Theory and Applications of Interaction Nets

Ian Mackie

ISR 2008: 3rd International School on Rewriting
21- 26 July 2008, Obergurgl, Austria

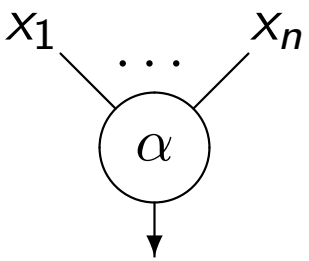
Organisation

- Introduction and examples
- Properties
- Calculus for interaction nets
- Types
- Programming with nets
- λ-calculus and strategies

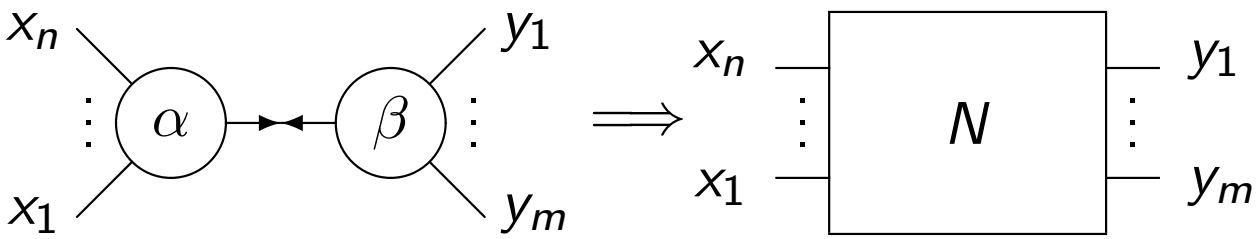
(Pure) Interaction Nets

A universal model of computation, based on net rewriting

Σ: Set of agents:



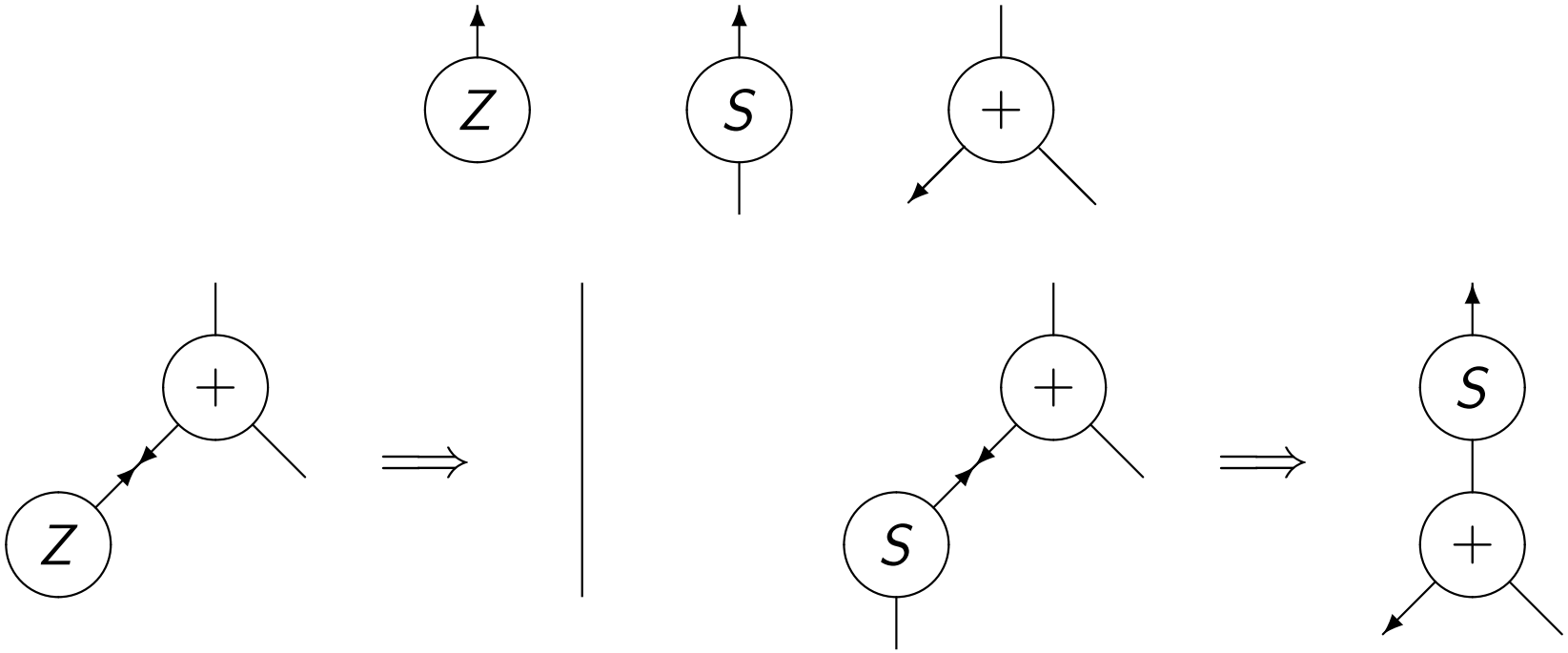
$\mathcal{R}$: Set of rewrite rules:



At most one rule for each pair of agents; interface is preserved

Example: Natural numbers

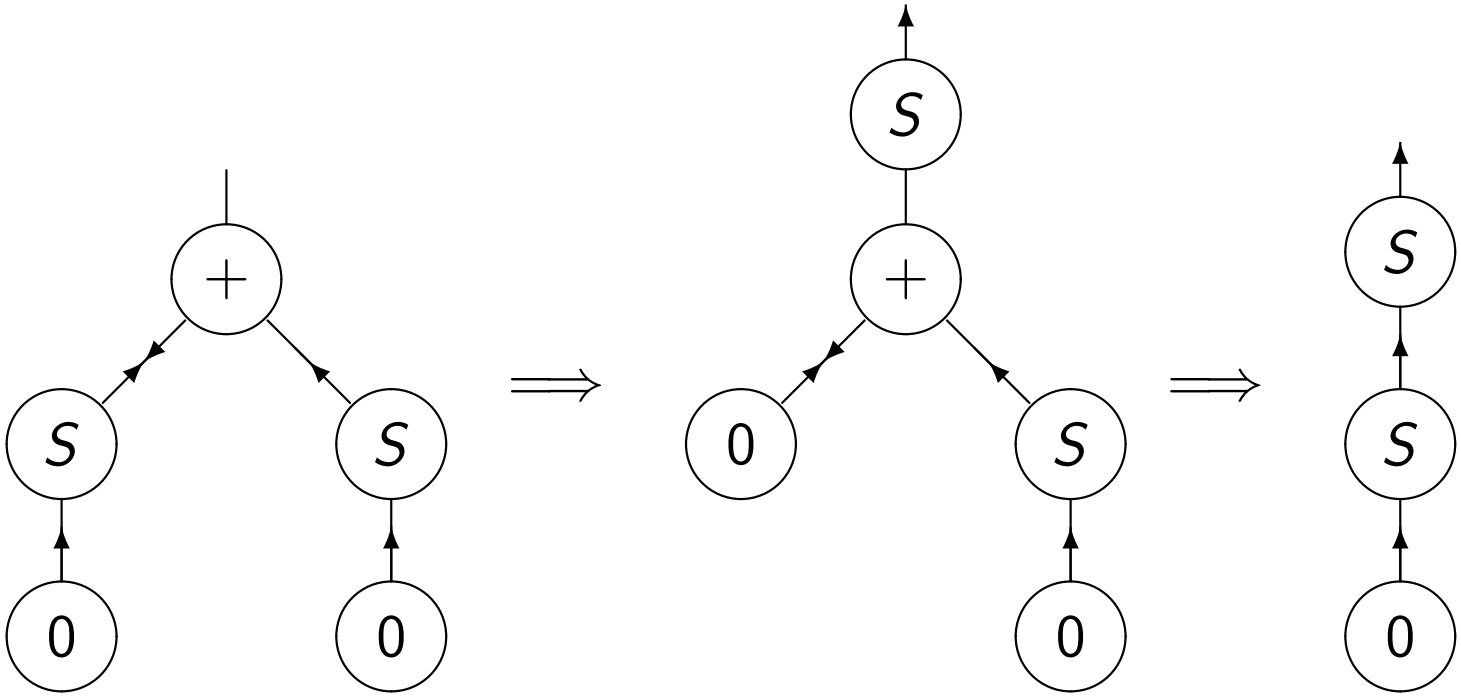
$$
\begin{array}{lcl}
add(0, x) & \rightarrow & x \\
add(S(x), y) & \rightarrow & S(add(x, y))
\end{array}
$$



Exercise: Can this be defined any other way?

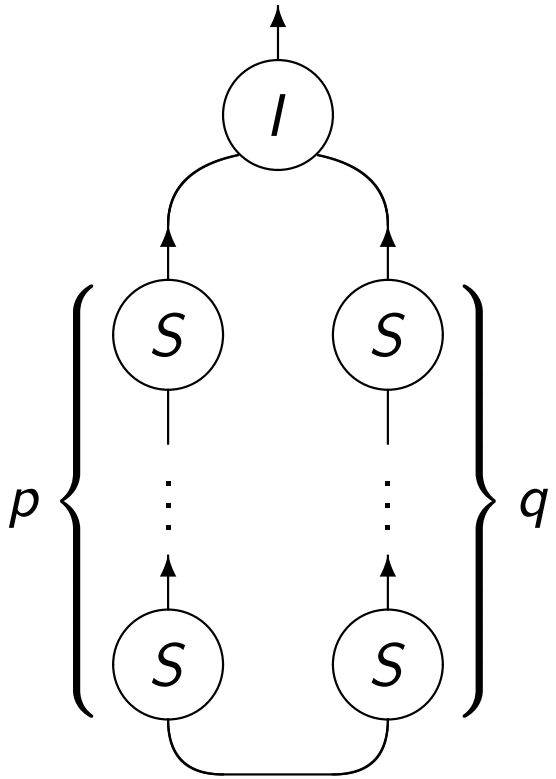
An example net and reduction

$S(0) + S(0)$ rewrites to $S(S(0))$



Exercise: Try $S(0) + (S(0) + S(0))$. There is a choice in the reduction sequence. What about $(S(0) + S(0)) + S(0)$?

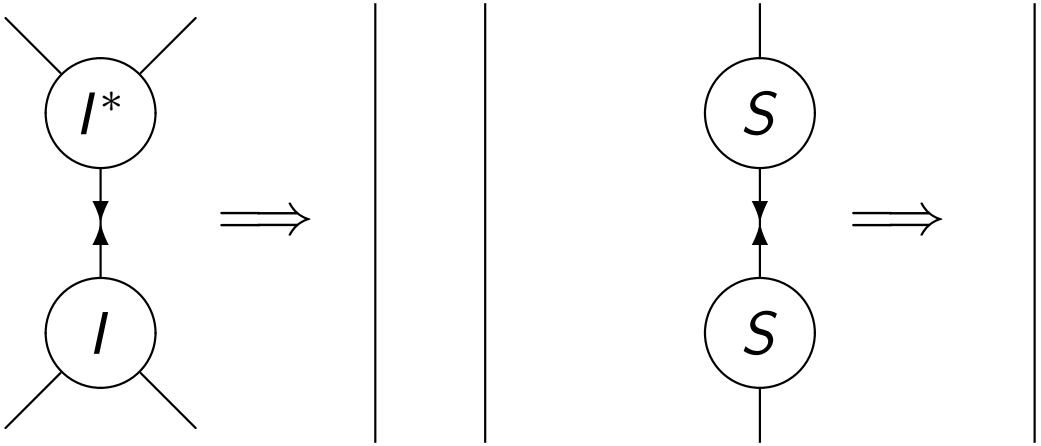
Example 2: Integers

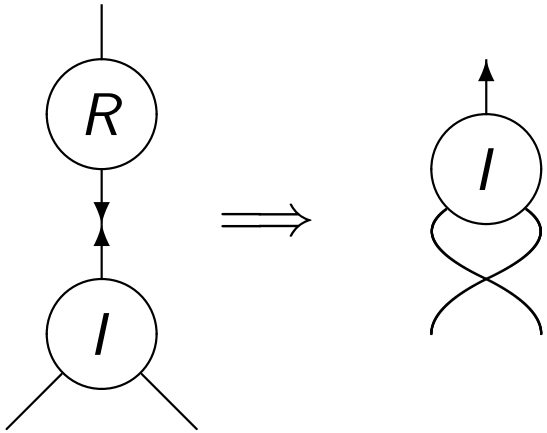


- Define $z = p - q$
- Representation not necessarily unique: e.g. 2=3-1, 2=10-8
- But there is a notion of canonical form

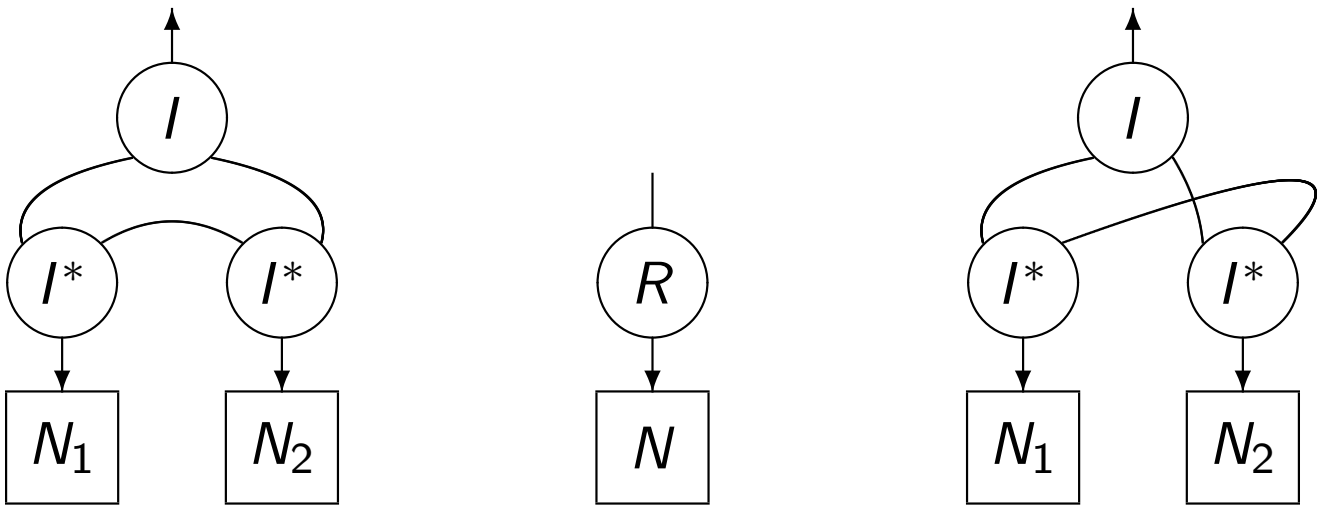
Exercise: Give agents and rules that will give the canonical form

Integers continued



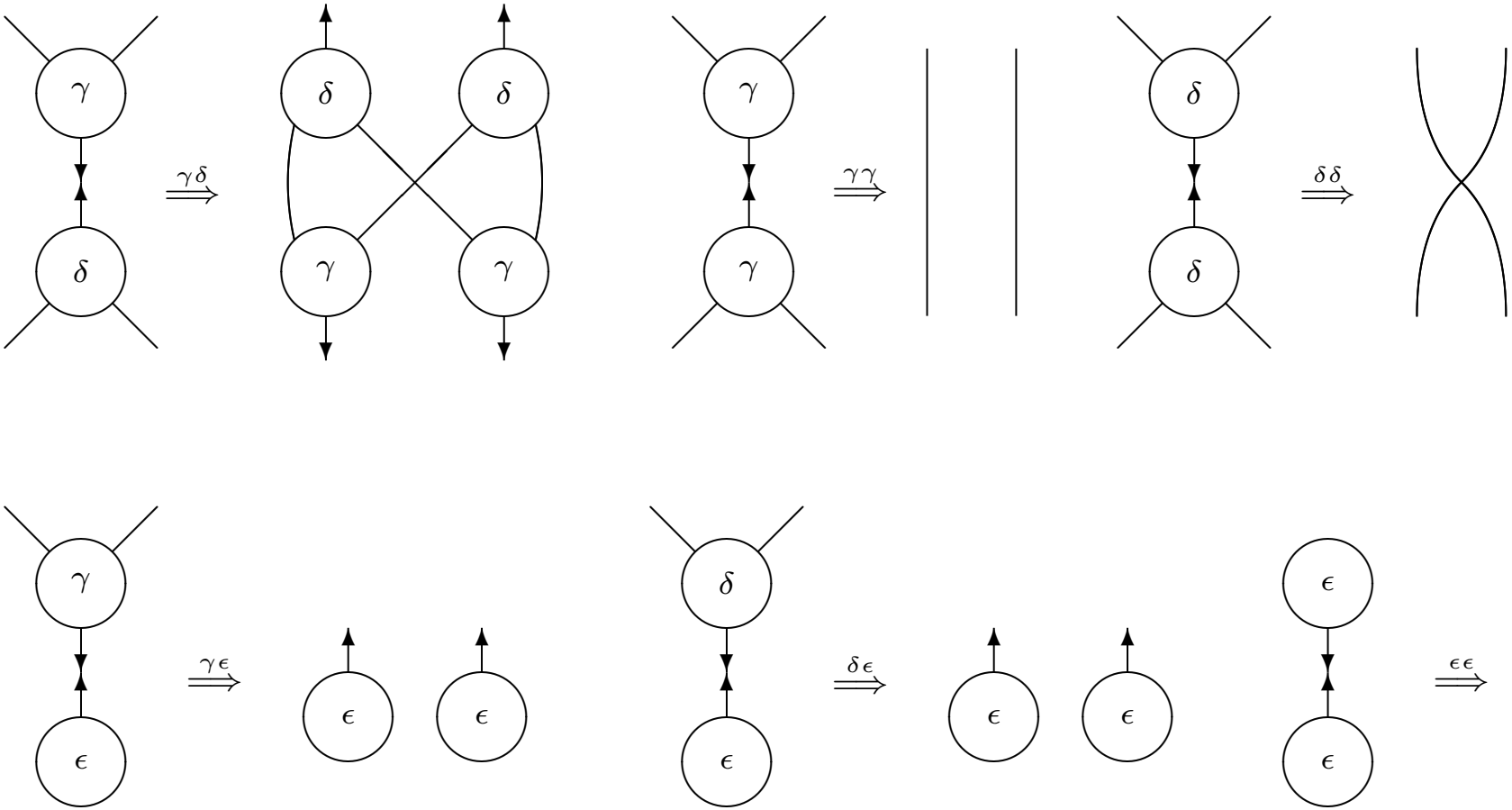


Addition, Negation, Subtraction



Exercise: What about multiplication?

Example 3: Interaction Combinators

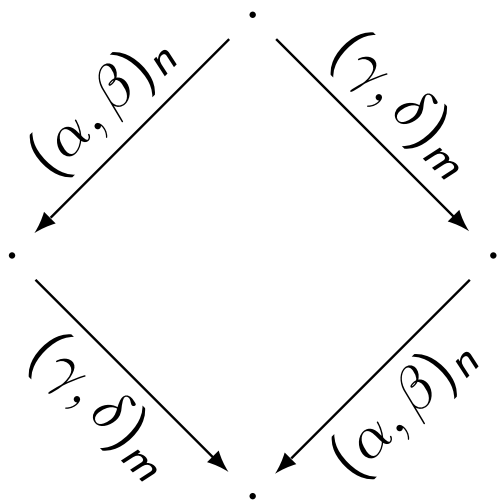


Why Interaction Nets?

- Language between specification and implementation
- Offers a low level operational semantics of computation (cf. explicit substitution calculi)
- Implementation technique: encodings of λ-calculus, functional programming languages, term rewriting systems, etc.
- Good properties: confluence, etc.
- Implementation: sequential and parallel
- Theory well developed

Properties

Diamond property:



for $n \neq m$. In particular active pairs are neither copied nor erased. Consequently, we have the following result

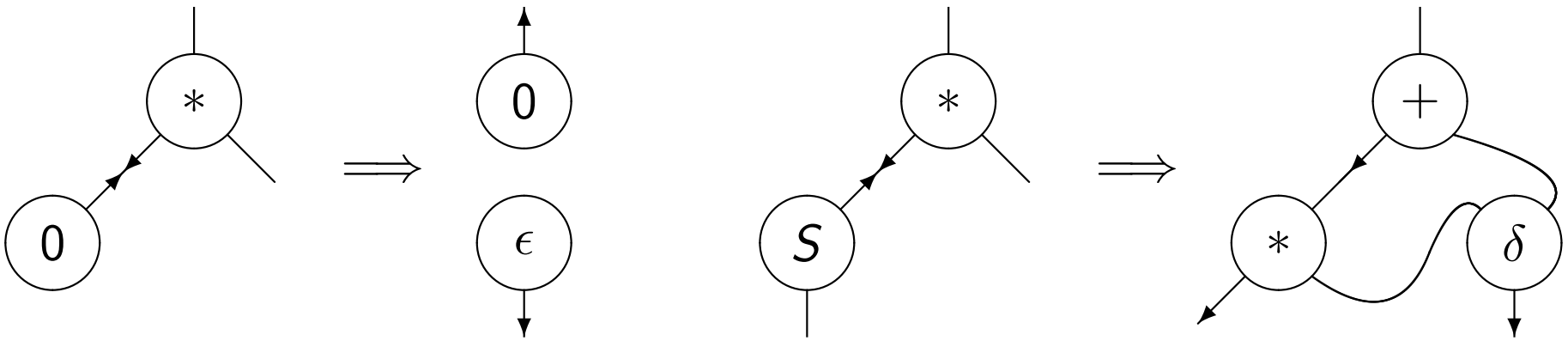
Theorem: Let N be a net in an interaction system $(\Sigma, \mathcal{R})$, then:

1. If $N \Downarrow N'$ then all reduction sequences are terminating (N is strongly normalizing).
2. Normal forms are unique: if $N \Downarrow N'$ and $N \Downarrow N''$ then $N' = N''$.

I.e. reductions sequences differ by permutation

Multiplication

$$
\begin{array}{lcl}
*(0, y) & \longrightarrow & 0 \\
(S(x), y) & \longrightarrow & +((x, y), y)
\end{array}
$$



Linearity: when a term is non-linear, we must do our own duplication and erasing

Maximum

How would we encode this example:

$$\begin{array}{lcl} \max(0, y) & \rightarrow & y \\ \max(x, 0) & \rightarrow & x \\ \max(S(x), S(y)) & \rightarrow & S(\max(x, y)) \end{array}$$

Pattern matching only on 1 argument (principal port) and only at depth 1

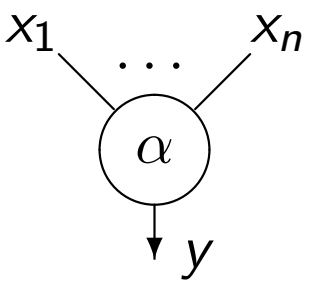
In this case there is an equivalent:

$$\begin{array}{lcl} \max(0, y) & \rightarrow & y \\ \max(S(x), y) & \rightarrow & \max'(x, y) \\ \max'(x, 0) & \rightarrow & S(x) \\ \max'(x, S(y)) & \rightarrow & S(\max(x, y)) \end{array}$$

Exercise: Define the corresponding interaction system

A calculus for interaction nets

Representing Agents: If we have an agent α of arity n:



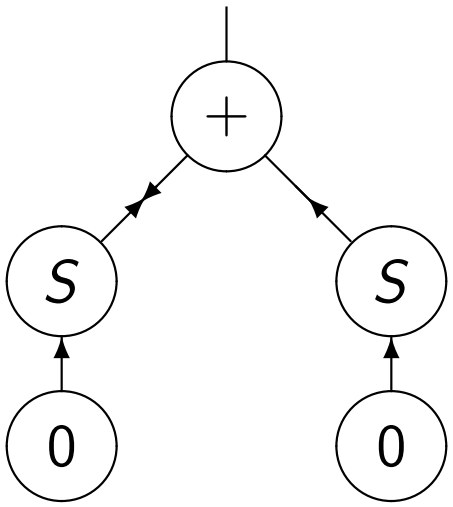
then we can write this textually as:

$$y = \alpha(x_1, \ldots, x_n)$$

Examples:

$$x = \mathsf{Z}() \qquad x = \mathcal{S}(y) \qquad x = \mathsf{Add}(y, z) \qquad x = \mathsf{Mult}(y, z)$$

Representing Nets

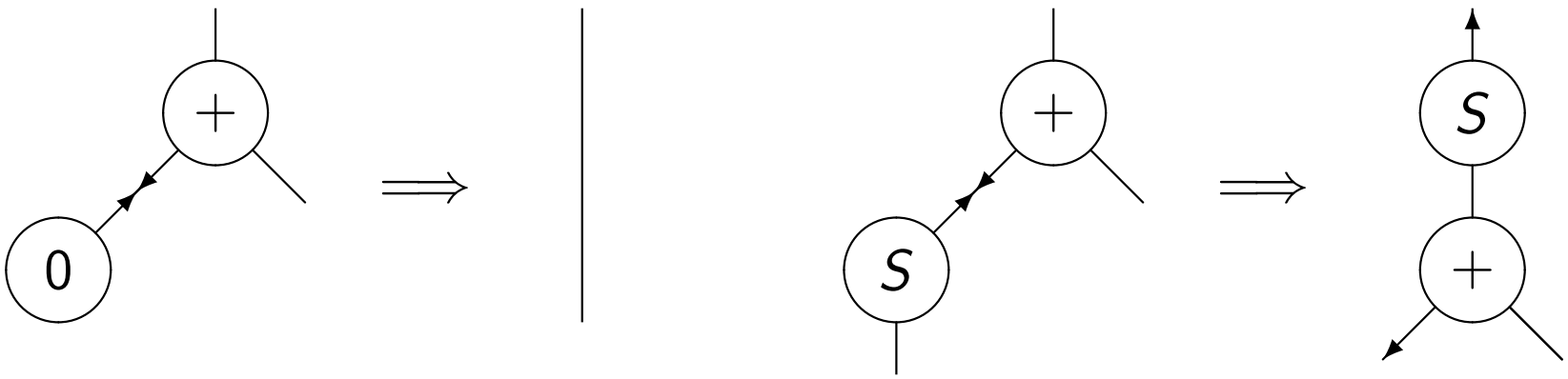


$$x = \mathsf{Add}(z, u), x = \mathcal{S}(y), y = \mathsf{Z}, u = \mathcal{S}(v), v = \mathsf{Z}$$

replacing:

$$\mathcal{S}(\mathsf{Z}) = \mathsf{Add}(z, \mathcal{S}(\mathsf{Z}))$$

Representing Rules



$$
\begin{array}{lcl}
a = \mathrm{Add}(x, y), a = \mathrm{Z} & \Longrightarrow & x = y \\
a = \mathrm{Add}(x, y), a = \mathcal{S}(z) & \Longrightarrow & x = \mathcal{S}(b), z = \mathrm{Add}(b, y)
\end{array}
$$

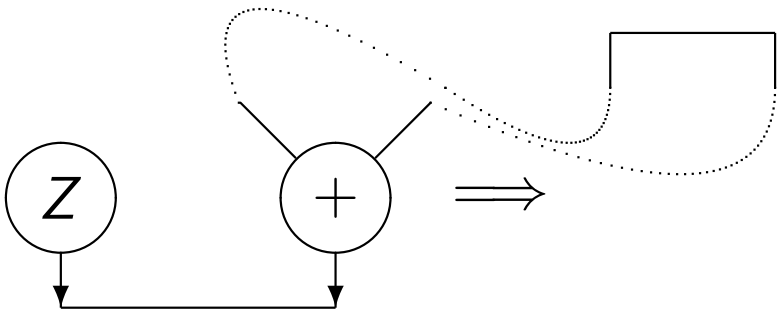
can be simplified to give:

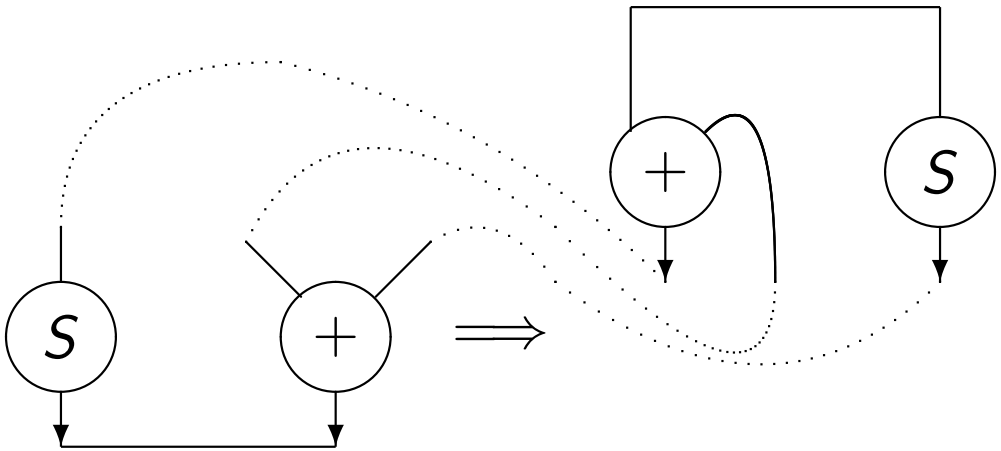
$$
\begin{array}{ll}
\mathrm{Z} = \mathrm{Add}(x, x) & \Longrightarrow \\
\mathcal{S}(\mathrm{Add}(b, y)) = \mathrm{Add}(\mathcal{S}(b), y) & \Longrightarrow
\end{array}
$$

observe that the right-hand side of the rule is always empty.

Representing Nets, diagrams

$$\begin{array}{rcl} \mathsf{Z} & >< & \mathsf{Add}(x, x) \\ \mathcal{S}(\mathsf{Add}(b, y)) & >< & \mathsf{Add}(\mathcal{S}(b), y) \end{array}$$





Configurations

A *configuration* is a pair: $c = (\mathcal{R}, \langle \mathbf{t} \mid \Delta \rangle)$

- $\mathcal{R}$ is a set of rules
- $\mathbf{t}$ a sequence $\{t_1, \ldots, t_n\}$ of terms
- Δ a multiset of equations

Each variable occurs at most twice in c. If a name occurs once in c then it is *free*, otherwise it is *bound*. For simplicity we sometimes omit $\mathcal{R}$ when there is no ambiguity. We use c, c' to range over configurations. We call $\mathbf{t}$ the *head* or *observable interface* of the configuration.

Computation Rules

The operational behaviour of the system is given by a set of four computation rules.

- Interaction: $(\alpha(t'_1, \ldots, t'_n), \beta(u'_1, \ldots, u'_m)) \in \mathcal{R} \Rightarrow$

$$\langle \mathbf{t} \mid \alpha(t_1, \ldots, t_n) = \beta(u_1, \ldots, u_m), \Gamma \rangle \longrightarrow \langle \mathbf{t} \mid t_1 = t'_1, \ldots, t_n = t'_n, u_1 = u'_1, \ldots, u_m = u'_m, \Gamma \rangle$$

- Indirection: $x \in \mathcal{N}(u) \;\Rightarrow\; \langle \mathbf{t} \mid x = t, u = v, \Gamma \rangle \longrightarrow \langle \mathbf{t} \mid u[t/x] = v, \Gamma \rangle$.
- Collect: $x \in \mathcal{N}(\mathbf{t}) \;\Rightarrow\; \langle \mathbf{t} \mid x = u, \Delta \rangle \longrightarrow \langle \mathbf{t}[u/x] \mid \Delta \rangle$.
- Multiset: $\Theta \rightleftharpoons^* \Theta', \langle \mathbf{t_1} \mid \Theta' \rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta' \rangle, \Delta' \rightleftharpoons^* \Delta \;\Rightarrow\; \langle \mathbf{t_1} \mid \Theta \rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta \rangle$.

Example: numbers

Let $\Sigma = \{Z, S, \mathrm{Add}\}$ with $ar(Z) = 0$, $ar(S) = 1$, $ar(\mathrm{Add}) = 2$, and $\mathcal{R}$:

$$\begin{array}{lcl} \mathrm{Add}(S(x), y) & \bowtie & S(\mathrm{Add}(x, y)) \\ \mathrm{Add}(x, x) & \bowtie & Z \end{array}$$

The net for 1+0 is given by the configuration $(\mathcal{R}, \langle a \mid \mathrm{Add}(a, Z) = S(Z) \rangle)$. One possible sequence of rewrites for this net is the following:

$$\begin{array}{l} \langle a \mid \mathrm{Add}(a, Z) = S(Z) \rangle \\ \quad \longrightarrow \langle a \mid a = S(x'), y' = Z, Z = \mathrm{Add}(x', y') \rangle \\ \quad \longrightarrow^* \langle S(x') \mid Z = \mathrm{Add}(x', Z) \rangle \\ \quad \longrightarrow \langle S(x') \mid x'' = x', x'' = Z \rangle \\ \quad \longrightarrow^* \langle S(Z) \mid \rangle \end{array}$$

Example: integers

Let $\Sigma = \{S, I, I^*\}$, $ar(S) = 1$, $ar(I) = ar(I^*) = 2$. Numbers are represented as a list of S agents, where I is the constructor holding a pointer to the head and tail of the list. 0 is defined by $I(x, x)$, and n by $I(S^n(x), x)$. The operation of addition can then be encoded by the configuration

$$\langle I(b, c), I^*(a, b), I^*(c, a) \mid \rangle$$

which simply appends two numbers. There is only one rule that we need: $I(a, b) \bowtie I^*(b, a)$, which is clearly a constant time operation. To show how this works, we give an example of the addition of 1+1:

$$\begin{aligned}
&\langle I(b, c) \mid I(S(x), x) = I^*(a, b), I(S(y), y) = I^*(c, a)\rangle \\
&\quad \longrightarrow^* \langle I(b, c) \mid b = S(a), a = S(c)\rangle \\
&\quad \longrightarrow^* \langle I(S(S(c)), c) \mid \rangle
\end{aligned}$$

Example: proof nets for multiplicative linear logic

$$
\begin{aligned}
&\langle c \mid \parr(a,a) = \otimes(\parr(b,b),c)\rangle \\
&\quad \longrightarrow \langle c \mid x_1 = a, y_1 = a, x_1 = \parr(b,b), y_1 = c\rangle \\
&\quad \longrightarrow^* \langle c \mid c = \parr(b,b)\rangle \\
&\quad \longrightarrow \langle \parr(b,b) \mid \rangle
\end{aligned}
$$

In fact this examples corresponds to a reduction in the λ-calculus:

$$
\begin{aligned}
(\lambda x.x)(\lambda x.x) &\rightarrow x[\lambda x.x/x] \\
&= \lambda x.x
\end{aligned}
$$

Normal forms and strategies

Interface Normal Form

$(\mathcal{R}, \langle \mathbf{t} \mid \Delta \rangle)$ is in *interface normal form* (INF) if each t_i in $\mathbf{t}$ is of one of the following forms:

- $\alpha(\vec{s})$. E.g. $\langle S(x) \mid x == Z, \Delta \rangle$.
- x where $x \in \mathcal{N}(t_j)$, $i \neq j$. This is called an *open path*. E.g. $\langle x, x \mid \Delta \rangle$.
- x where x occurs in a *cycle of principal ports* in Δ. E.g. $\langle x \mid y == \alpha(\beta(y), x), \Delta \rangle$.

Exercise: Computing interface normal form: give an algorithm

Types

Identity Group.

$$\frac{}{x\colon \sigma, x\colon \sigma^{\perp}}\ (\text{Axiom}) \qquad \frac{\Gamma, t\colon \sigma \quad \Delta, u\colon \sigma^{\perp}}{\Gamma, \Delta, t = u\colon \diamond}\ (\text{Cut})$$

Structural Group.

$$\frac{\Gamma, t\colon \sigma, u\colon \tau, \Delta}{\Gamma, u\colon \tau, t\colon \sigma, \Delta}\ (\text{Exchange}) \qquad \frac{\Gamma \quad \Delta}{\Gamma, \Delta}\ (\text{Mix})$$

User-defined Group.

$$\frac{\Gamma_1, t_1\colon \sigma_1, \ldots, t_{i_1}\colon \sigma_{i_1} \quad \ldots \quad \Gamma_k, t_k\colon \sigma_k, \ldots, t_{i_k}\colon \sigma_{i_k}}{\Gamma_1, \ldots, \Gamma_k, \alpha(t_1, \ldots, t_n)\colon \sigma}\ (\text{Graft})$$

Examples

$$\frac{\Gamma, t : \mathbf{int}^+}{\Gamma, S(t) : \mathbf{int}^+}\,(S) \qquad \frac{\Gamma}{\Gamma, Z : \mathbf{int}^+}\,(Z)$$

$$\frac{\Gamma, t_1 : \mathbf{int}^- \quad \Delta, t_2 : \mathbf{int}^+}{\Gamma, \Delta, \mathrm{Add}(t_1, t_2) : \mathbf{int}^-}\,(\mathrm{Add}) \qquad \frac{\Gamma}{\Gamma, \epsilon : \sigma}\,(\epsilon)$$

Typeable Configurations

Definition:
Let $\{x_1, \ldots, x_m\}$ be the set of free names of t, then t is a term of type σ if there is a derivation ending in $x_1 : \tau_1, \ldots, x_m : \tau_m, t : \sigma$.

Equations are typed in a similar way: if $t == u$ is an equation with free names $\{x_1, \ldots, x_m\}$, then it is typeable if there is a derivation for $x_1 : \tau_1, \ldots, x_m : \tau_m, t == u : \diamond$. This notion can be extended to multisets of equations in a straightforward way.

A configuration $\langle \mathbf{t} \mid s_1 == u_1, \ldots, s_m == u_m \rangle$ with free names $x_1, \ldots, x_p$ is typeable by $\sigma_1, \ldots, \sigma_n$, if there is a derivation for $x_1 : \rho_1, \ldots, x_p : \rho_p, t_1 : \sigma_1, \ldots, t_n : \sigma_n, s_1 == u_1 : \diamond, \ldots, s_m == u_m : \diamond$.

Example:

Type derivation for $\langle a \mid \mathrm{Add}(a, Z) = S(Z) \rangle$

$$\dfrac{\dfrac{\dfrac{}{a\colon \mathbf{int}^+, a\colon \mathbf{int}^-}\,(\mathrm{Axiom}) \qquad \dfrac{}{Z\colon \mathbf{int}^+}\,(Z)}{a\colon \mathbf{int}^+, \mathrm{Add}(a, Z)\colon \mathbf{int}^-}\,(\mathrm{Add}) \qquad \dfrac{\dfrac{}{Z\colon \mathbf{int}^+}\,(Z)}{S(Z)\colon \mathbf{int}^+}\,(S)}{a\colon \mathbf{int}^+, \mathrm{Add}(a, Z) = S(Z)\colon \diamond}\,(\mathrm{Cut})$$

Typeable rules

Definition: $\alpha(t_1, \ldots, t_n) \bowtie \beta(s_1, \ldots, s_m)$ is typeable if

1. there is a derivation D with conclusion $\alpha(x_1, \ldots, x_n) = \beta(y_1, \ldots, y_m) : \diamond$ and leaves containing assumptions for $x_1, \ldots, x_n, y_1, \ldots, y_m$, built by application of the Cut rule and the Graft rules for α, β,
2. there is a type derivation with the same assumptions leading to the conclusion $x_1 = t_1 : \diamond, \ldots, x_n = t_n : \diamond, y_1 = s_1 : \diamond, \ldots, y_m = s_m : \diamond$,
3. and whenever an equation $\alpha(t'_1, \ldots, t'_n) = \beta(s'_1, \ldots, s'_m)$ is typeable, its type derivation is obtained by using the Cut rule and instances of the Graft rules for α, β applied in D.

Type derivation for $\mathrm{Add}(S(x), y) \bowtie S(\mathrm{Add}(x, y))$

Build the most general derivation for the active pair

$$\dfrac{\dfrac{x_1 : \mathbf{int}^- \quad x_2 : \mathbf{int}^+}{\mathrm{Add}(x_1, x_2) : \mathbf{int}^-}\,(\mathrm{Add}) \qquad \dfrac{x_3 : \mathbf{int}^+}{S(x_3) : \mathbf{int}^+}\,(S)}{\mathrm{Add}(x_1, x_2) = S(x_3) : \diamond}\,(\mathrm{Cut})$$

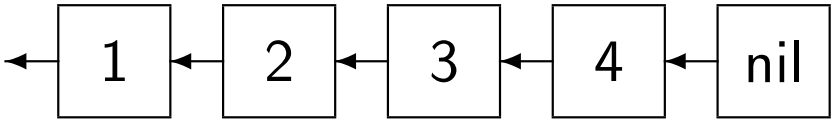
Then we use this template to build a second derivation:

$$\dfrac{\dfrac{x_1 : \mathbf{int}^- \qquad \dfrac{x_2 : \mathbf{int}^+ \qquad \dfrac{x_3 : \mathbf{int}^+ \qquad \dfrac{\dfrac{\dfrac{}{x : \mathbf{int}^-, x : \mathbf{int}^+}\,(\mathrm{Axiom})}{x : \mathbf{int}^-, S(x) : \mathbf{int}^+}\,(S) \qquad \dfrac{}{y : \mathbf{int}^-, y : \mathbf{int}^+}\,(\mathrm{Axio}}{y : \mathbf{int}^-, S(x) : \mathbf{int}^+, \mathrm{Add}(x, y) : \mathbf{int}^-}\,(\mathrm{Add})}{y : \mathbf{int}^-, S(x) : \mathbf{int}^+, x_3 = \mathrm{Add}(x, y) : \diamond}\,(\mathrm{Cut})}{S(x) : \mathbf{int}^+, x_2 = y : \diamond, x_3 = \mathrm{Add}(x, y) : \diamond}\,(\mathrm{Cut})}{x_1 = S(x) : \diamond, x_2 = y : \diamond, x_3 = \mathrm{Add}(x, y) : \diamond}\,(\mathrm{Cut})$$

Applied nets

- So far we have looked at pure nets (cf. pure λ-calculus)
- Adding simple data types can be done in an obvious way, to preserve the main properties
- One can then build a variety interesting data types, for instance lists

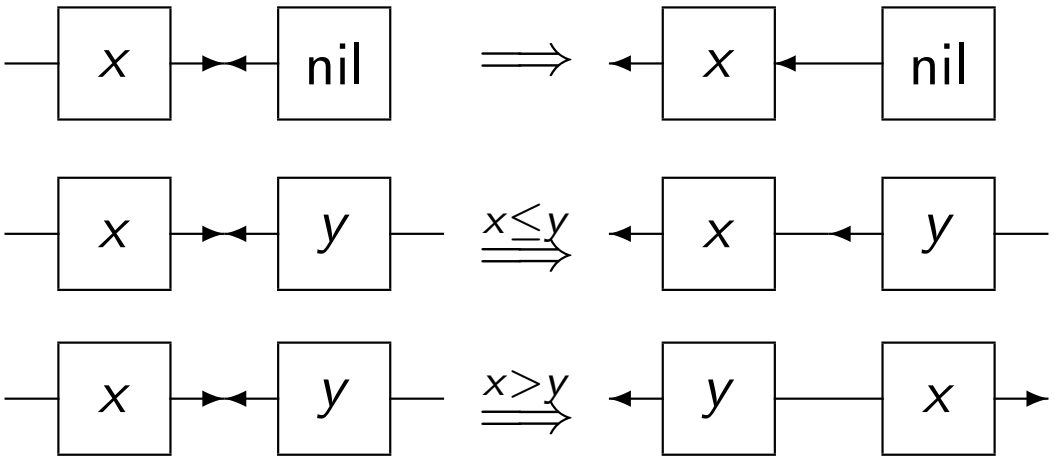
To simplify the diagrams, we often just write the contents of the node and omit the name when no confusion will arise. For example, here is a list of 4 elements:



Exercise: What about the pointers?

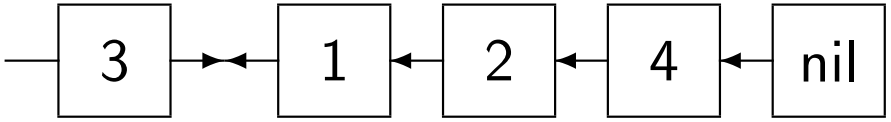
Example: Insertion

Here are the rules to insert an element into a sorted list:



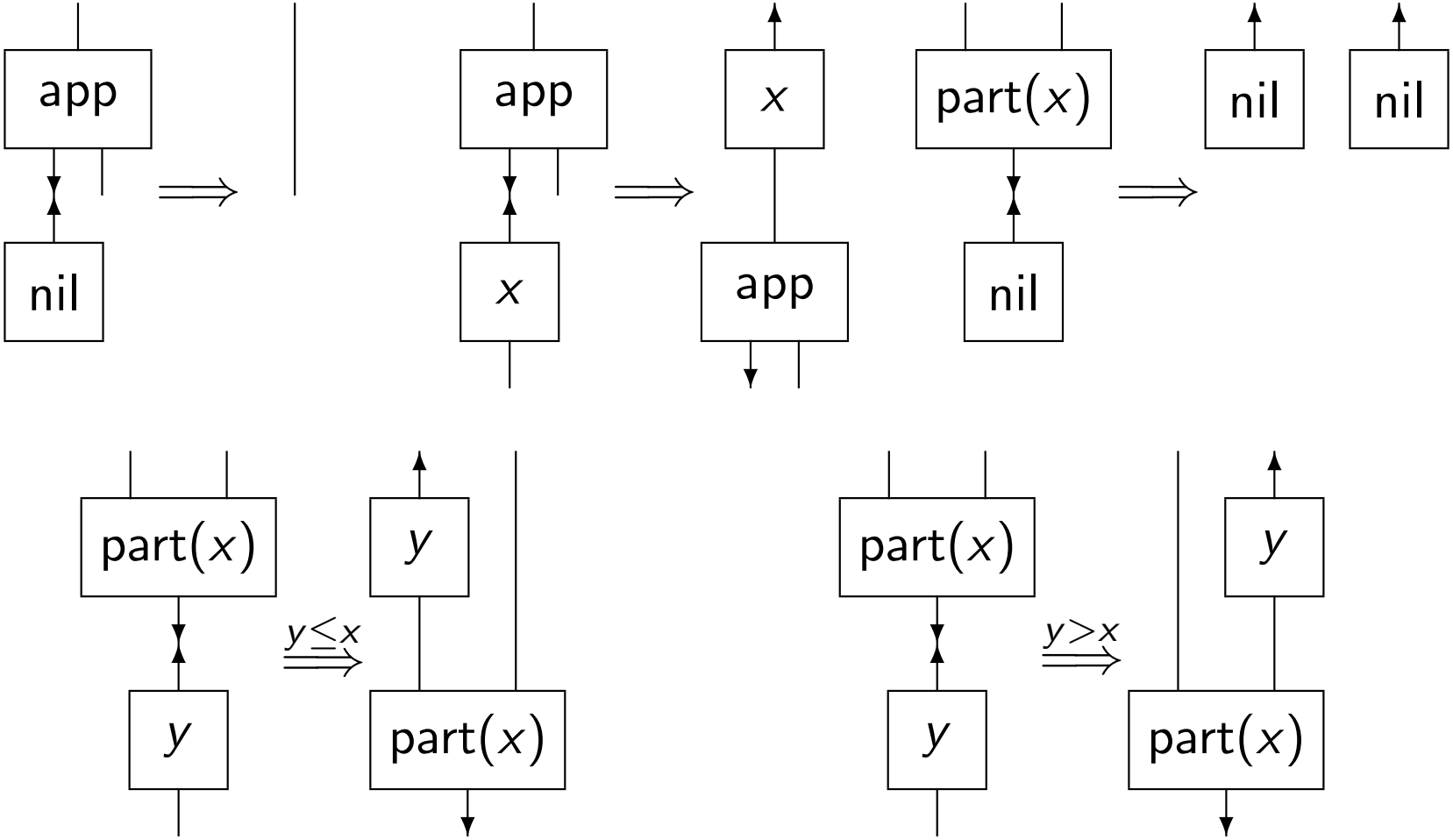
Exercises:

- Try this example:

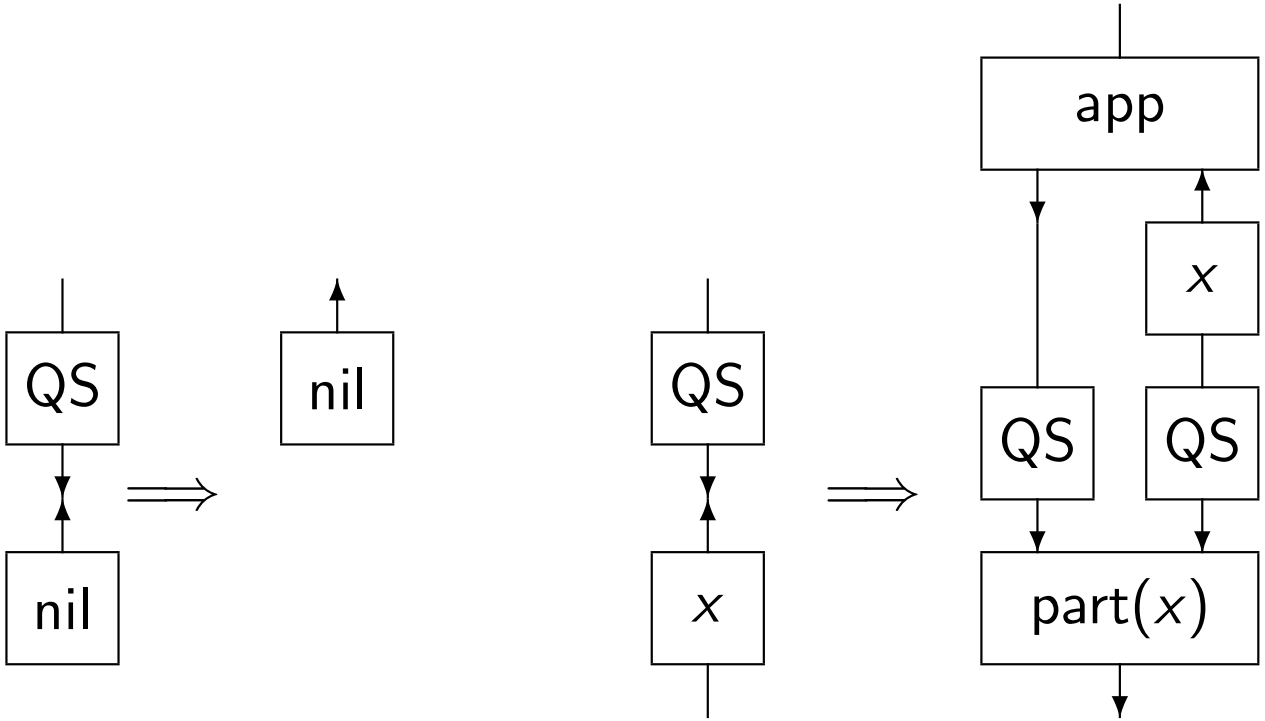


- Define insertion sort
- Define bubble sort, and compare with insertion sort

Examples: Append and Partition



Example: Quicksort



Compare this with a functional version:

```
QS nil = nil
QS (x:t) = let (a,b) = part x t in app(QS(a),x:QS(b))
```

which uses additional datatypes

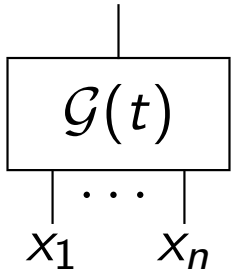
Main points of these examples

1. The graphical representation of the problem is directly cast into the graphical language. The algorithm given can be understood as programming directly with the internal data structures, rather than some syntax describing it.
2. All rewrite steps correspond to steps in the computation: there is no need to introduce additional data structures and operations that are not part of the problem.
3. Because of point 1 above, if we single-step the computation we get an animation of the algorithm directly from the rewriting system.

Applications

- One of the main applications of interaction nets is the implementation of the λ-calculus.
- Sharing: cannot duplicate an active pair
- Here we give a hint of this work, and show the linear λ-calculus

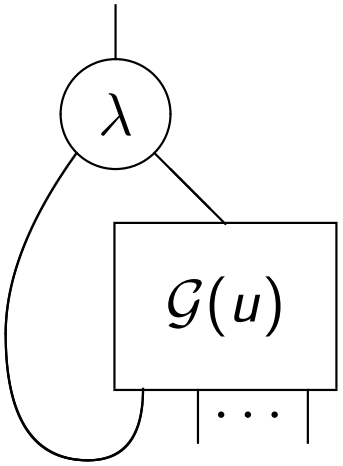
A λ-term t with $fv(t) = \{x_1, \ldots, x_n\}$ will be translated as a net $\mathcal{G}(t)$ with the root edge at the top, and n free edges corresponding to the free variables, which we draw as:



Variables and Abstraction

If t is a variable, say x, then $\mathcal{G}(t)$ is given by an edge:

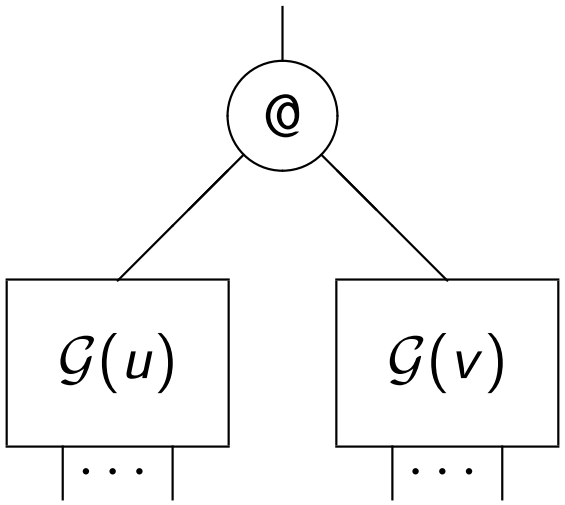
If t is an abstraction, say $\lambda x.u$, then $\mathcal{G}(\lambda x.u)$ is given by



where we have assumed (without loss of generality) that the (unique occurrence of the) variable x occurs in the leftmost position of the free variables of $\mathcal{G}(u)$.

Application

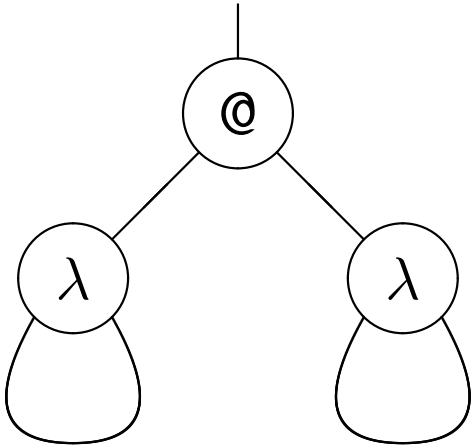
If t is an application, say uv, then $\mathcal{G}(uv)$ is given by:

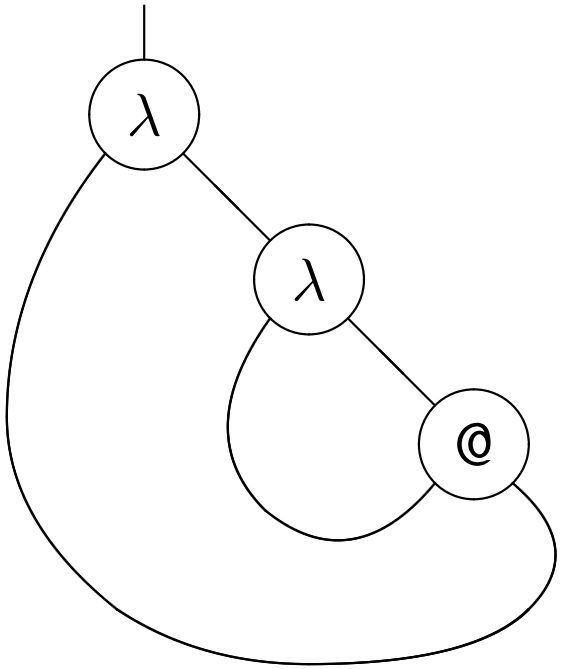


Remark that there will not be any common free variables between the graphs for u and v by the linearity constraint.

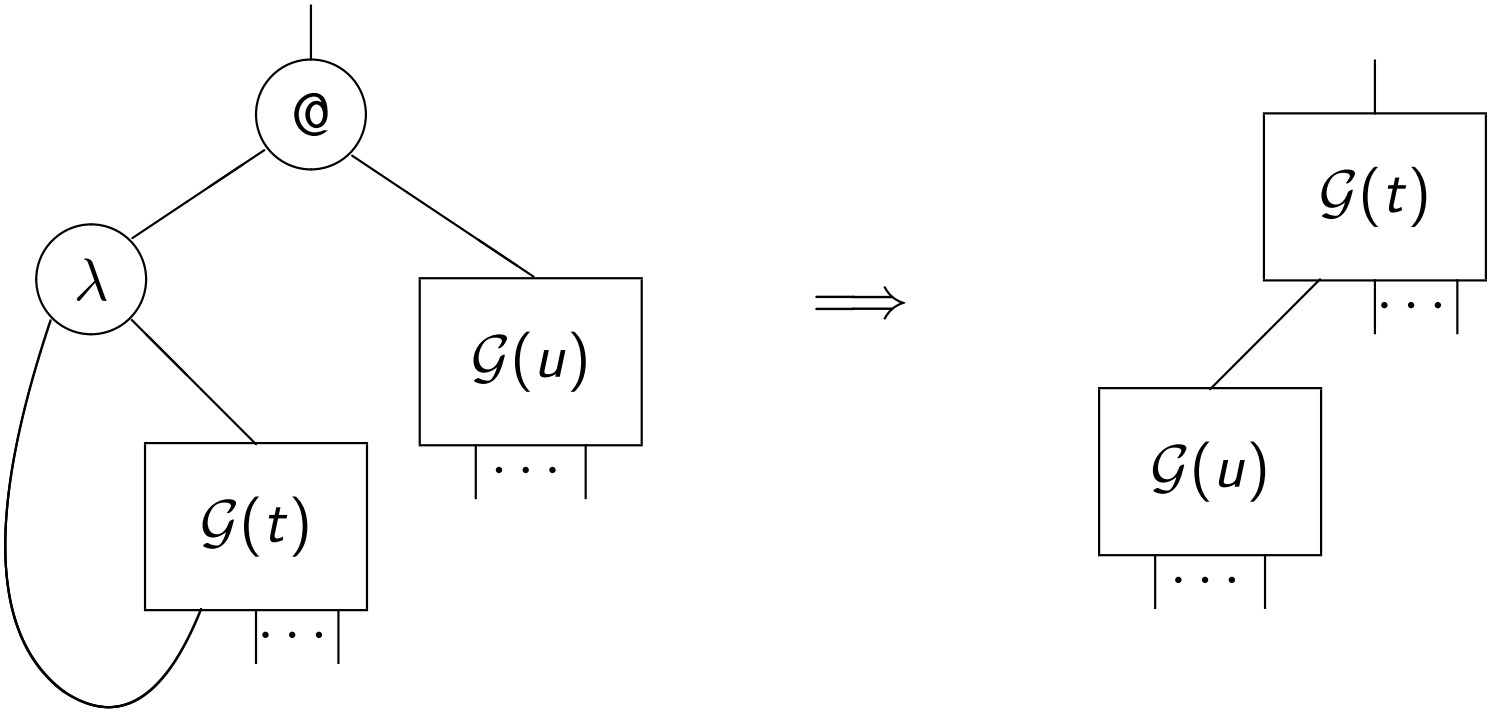
Exercise: Put in the principal ports for the agents @ and λ

Example nets

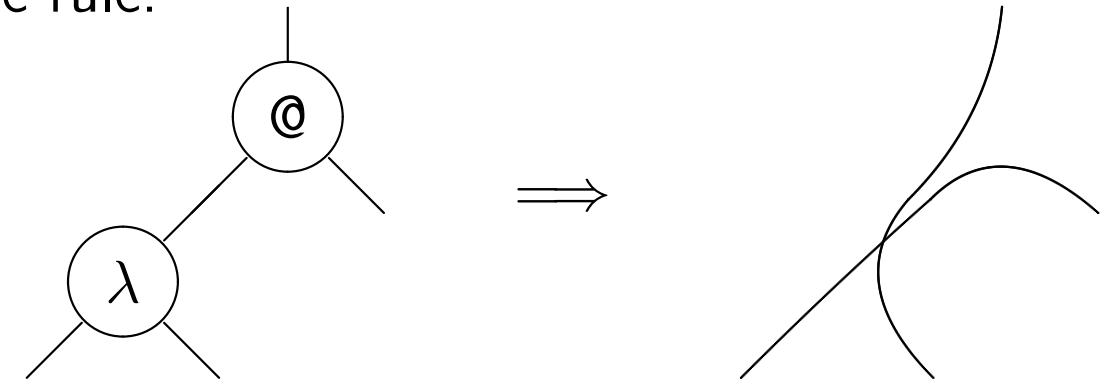




Beta reduction



Isolating the rule:



Exercise

- Build the net for $\mathcal{G}(()(\lambda xy.yx)(\lambda x.x)(\lambda x.x))$
- Reduce the net to normal form, and compare with the following:

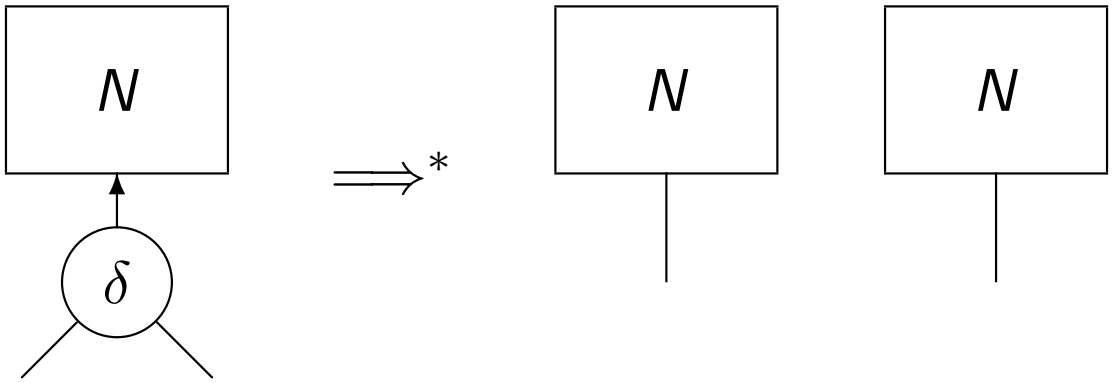
$$
\begin{array}{rcl}
(\lambda xy.yx)(\lambda x.x)(\lambda x.x) & \to_\beta & ((\lambda y.yx)[(\lambda x.x)/x])(\lambda x.x) \\
& = & (\lambda y.((yx)[(\lambda x.x)/x]))(\lambda x.x) \\
& = & (\lambda y.y(x[(\lambda x.x)/x]))(\lambda x.x) \\
& = & (\lambda y.y(\lambda x.x))(\lambda x.x) \\
& \to_\beta & (y(\lambda x.x))[(\lambda x.x)/y] \\
& = & (y[(\lambda x.x)/y])(\lambda x.x) \\
& = & (\lambda x.x)(\lambda x.x) \\
& \to_\beta & x[(\lambda x.x)/x] \\
& = & \lambda x.x
\end{array}
$$

what do you notice about the way substitution is implemented?

There are many works on implementing the full λ-calculus

Duplicating Nets

- Top feature of interactions nets: it's impossible to duplicate an active pair.
- However, to encode a specific reduction strategy we may need to duplicate a redex.



1. N must be free from δ agents
2. N must be in normal form
3. N must be deadlock-free

(Similar conditions for erasing nets)

Satisfying the Conditions

- For implementations of the λ-calculus, the presence of δ agents is a real problem.
- Several solutions have been proposed:
  - indexing: Lamping's algorithm
  - synchronizing boundary: Abramsky
  - extraction of δs: Lafont
- Each of these techniques is worthy of study: here we just look at the last one

Active Pair (redex) Extraction

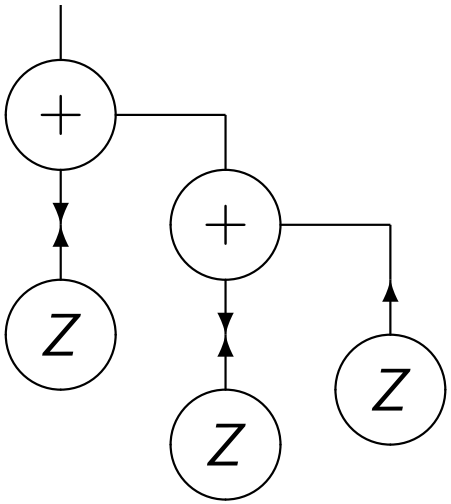
Generalization of Lafont's δ extraction:

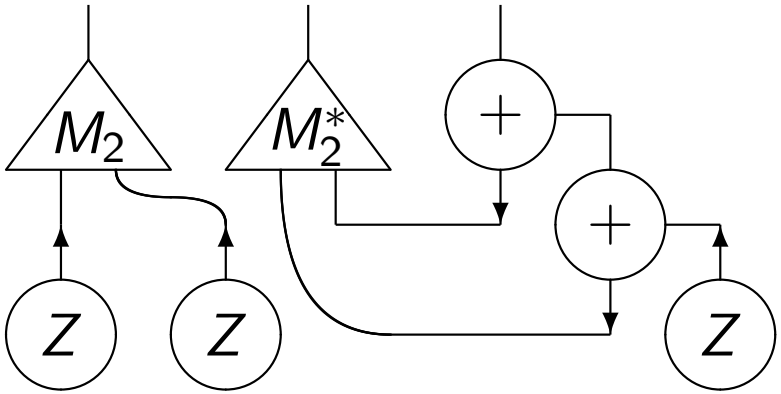
1. Extract edges corresponding to active pairs
2. Collect the free edges together with multiplexing nets.
3. The resulting net is a net in *normal form* and can be duplicated.
4. The active pairs can be recovered by a simple decoding net.

Using this idea, we can "freeze" nets, and when used in a controlled way, can be used to give different reduction strategies.

Example

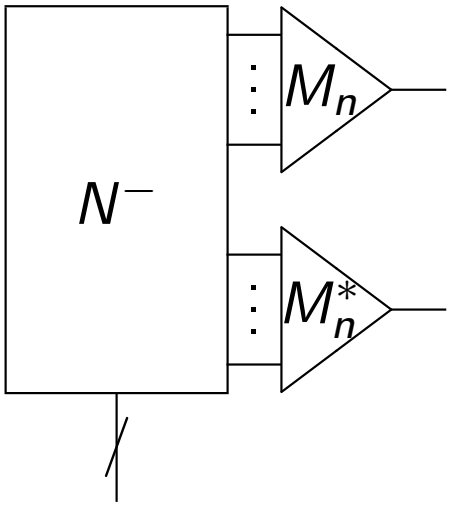
Extract active pairs





Exercise: Define, using γ from the interaction combinators, the multiplexing nets M_2 and M_2^*

General form of edge extraction



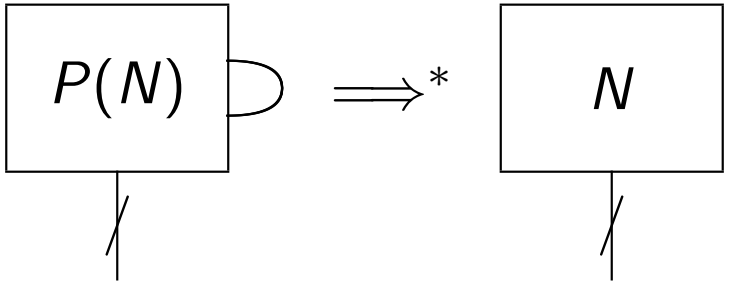
The resulting system has agents in $\Sigma \cup \{\gamma\}$.

Examples of use:

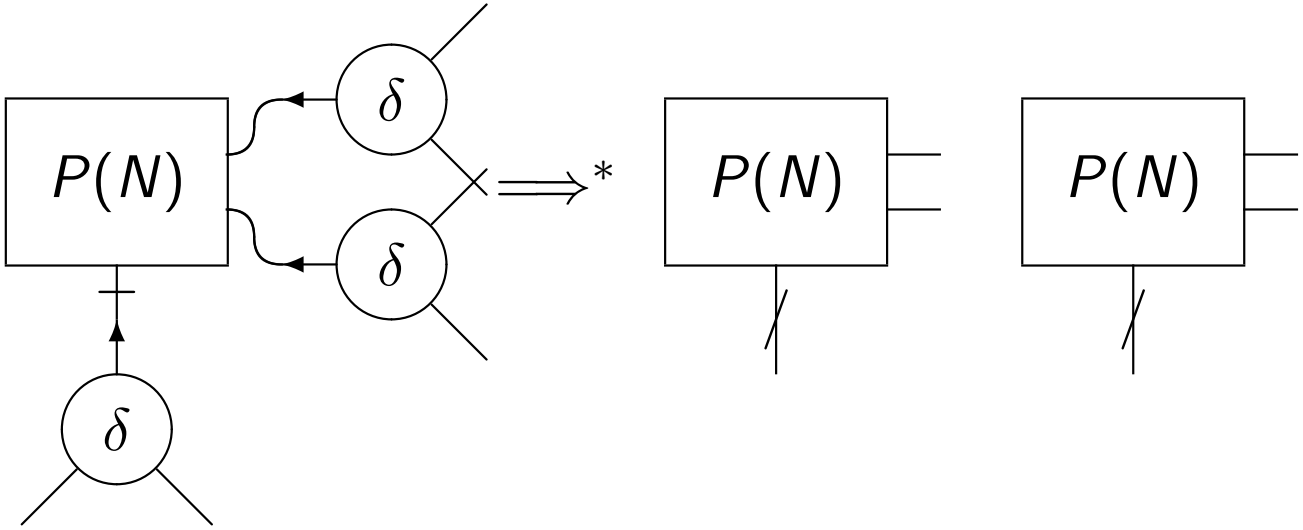
- Active pairs
- Cycles
- etc.

Properties

Unpacking



Duplication (also erasing)



Summary of λ-calculus

- Many different implementations of the λ-calculus, usually giving a new strategy
- Using known techniques for interaction nets, we can recover extant strategies:

	WHNF	*HNF*
Call-by-value	$!(A \multimap B)$	-
Call-by-name	$!(!A \multimap B)$	$!A \multimap B$

as well as several others.

These are generally significantly more efficient than current implementations of functional languages.

Conclusions

- Only touched on a few topics in interaction nets
- Current research:
  - programming languages built around interaction nets
  - tools for visual programming
  - compiling a variety of paradigms to interaction nets
  - etc.
- Many additional resources on the web page, including an implementation