Interaction Nets vs. the ρ-calculus: Introducing Bigraphical Nets

Maribel Fernández[1] Ian Mackie[1] François-Régis Sinot[2]

[1]DCS, King's College London

[2]LIX, École Polytechnique

Rho-Calculus Workshop, May 2005

Motivations

- Interaction Nets (IN) [Lafont90] are graph rewriting systems with a good level of sharing: redexes cannot be copied! They have been used to implement efficient strategies in the λ-calculus (e.g. optimal reduction [Lamping 90, Gonthier,Abadi and Levy92], YALE[Mackie98], BOHM [Asperti et al.00]).

Motivations

- Interaction Nets (IN) [Lafont90] are graph rewriting systems with a good level of sharing: redexes cannot be copied! They have been used to implement efficient strategies in the λ-calculus (e.g. optimal reduction [Lamping 90, Gonthier,Abadi and Levy92], YALE[Mackie98], BOHM [Asperti et al.00]).
- Extensions of IN have also been used to implement process calculi, object calculi, and term rewriting systems.

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:
  - Can an interaction net implementation of the λ-calculus be (easily?) adapted to implement the rewriting calculus?

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:
  - Can an interaction net implementation of the λ-calculus be (easily?) adapted to implement the rewriting calculus?
  - Will an efficient implementation of the λ-calculus produce an efficient implementation of the rewriting calculus?

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:
  - Can an interaction net implementation of the λ-calculus be (easily?) adapted to implement the rewriting calculus?
  - Will an efficient implementation of the λ-calculus produce an efficient implementation of the rewriting calculus?
  - Will a graphical encoding of the calculus bring more understanding to the theory?

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:
  - Can an interaction net implementation of the λ-calculus be (easily?) adapted to implement the rewriting calculus?
  - Will an efficient implementation of the λ-calculus produce an efficient implementation of the rewriting calculus?
  - Will a graphical encoding of the calculus bring more understanding to the theory?

Motivations

- The rewriting calculus [Cirstea and Kirchner00] generalises the λ-calculus by introducing patterns and a customisable matching theory.
- The obvious questions are:
  - Can an interaction net implementation of the λ-calculus be (easily?) adapted to implement the rewriting calculus?
  - Will an efficient implementation of the λ-calculus produce an efficient implementation of the rewriting calculus?
  - Will a graphical encoding of the calculus bring more understanding to the theory?

- We will address some of these questions in this talk.

Background

- We will consider a set $\mathcal{T}$ of ρ-terms generated by:

$$t, u \ ::= x \mid f \mid p \twoheadrightarrow t \mid [p \ll u].t \mid (t \ u)$$

$$p ::= x \mid f \ p_1 \ldots p_n$$

where p is a *linear* algebraic pattern (i.e. each variable occurs at most once in p), $p \twoheadrightarrow t$ is a *generalised abstraction* where the variables in p are boudn in t, and $[p \ll u].t$ is a *delayed matching constraint*. As usual terms are defined modulo α-conversion.

Background

- We will consider a set $\mathcal{T}$ of ρ-terms generated by:

$$t, u ::= x \mid f \mid p \twoheadrightarrow t \mid [p \ll u].t \mid (t\ u)$$

$$p ::= x \mid f\ p_1 \ldots p_n$$

  where p is a *linear* algebraic pattern (i.e. each variable occurs at most once in p), $p \twoheadrightarrow t$ is a *generalised abstraction* where the variables in p are boudn in t, and $[p \ll u].t$ is a *delayed matching constraint*. As usual terms are defined modulo α-conversion.

- The reduction rules are the following:

$$\begin{array}{lrcl} (\rho) & (p \twoheadrightarrow t)\ u & \to & [p \ll u].t \\ (\sigma) & [p \ll u].t & \to & \sigma_{p \ll u}(t) \end{array}$$

  and as usual the reduction relation is the contextual closure of the rules.

Isolating the problem of matching

$\Rightarrow$ Divide-and-conquer: We will focus on the implementation of the σ rule, isolating as much as possible the problem of matching from the well-known problems of implementing binders (the λ-calculus).

Isolating the problem of matching

- Divide-and-conquer: We will focus on the implementation of the σ rule, isolating as much as possible the problem of matching from the well-known problems of implementing binders (the λ-calculus).

$\Rightarrow$ The rule (ν) defined by:

$$(\nu) \qquad p \rightarrowtriangle t \quad \rightarrow \quad x \rightarrowtriangle [p \ll x].t \qquad p \text{ not a variable, } x \text{ fresh}$$

is derivable: $p \rightarrowtriangle t \leftarrow_\eta x \rightarrowtriangle (p \rightarrowtriangle t)\ x \rightarrow_\rho x \rightarrowtriangle [p \ll x].t$
ν is normalising and ν-normal forms have only variables as patterns in abstractions (i.e. we are in a λ-calculus with a match construct).

Implementing the σ rule

σ asks for an *external* matching algorithm to find the solutions (if any) of the matching of p with u, and applies the corresponding substitution to t.
We will give a full implementation of the calculus, including the external matching algorithm.
First we define a version of the ρ-calculus with explicit matching inspired by [Cirstea,Faure,Kirchner04].
Substitution will still be implicit since it will be realised for free in the graphical representation.

$$
\begin{array}{lrcl}
(\rho) & (p \twoheadrightarrow t)\ u & \longrightarrow & [p \ll u].t \\
(\sigma_v) & [x \ll u].t & \longrightarrow & t\{x = u\} \\
(\sigma_{a_n}) & [f\ p_1 \ldots p_n \ll f\ u_1 \ldots u_n].t & \longrightarrow & [p_1 \ll u_1] \ldots [p_n \ll u_n].t
\end{array}
$$

Decomposing σ_{a_n}

We replace σ_{a_n} by a finite set of local rules:

$$
\begin{array}{lrcl}
(a_c) & f\ t & \longrightarrow & f \bullet t \\
(a_a) & (t \bullet u)\ v & \longrightarrow & (t \bullet u) \bullet v \\
(\sigma_c) & [f \ll f].t & \longrightarrow & t \\
(\sigma_a) & [(p \bullet r) \ll (u \bullet v)].t & \longrightarrow & [p \ll u].[r \ll v].t
\end{array}
$$

Blocked Matching, Semantics

- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.

Blocked Matching, Semantics

- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.
- We can add a rule to detect failure:
  $(\bot) \qquad [f \ll g].t \to \bot \qquad \text{if } f \neq g$

Blocked Matching, Semantics

- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.
- We can add a rule to detect failure:
  $(\bot) \qquad [f \ll g].t \to \bot \qquad \text{if } f \neq g$
- We can propagate it in two ways:

Blocked Matching, Semantics

- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.
- We can add a rule to detect failure:
  $(\bot) \qquad [f \ll g].t \to \bot \qquad \text{if } f \neq g$
- We can propagate it in two ways:
  - (*strict*) $\qquad C[\bot] \to \bot \qquad$ for any $C[\cdot]$
    corresponds to an exception-like semantics of matching failure.

Blocked Matching, Semantics

- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.
- We can add a rule to detect failure:
  $(\bot) \qquad [f \ll g].t \to \bot \qquad \text{if } f \neq g$
- We can propagate it in two ways:
  - (*strict*) $\qquad C[\bot] \to \bot \qquad$ for any $C[\cdot]$
    corresponds to an exception-like semantics of matching failure.
  - (*non-strict*) $\qquad C[\bot] \to \bot \qquad$ if $C[\cdot] \in \mathcal{C}$.
    Here we consider $\mathcal{C} = \{([\,]\ t), t \in \mathcal{T}\}$.

Blocked Matching, Semantics

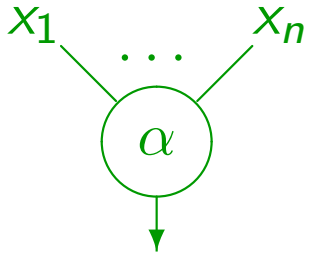
- If $[p \ll u]$ has no solution then $[p \ll u].t$ is a normal form if p, u, t are. This is called a *blocked matching*.
- We can add a rule to detect failure:
  $(\bot) \qquad [f \ \ll g].t \to \bot \qquad \text{if } f \neq g$
- We can propagate it in two ways:
  - (*strict*) $\qquad C[\bot] \to \bot \qquad$ for any $C[\cdot]$
    corresponds to an exception-like semantics of matching failure.
  - (*non-strict*) $\qquad C[\bot] \to \bot \qquad$ if $C[\cdot] \in \mathcal{C}$.
    Here we consider $\mathcal{C} = \{([\,]\ t), t \in \mathcal{T}\}$.
  - Rules of the form $[p \ll u].t \to \bot$ if p and u have different head symbols, are unsafe and lead to non-confluence. Our IN encodings cannot implement non-confluent (unsafe) rules (IN are strongly confluent).

Interaction Nets

- Agents have one principal port each, depicted by an arrow, and a finite number of auxiliary ports:

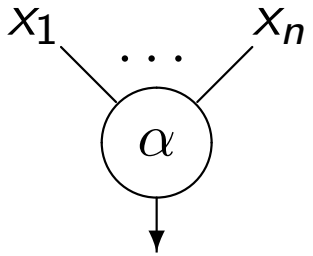
Interaction Nets

- Agents have one principal port each, depicted by an arrow, and a finite number of auxiliary ports:



Interaction Nets

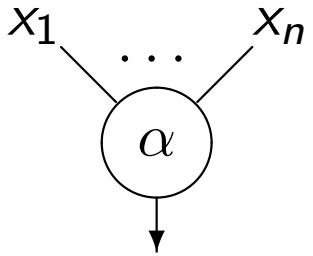
- Agents have one principal port each, depicted by an arrow, and a finite number of auxiliary ports:



- A net N is a graph (not necessarily connected) with agents at the vertices and edges connecting agents together at the ports, such that there is only one edge at every port.

Interaction Nets

- Agents have one principal port each, depicted by an arrow, and a finite number of auxiliary ports:



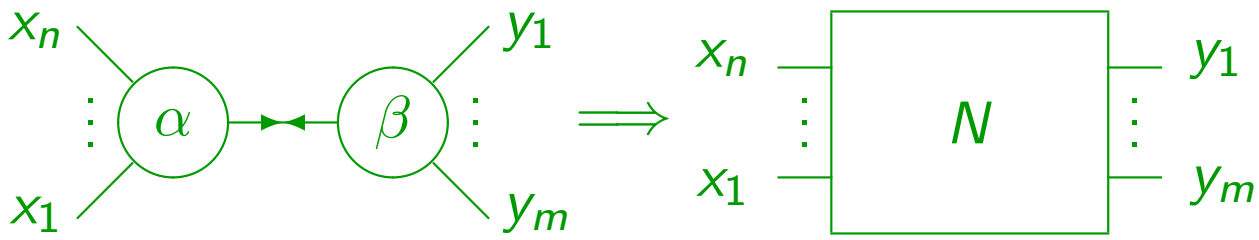
- A net N is a graph (not necessarily connected) with agents at the vertices and edges connecting agents together at the ports, such that there is only one edge at every port.
- There are two special instances of a net: a wiring (no agents) and the empty net; the extremes of wirings are also called free ports. The interface of a net is its set of free ports.

Interaction Rules

- An interaction rule $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces a pair of agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected together on their principal ports (this is called an *active pair*) by a net N with the same interface.

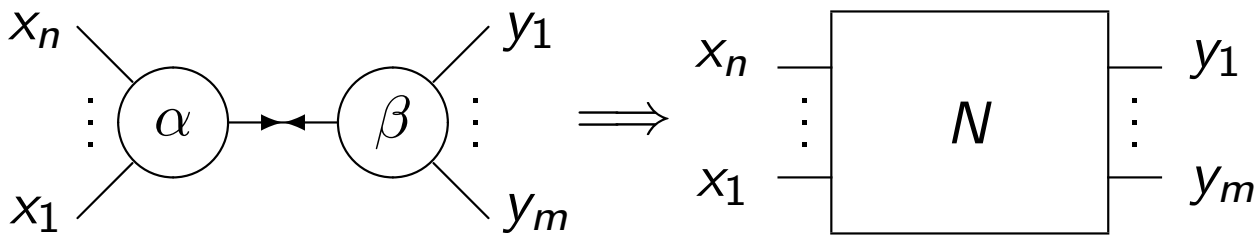
Interaction Rules

- An interaction rule $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces a pair of agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected together on their principal ports (this is called an *active pair*) by a net N with the same interface.



Interaction Rules

- An interaction rule $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces a pair of agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected together on their principal ports (this is called an *active pair*) by a net N with the same interface.



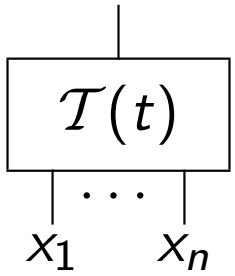
- Reduction is local, and moreover there can be at most one rule for each pair of agents.

IN Encoding of the Rewriting Calculus

We will derive an encoding of the rho-calculus from an off-the-shelf IN encoding of the λ-calculus by adding the matching rules.
The first is a simple encoding implementing the strict semantics.

A Simple Encoding

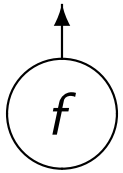
We define by induction a translation $\mathcal{T}(\cdot)$ of ρ-terms.
For a ρ-term t with free variables $\mathrm{fv}(t) = \{x_1, \ldots, x_n\}$, $\mathcal{T}(t)$ will look like:



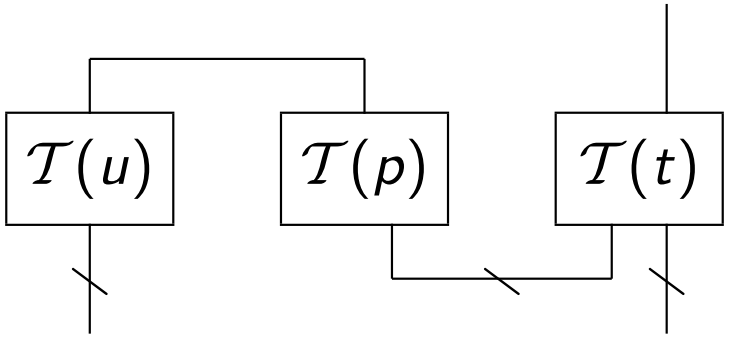
A Simple Encoding

A variable is represented as a wire:

For each constant f we introduce an agent:



$[p \ll u].t$ is encoded as:



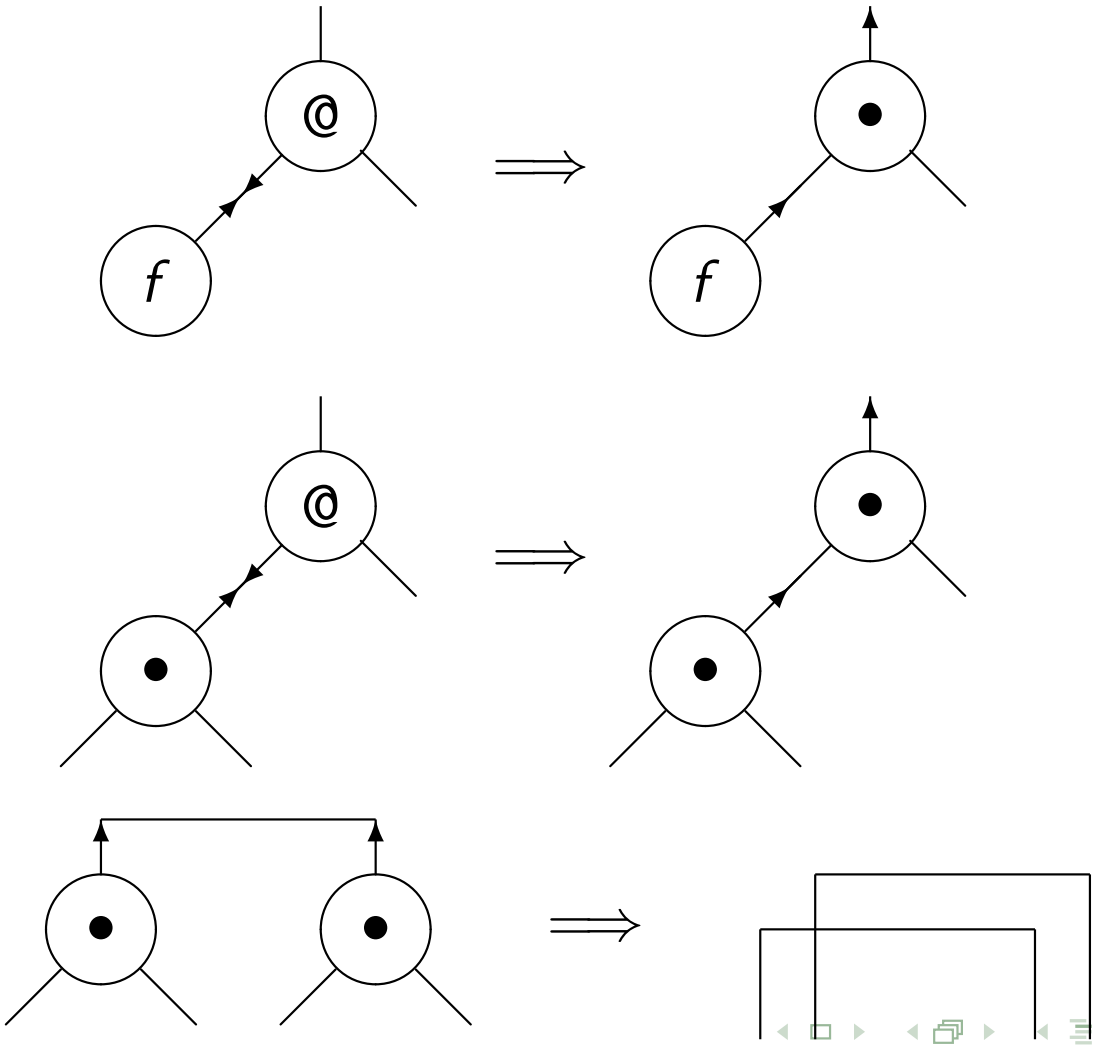
For abstractions and applications we will assume that we are working with ν-normal forms, hence we can reuse the λ-calculus encodings, also for the ρ (i.e. β) rule.

Interaction Rules for Matching

The matching algorithm is initiated by connecting the root of a pattern with the term to match.
(σ_v) (matching against a variable) is realised for free, as in the λ-calculus.
To simulate (σ_a) (and $(\perp)$), constants will interact.

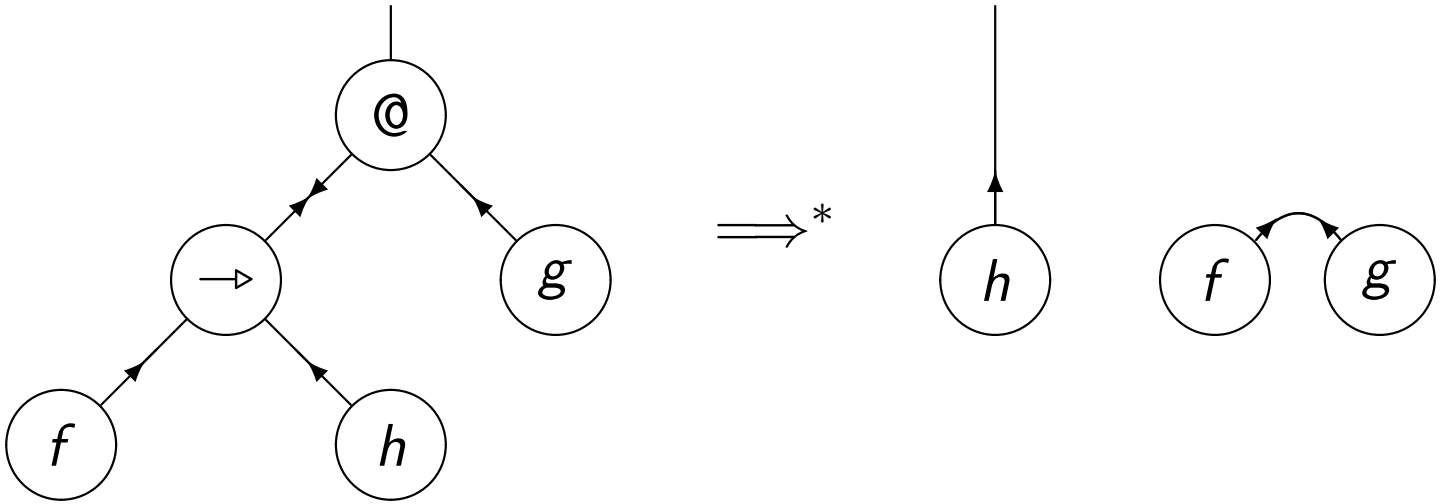
More Interaction Rules for Matching

Since in interaction nets every agent only has one principal port, we need two different application agents, with rules to convert @ into • and matching rules for •.



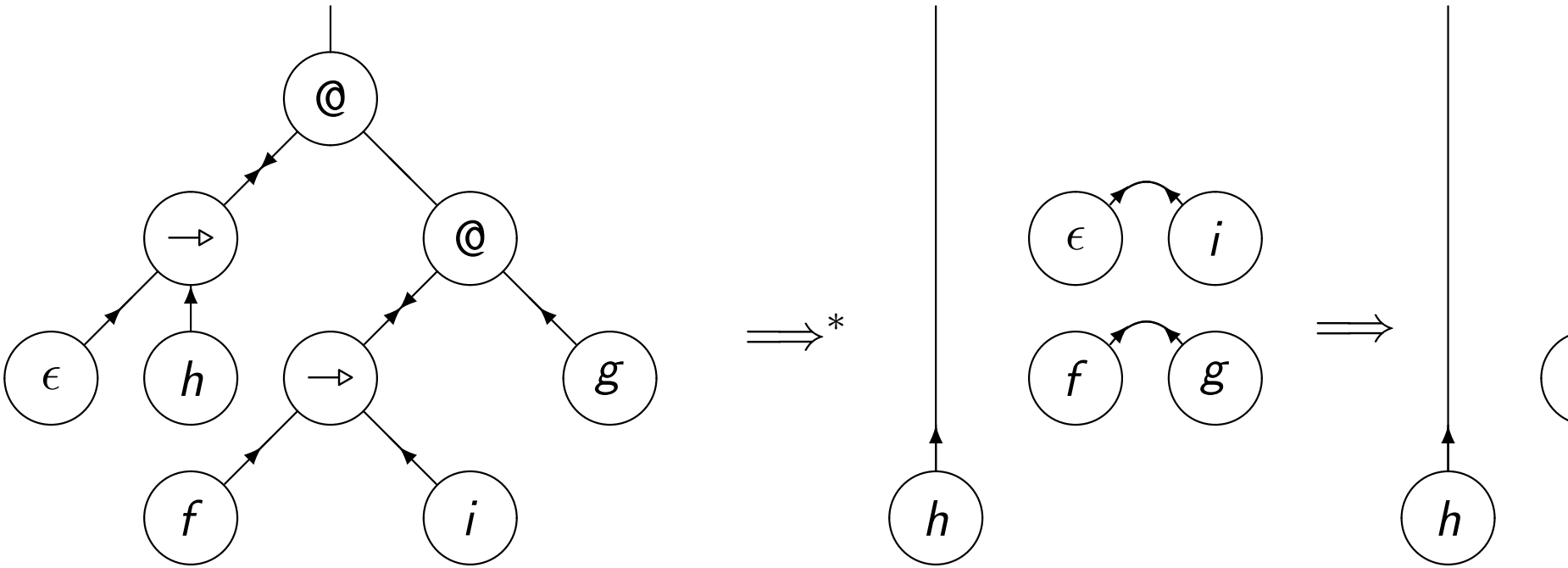
Example

$(f \rightarrowtriangle h)\ g$ evaluates to a blocked matching (failure) $[f \ll g].h$



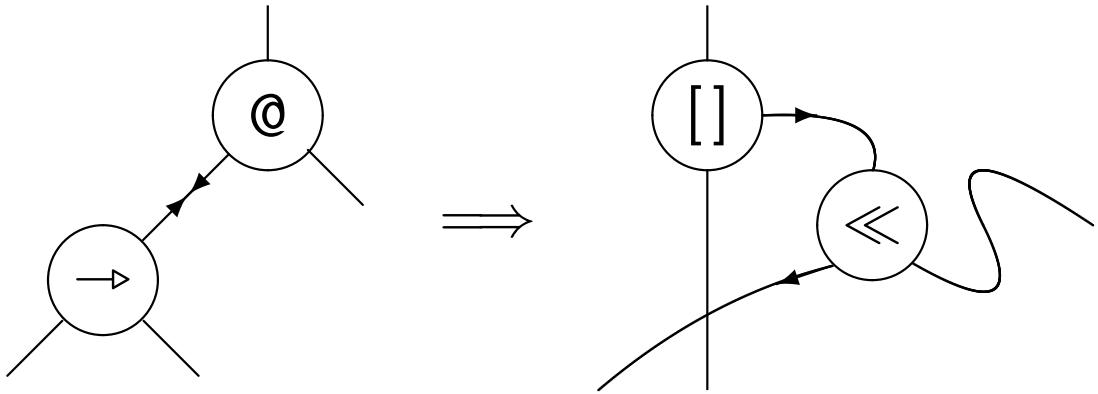
Properties

This encoding implements a strict semantics. A net containing a fail agent is interpreted as failure.
Note that we cannot get a non-strict semantics with this encoding!
Reduction of $(x \rightarrowtriangle h)((f \rightarrowtriangle i)\ g)$ should produce h but we get the same net as above:



Introducing a Matching Agent

To implement a non-strict semantics, the obvious solution is to keep track of the point where a matching failure may have occurred by using an explicit agent $\ll$ for matching.



The agent [] will wait until the matching returns a result. Success will be represented by an agent $\top$, failure by an agent $\bot$, both 0-ary, with the expected interactions with agent [].
If the matching cannot be solved in either way, this agent will stay there forever.

Properties

- Correctness and Completeness: If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$, then $t \rightarrow^* u$. If t is closed and $t \rightarrow^* u$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$.

Properties

- Correctness and Completeness: If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$, then $t \rightarrow^* u$. If t is closed and $t \rightarrow^* u$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$.
- The agent [] blocks any further computation between the root and what was the body of the abstraction.
  For instance, in $(a \rightarrowtriangle b)\ ((p \rightarrowtriangle c)\ t)$ where a, b, c are constants and p, t are terms, we have to complete the matching of t against p before noticing that a and c do not match. This was not the case in the first encoding.

Properties

- Correctness and Completeness: If $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$, then $t \rightarrow^* u$. If t is closed and $t \rightarrow^* u$ in normal form, then $\mathcal{T}(t) \Longrightarrow^* \mathcal{T}(u)$.
- The agent [] blocks any further computation between the root and what was the body of the abstraction.
  For instance, in $(a \rightarrowtriangle b)\ ((p \rightarrowtriangle c)\ t)$ where a, b, c are constants and p, t are terms, we have to complete the matching of t against p before noticing that a and c do not match. This was not the case in the first encoding.
- The drawback of the first encoding is not about connectivity, but about locality!

Bigraphs

- Bigraphs [Jensen, Milner03] are nested graphs representing two kinds of structure: locality (nodes may occur inside other nodes) and connectivity (nodes have ports that may be connected by links)

Bigraphs

- Bigraphs [Jensen, Milner03] are nested graphs representing two kinds of structure: locality (nodes may occur inside other nodes) and connectivity (nodes have ports that may be connected by links)
- Links are attached to nodes from the inside or the outside, so bigraphs have both an inner and an outer interface. A control (or agent) is *atomic* if it cannot contain a nested graph, otherwise it is non-atomic.

Bigraphs

- Bigraphs [Jensen, Milner03] are nested graphs representing two kinds of structure: locality (nodes may occur inside other nodes) and connectivity (nodes have ports that may be connected by links)
- Links are attached to nodes from the inside or the outside, so bigraphs have both an inner and an outer interface. A control (or agent) is *atomic* if it cannot contain a nested graph, otherwise it is non-atomic.
- The reduction relation is defined by a set of reaction rules, which are pairs of bigraphs (redex and reactum). A non-atomic control K can be specified as active, in which case reactions can occur inside, or passive, in which case reactions in the internal bigraph can only occur after the control K has been destroyed.

Bigraphs

- Bigraphs [Jensen, Milner03] are nested graphs representing two kinds of structure: locality (nodes may occur inside other nodes) and connectivity (nodes have ports that may be connected by links)
- Links are attached to nodes from the inside or the outside, so bigraphs have both an inner and an outer interface. A control (or agent) is *atomic* if it cannot contain a nested graph, otherwise it is non-atomic.
- The reduction relation is defined by a set of reaction rules, which are pairs of bigraphs (redex and reactum). A non-atomic control K can be specified as active, in which case reactions can occur inside, or passive, in which case reactions in the internal bigraph can only occur after the control K has been destroyed.
- Lafont's IN are a particular kind of bigraphs without nesting.

Bigraphical Nets

To encode the ρ-calculus we will use a subclass of bigraphs that contains interaction nets, and that we call *bigraphical nets*.

Definition

Bigraphical nets are bigraphs in which each control has a distinguished *principal* port (the remaining ports are called auxiliary). Reaction rules define interactions between two controls connected by their principal ports (and their sites), or interactions of a control with its local sites, preserving the interfaces.

Unlike IN, a left-hand side can specify the location in which the reacting controls are, or the locations contained in these controls, and reactions can take place across boundaries. Because of this, confluence does not hold in general for bigraphical nets. However reduction is still local.

Bigraphical nets can also be seen as a particular class of *higher-order nets* [Fernandez, Mackie, Pinto02]

Bigraphical Encoding

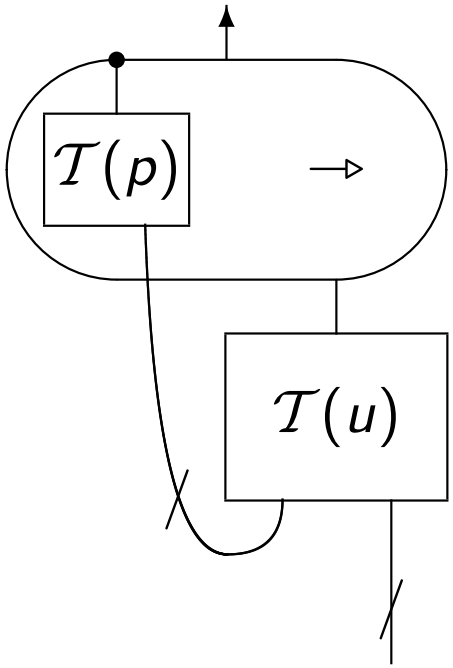
Agents:

- $\multimap$ of arity 2, which is a non-atomic agent representing abstraction;
- @ of arity 2, which is an atomic agent representing application;
- $f, g, \ldots$ of arity 0, for constants;
- a non-atomic agent M of arity 1 to represent matching problems;
- a family of non-atomic agents α_M, where α is any of the agents above except M;
- $\perp$ of arity 0 (atomic), to represent matching failure.

All non-atomic agents are active.

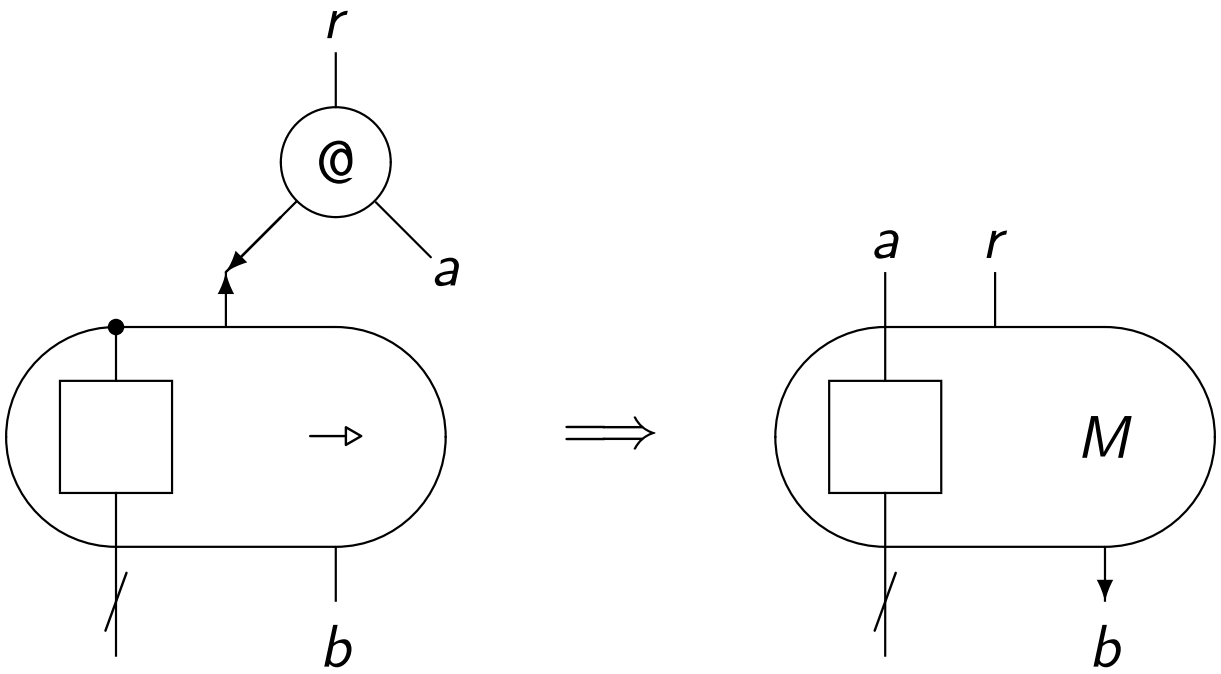
Bigraphical Encoding

The translation of an application, a variable, or a constant, are the same as in the first interaction net encoding. We give the translation of an abstraction $p \rightarrowtriangle u$ (we omit the encoding of the box).



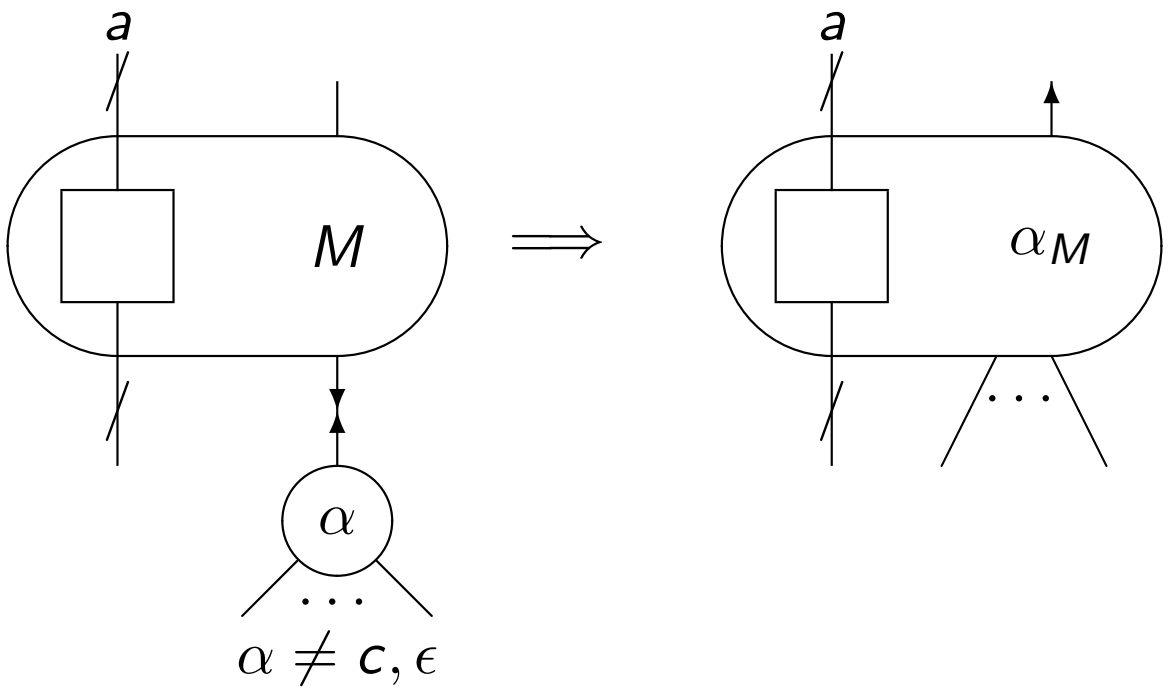
Bigraphical Rules

The ρ rule:

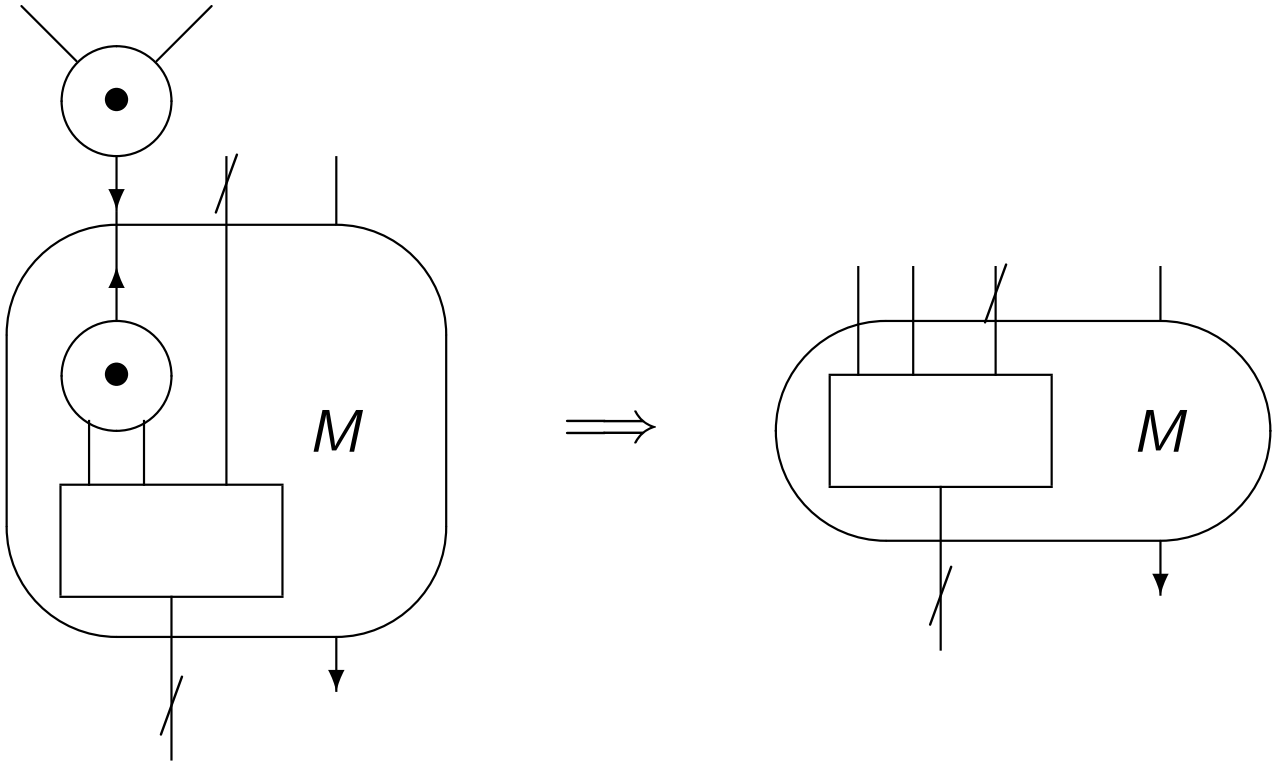


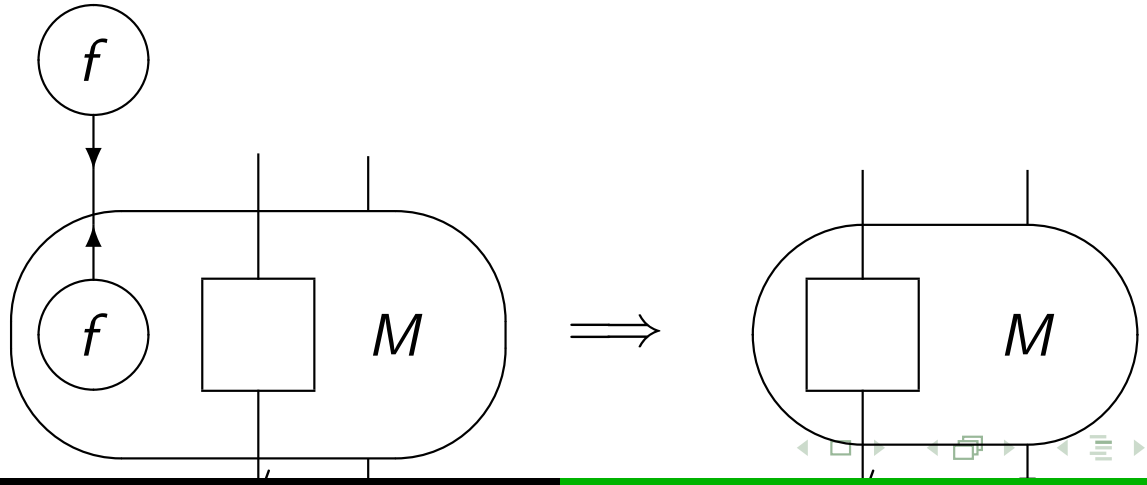
Bigraphical Rules

Rules implementing matching:

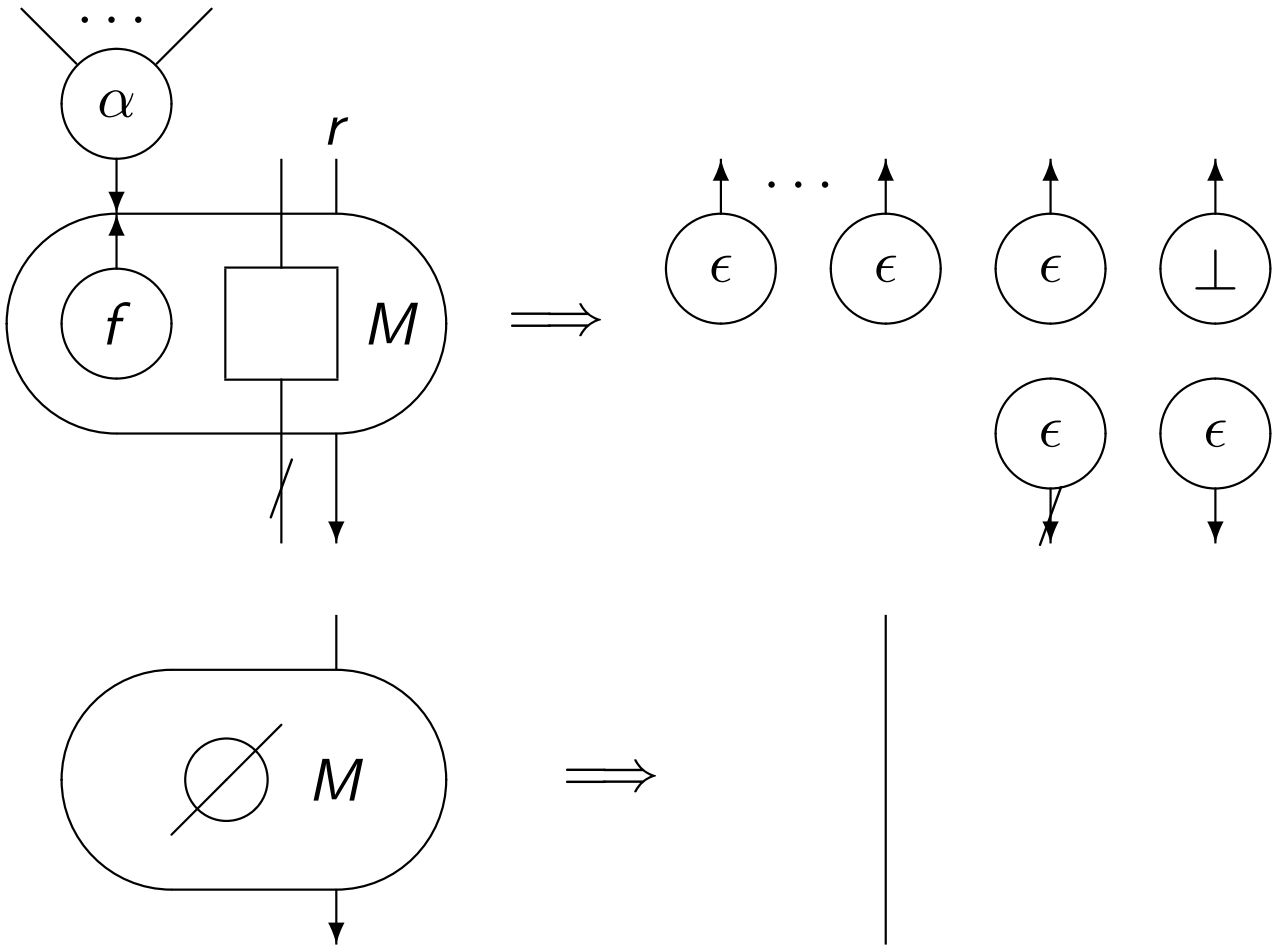


More Matching Rules



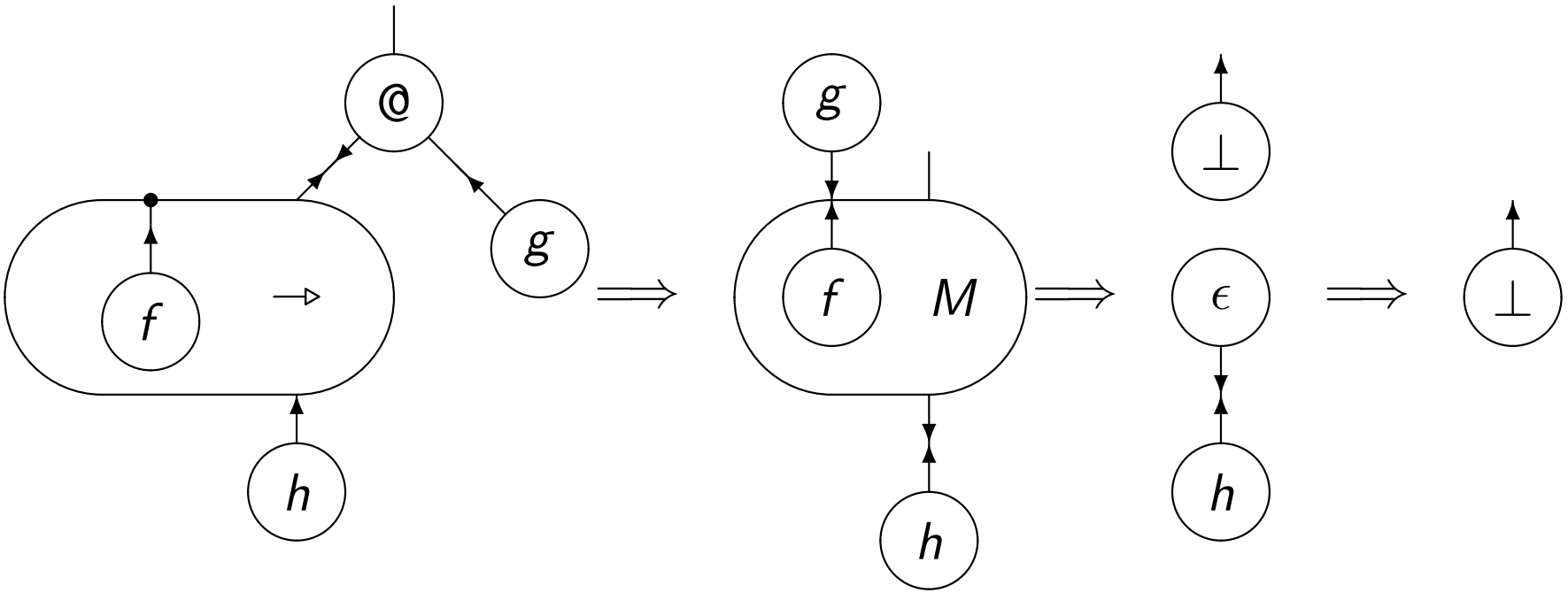


More Matching Rules



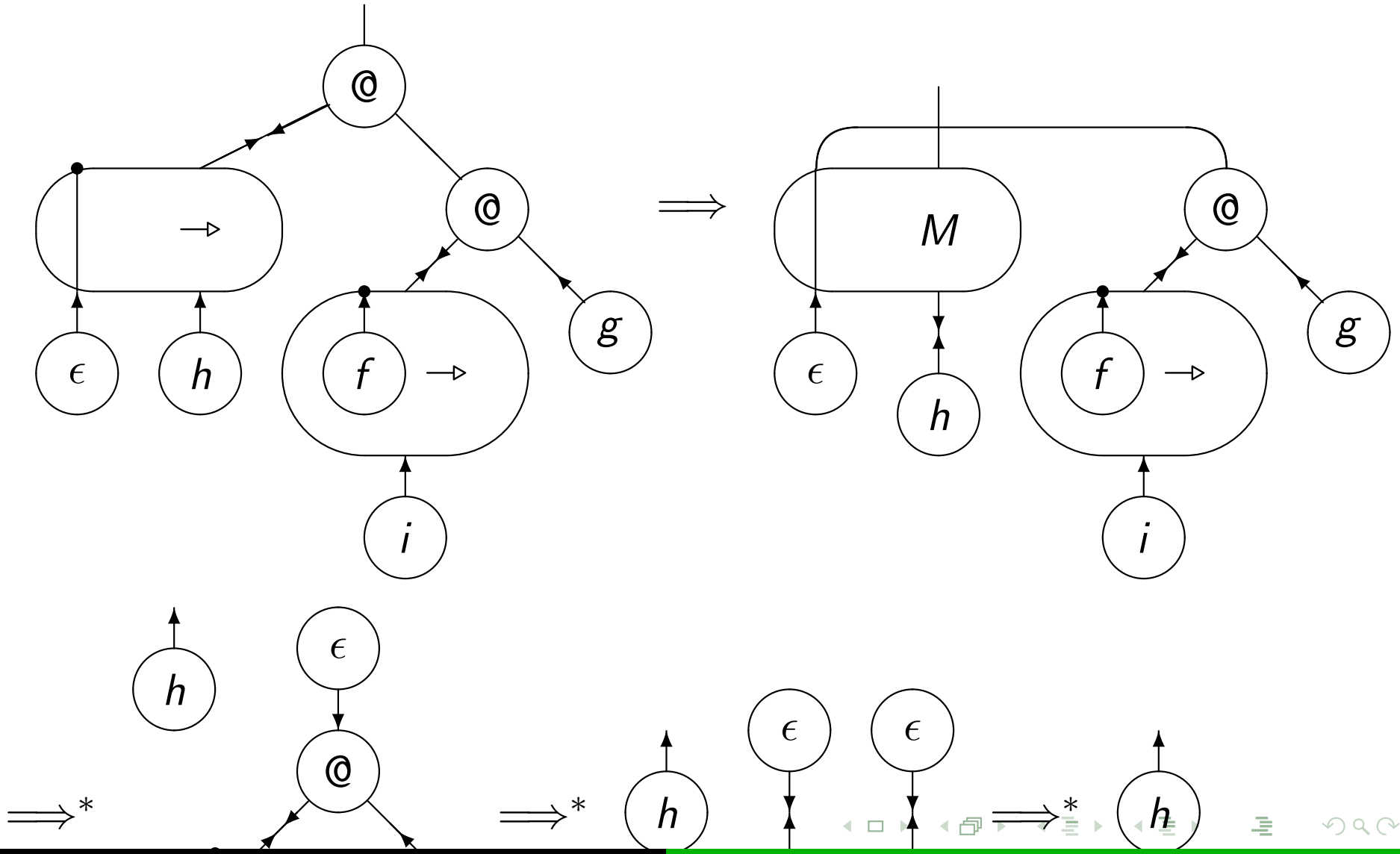
Example

$(f \rightarrowtriangle h)\ g$ produces a matching failure:



Example

$(x \rightarrow h)((f \rightarrow i)\ g)$ reduces to h:



Conclusion

The original motivations for this work were to provide grounds for implementing in a distributed setting the ρ-calculus, which can be seen as a foundational model for functional languages featuring pattern-matching, or for rewriting.
This actually led us to propose bigraphical nets as a model of distributed computation with local synchronisations which is suitable for our particular problem (and for a larger class of problems).
Future Work: study bigraphical nets!