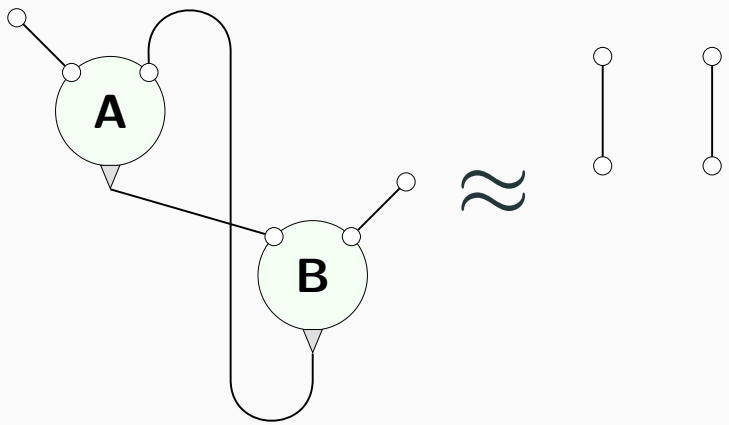


Observational value: Inaccessible nets

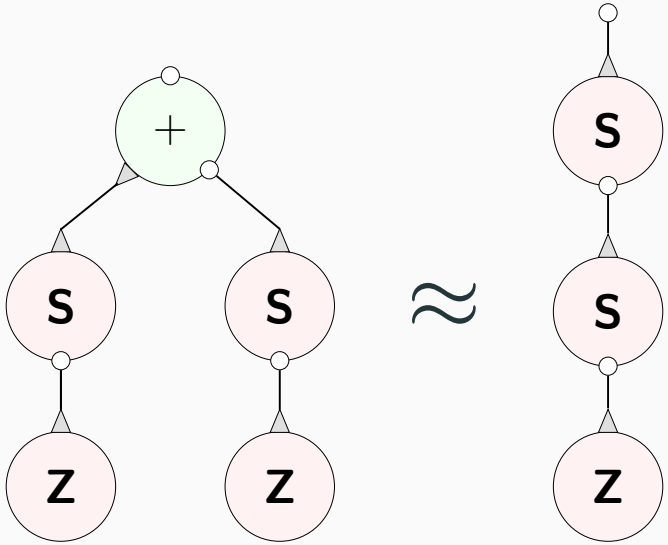


$\approx$

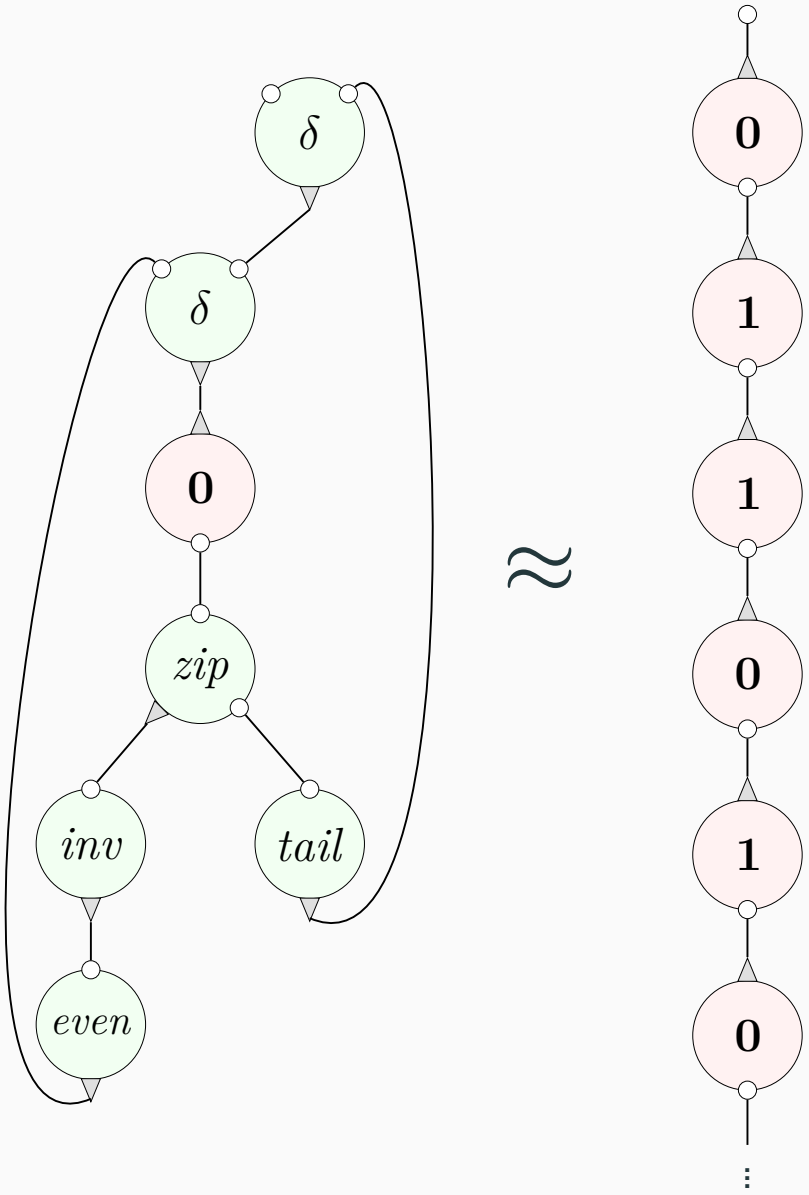
Observational value: Vicious circles



Observational value: Terminating productive nets



Observational value: Non-terminating productive nets



Denotational semantics

Semantic domain

At an output port: *constructor trees*, $\mathcal{T}$, defined coinductively by:

$$\mathcal{T} \ ::= \ \bot \ | \, ⅋(\mathcal{T}) \ | \, ⅋\big(C(\vec{\mathcal{T}})\big)$$

- $C \in \Sigma_C$
- Consecutive wires compose to a single wire.
- $\bot \sqsubseteq ⅋(\bot) \sqsubseteq ⅋(C(\bot)) \sqsubseteq ⅋(C(⅋(\bot))) \sqsubseteq ⅋(C(⅋(C()))) \ldots$
- $(\mathcal{T}, \sqsubseteq)$ is an ω-cpo.

For a k-output net:

$$\mathcal{T}^k \ ::= \ \underbrace{\mathcal{T} \times \cdots \times \mathcal{T}}_{k}$$

Interpretation function

$$[\![\cdot]\!]_{(\cdot)} \ : \ \mathcal{N} \times \mathcal{P}_{out} \longmapsto \mathcal{T}$$

$$\left[\!\!\left[\quad \begin{array}{c} \circ \\ | \\ \circ \\ x \end{array} \quad \right]\!\!\right] = \begin{array}{c} \circ \\ | \\ \circ \end{array}([\![x]\!])$$

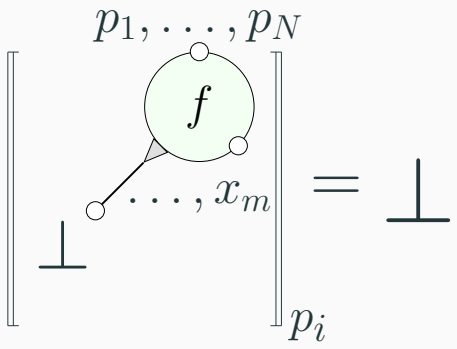
$$\left[\!\!\left[\quad \begin{array}{c} \circ \\ | \\ \circ \end{array} \quad \right]\!\!\right] = \begin{array}{c} \circ \\ | \\ \circ \end{array}(\bot)$$

Interpretation function

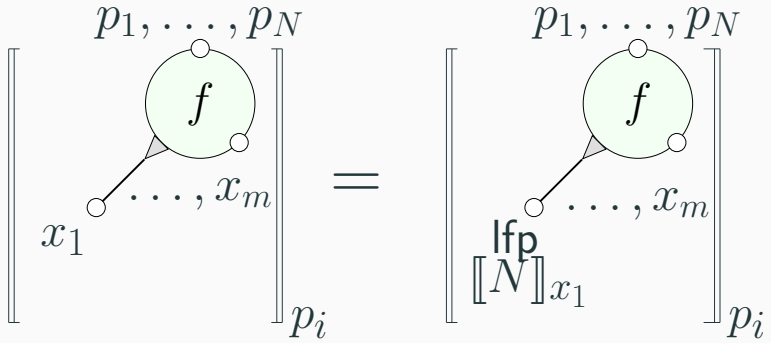
$$\left[\!\!\left[\begin{array}{c} C \\ y_1, \ldots, y_n \end{array} \right]\!\!\right] = \wr(C([\![y_1]\!], \ldots, [\![y_n]\!])$$

$$\left[\!\!\left[\begin{array}{c} p_1, \ldots, p_N \\ f \\ \ldots, x_m \\ C \\ y_1, \ldots, y_n \end{array} \right]\!\!\right]_{p_i} = [\![\mathsf{RHS}(y_1, \ldots, y_n, x_2, \ldots, x_m)]\!]_{p_i}$$

Interpretation function



$= \bot$



Existence of the least fixed point

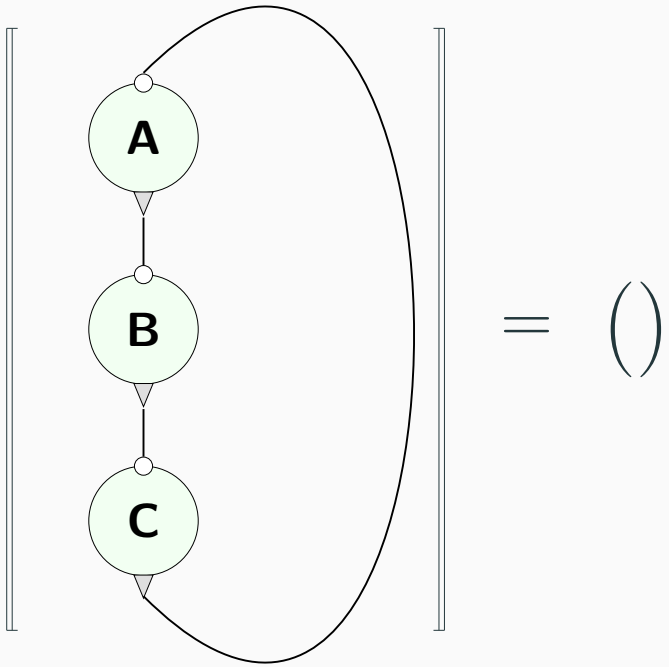
By Kleene's Fixed Point Theorem:

- $(\mathcal{T}, \sqsubseteq)$ is an ω-cpo; and
- Induced functional F is Scott-continuous.

Scott-continuity of F:

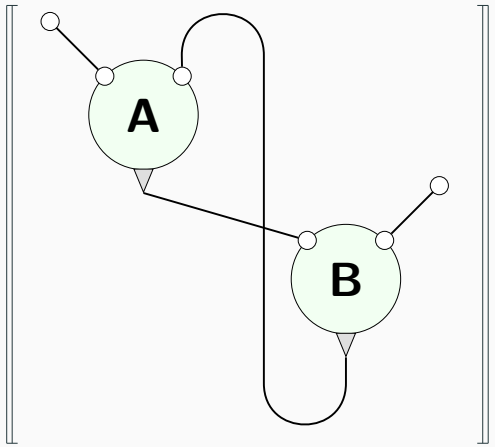
- Each clause is monotone;
- Each clause inspects only a *finite* constructor prefix of its arguments.

Denotational value: Inaccessible nets



$$= \quad ()$$

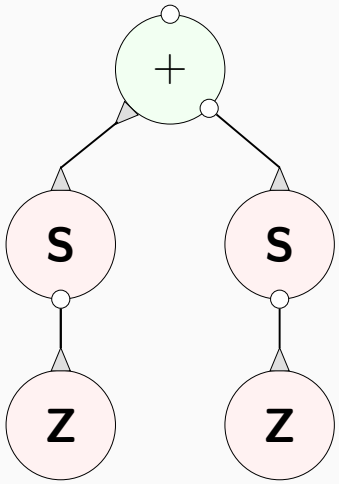
Denotational value: Vicious circles



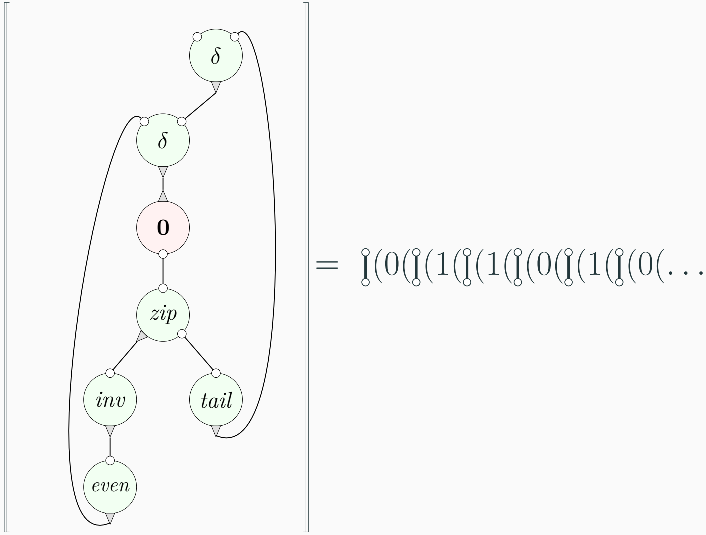
$$= \quad (\mathbb{I}(\bot), \mathbb{I}(\bot))$$

Denotational value: Terminating productive nets

$$\left[\!\!\left[\right.\right. \quad \left.\left.\right]\!\!\right] = \quad \lfloor(S(\lfloor(S(\lfloor(Z())\rfloor)\rfloor)\rfloor)$$



Denotational value: Non-terminating productive nets



Takeaways

> *"We propose a new kind of programming language. . . "*
>
> — *Lafont*

Taking this seriously leads to:

- Function-constructor nets with input/output and computation/data distinguished.
- Productivity replacing normal forms.
- Operational semantics.
- Denotational semantics.

And there's more: higher-order terms, a type system, textual syntax, effects, implementation. . .