Reaction Graph*

Yuxi Fu

Department of Computer Science
Shanghai Jiao Tong University
1954 Hua Shan Road, Shanghai 200030, China

Abstract

The paper proposes reaction graphs as graphical representations of computational objects. A reaction graph is a directed graph with all its arrows and some of its nodes labeled. Computations are modeled by graph rewriting of a simple nature. The basic rewriting rules embody the essence of both the communications among processes and cut-eliminations in proofs. Calculi of graphs are identified to give a formal and algebraic account of reaction graphs in the spirit of process algebra. With the help of the calculi, it is demonstrated that reaction graphs capture many interesting aspects of computations.

1 Introduction

Interaction diagrams are introduced in [24] as a diagrammatic description of mobile processes. There are three kinds of nodes. Free nodes are labeled; they represent free names in π-calculus. Parameter nodes are unlabeled; they correspond to local names. Input nodes are also unlabeled; they denote the input names. Arrows in interaction diagrams are classified into two groups. Input arrows model the input prefixes of π-processes whereas output arrows the output prefixes. An input arrow must point to an input node. When an output arrow ends with the node an input arrow starts, a communication can happen. Such a communication coalesces the start node of the output arrow and the end node of the input arrow, dragging the remaining arrows along the way. The communication also deletes the input and the output arrows. An interaction diagram is partitioned into regions, each of which representing a process. To achieve Turing computability, either recursion or duplication must be incorporated in the diagrammatic setting. Interaction diagrams are graphic representations of concurrent processes with changing access capability. As mathematical objects, they are not as simple as they could be.

Linear logic ([11, 33]) was introduced as a modification of classical logic. One of its aims is to achieve a proof theory for classical logic comparable to that for constructive logic. Computational interpretation of the linear logic was initiated by Abramsky ([2]). He subsequently gave a process interpretation of linear proofs (confer [3]). The approach was further investigated by Bellin and Scott ([6]) using π-calculus. The underlying idea of the proof-as-process paradigm is that cut eliminations can be interpreted as communications. What is not so clear in the process interpretation is that it is the underlying graphs of derivations of proofs that are being coded up.

In this paper we present reaction graphs as alternatives to interaction diagrams. Reaction graphs have inspiration from interaction diagrams. But the two differs significantly. Firstly there are two classes of nodes. The local nodes are unlabeled whereas the global nodes are labeled. Secondly there is only one kind of arrows. There is no structural difference between input and output arrows. In other words, computations are regarded as a symmetric operation. Thirdly the universal computing power is achieved by a simple form of duplication.

The main motivation of reaction graphs however comes from the idea of proof-as-process discipline. The rewriting of the graphs, which models computations, attempts to capture the essence of communications in concurrency theory and cut eliminations in logic. Compared to the interaction diagrams, the selling point of reaction graphs is its simplicity. There are variants of reaction graphs that have more convenient expressive power. The design decision we have made for reaction graphs trades off expressiveness for simplicity. The reaction graphs nevertheless are strong enough to capture most familiar aspects of computations. What is carried out in this paper can be seen in a different angle as repercussion of Abramsky's proposal. We attempt to substantiate a process-as-proof paradigm, which should contribute to a better understanding of both

*Journal of Computer Science and Technology, **13**(6):510-530, 1998.

disciplines.

Apart from the interaction diagram, other forms of diagrammatic representation of computing objects have also been proposed. Milner for example has studied π-nets ([19]) and Lafont has introduced interaction nets ([14]) and interaction combinators ([15]).

Section 2 defines in an informal way reaction graphs and reactions. Section 3 identifies two calculi of graphs in the spirit of process algebra. The terms of the calculus are the formal counterparts of the reaction graphs described in the previous section. Section 4 explains how reaction graphs capture some interesting aspects of computations. Some remarks are made in the final section.

The material in this paper has been announced in [9].

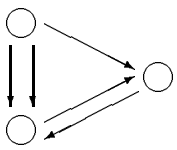
2 Reaction Graph

Reaction graphs try to represent the topological structures of the communicabilities of processes and the underlying graphs of proof derivations in a distilled form. Computations of reaction graphs model communications and cut eliminations. Technically reaction graphs are directed graphs with each of their arrows labeled by either $+$ or $-$. The labels indicate polarities of arrows and decide what can react upon each other. A reaction graph may have some of its nodes labeled. These nodes are important in that they are the channels through which environments interact with the graph. Normally we are not interested in those reaction graphs which do not have any labeled nodes. In sequel, the labels of nodes will be drawn from a set $\mathcal{N}$ of names ranged over by lower case letters. Throughout this paper, we assign a number to each of the reaction graphs given in the paper. We will refer to a reaction graph by G_i where i is the number assigned to it.

The graphs we are interested in this paper are finite directed graphs with possibly more than one arrow from one node to another. The start node and the end node of an arrow can be the same. The following is a formal definition of such graphs.

Definition 2.1 *A finite directed graph is a quadruple $\langle N, E, d_0, d_1\rangle$ where both N and E are finite sets, d_0 and d_1 are functions from E to N that specify respectively the start and the end nodes of arrows.*

Graphs may be given by diagrams. For instance



is a graph. Graphs will often be presented diagrammatically.

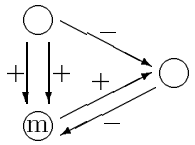
2.1 Simple Reaction Graph

Reaction graphs have two kinds of nodes. There are local nodes and global nodes. A local node is unlabeled, represented in a graph by a circle. A global node is labeled with a small case letter, denoted by a circle with the label written inside the circle.

local global

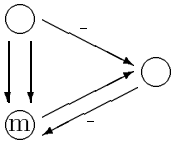
A simple reaction graph is a finite directed graph with each of its arrows labeled by either $+$ or $-$. In addition, for each name m there is at most one node labeled by m. Loops are allowed, meaning that an arrow can start and end with a same node. There can be more than one arrow from one node to another. The following is an example of simple reaction graph.

(1)



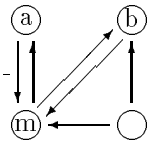
In sequel, we will omit the label '+' and shorten the label '$-$' to '-'. The above graph will look like

(1)



Next is another example of simple reaction graphs.

(2)



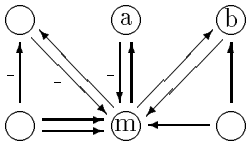
A formal definition of simple reaction graph goes as follows:

Definition 2.2 *A simple reaction graph is a sextuple $\langle N, E, d_0, d_1, o, e\rangle$ such that $\langle N, E, d_0, d_1\rangle$ is a finite directed graph, o is a partial function from N to $\mathcal{N}$ that is injective on the domain of definition, and e is a function from E to $\{-, +\}$.*

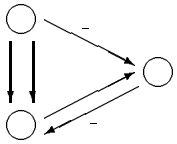
The set $\{-, +\}$ will be ranged over by p. Define $-p$ to be $+(-)$ when p is $-(+)$.

Let $F, G, \cdots$ range over reaction graphs. There are four operations on reaction graphs that can be introduced at this stage. The composition of two

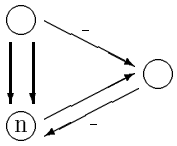
graphs F and G, notation $F|G$, is obtained from the union of F and G by identifying global nodes with same labels. The composition $G_1|G_2$, of the simple reaction graphs in (1) and (2), is the following simple reaction graph:

(3) 

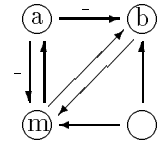
For each label m, $(\ )\backslash m$ is a unary operation on reaction graphs. $G\backslash m$ is obtained from G by removing the label m from the graph. For instance, $G_1\backslash m$ is the following graph:

(4) 

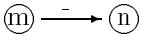
The third operation is substitution. For names a and b, $G[a/b]$ is the graph obtained from G by relabeling the node labeled b by a. For instance, $G_1[n/m]$ is the following graph:

(5) 

Notice that distinct global nodes of G might collapse into one as a result of substitution operation. The fourth operation is also unary. For each pair of names m, n, $G[m\xrightarrow{p}n]$, where $p \in \{-,+\}$, is what is obtained by adding to G an arrow with label p starting from the node labeled by m and ending at the node labeled by n. For example, $G_2[a\xrightarrow{-}b]$ is the following graph:

(6) 

In case there isn't a node labeled by m (n) in G, the operation has to add to the graph a node with that label. This operation is a derived one. $G[m\xrightarrow{-}n]$ can be defined as the composition of G with the following simple reaction graph:

(7) 

The four operations defined here also apply to reaction graphs that are not simple.

2.2 Atomic Reaction

Simple reaction graphs are meant to be computing agents. The nodes of a graph are like atoms. They react with one another, producing new atoms. The arrows indicate the valences the nodes can exhibit. For example, in the simple reaction graph G_7 given in section 2.1, the node labeled m shows up negative valence to the node labeled n.

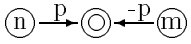
An atomic reaction happen between two local nodes or between a local node and a global node. When two local nodes show up opposite valences to a same node, they can react by removing the two arrows indicating the valences and coalescing the two nodes. This computation step can be illustrated by the following simple reaction rule, where $p \in \{-,+\}$:

(A1) 

Here the node with the incoming arrows labeled p and $-p$ acts as a catalyst. A circle with another circle inside denotes either a global node or a local node. The rule implies that a catalyst can be either local or global. Reactions in our formalism reminds one of the chemical reactions in the abstract chemical machine ([5]). If a local node and a global node show up opposite valences to a catalyst, they can also react by removing the two arrows indicating the valences and coalescing the two nodes. But now label the resulting node by the same name as the label of the global reacting node. This computation step can be illustrated by the following atomic reaction rule:

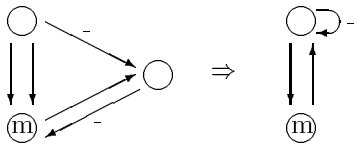
(A2) 

Another situation arises when two global nodes with opposite valences or neutral valences are connected to a same node, as shown in the following graph:

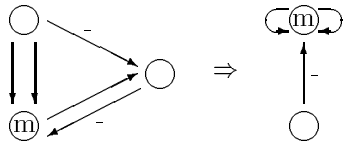
(8) 

If we merge the two nodes labeled respectively by n and m, we wouldn't know how to label the resulting node. We therefore disallow reactions between two global nodes.

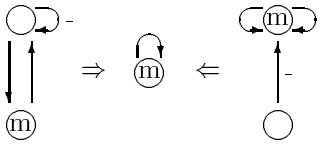
Applying the atomic computation rules, we can rewrite simple reaction graphs. Graph G_1 can be rewritten in three ways. The global name labeled by m can act as a catalyst for two reactions, which are computationally the same:



In graph G_1, the node on the right can also be a catalyst that triggers a reaction. The reduction can be pictured as follows:

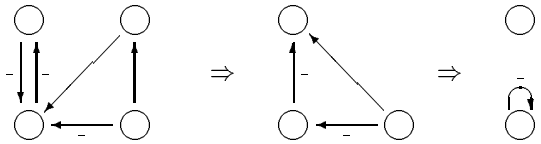


The above two reductions are independent; they can happen in parallel. In our interleaving scenario, a choice has to be made as to which reduction is conducted first. Nondeterminism is therefore intrinsic. For two independent reductions, the choice of reduction order is immaterial. For instance, the above two computations can be continued, ending with a same graph:



The middle simple reaction graph is normal; it cannot be reduced further.

Reactions are not always independent. Take the next reduction sequence for instance.

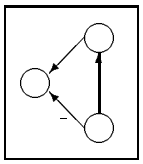


Here the second reduction depends causally on the first one. Causality is an important aspect of any model for computations.

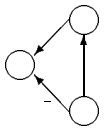
2.3 Reaction Graph

A reaction graph can be repetitively rewritten by applying (A1) and (A2). This procedure must terminate in a finite number of steps as each step decreases the number of arrows by two. To achieve the universal computing power, we must go beyond the *simple* framework.

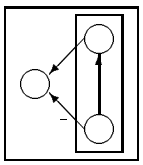
The solution we are interested in is to introduce another kind of nodes, called molecular nodes. A molecule consists of a set of non-global nodes with coupling relationships among them. Graphically a molecule is described by a square so that all the relevant nodes lie within the square. The following is a molecule:



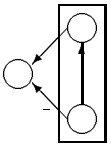
We say that the simple reaction graph



is the internal structure of the molecule and the three nodes the component nodes of the molecule. A component node of a reaction graph can itself be a molecule:



whose internal structure is the reaction graph



which is not simple.

Reaction graphs and molecules are mutually defined as follows:

- A simple reaction graph is a reaction graph;
- If G is a reaction graph, then one can get another reaction graph by placing a square in G that covers at least one non-global node. The nodes lie inside a square must be either local or molecular
- A molecular node, or molecule, is a reaction graph encompassed in a square.

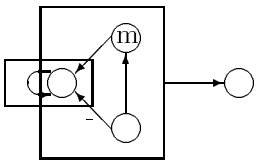
In a sense, the internal structure of a molecule is not completely shielded from outside. In a reaction graph, an arrow pointing to a molecular node actually points to a component node of the molecule. Similarly, an arrow comes from a molecule actually comes from a node inside the molecule.

When drawing a molecule, the following conditions should be observed:

- all the arrows between the nodes inside a square lie inside the square as well;
- all the arrows between the nodes outside a square lie outside the square as well;

- a node can not lie on the edge of a square;
- squares can be nested but not overlapped.

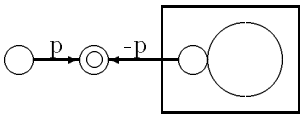
The following is not a reaction graph:



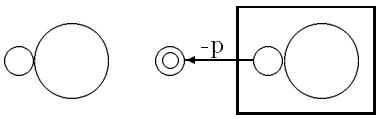
It contains a global node. The diagrammatic representation violates our requirement in three accounts: Firstly the loop on the left node props out of the big square. Secondly the arrow points to the right node does not come from a component node. Finally the two squares overlap.

2.4 Molecular Reaction

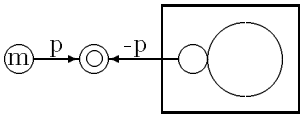
Molecular reactions allow a molecule to duplicate itself, thus permitting divergent computations. When a local node reacts to a molecule, it first makes a copy of the internal structure of the molecule and then the local node reacts to the associated component node in the replica. The following molecular rule intend to capture the intuition:



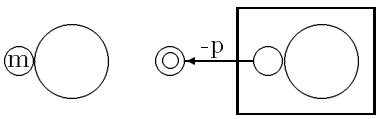
(M1) ⇓



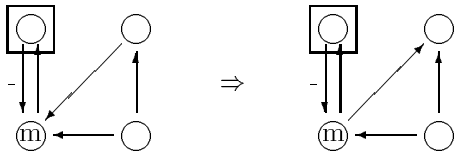
Similarly we have:



(M2) ⇓

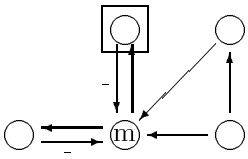


Here is an example of molecular rewriting:

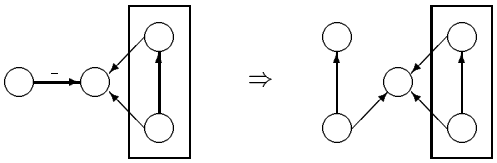


As we have said, this reduction is carried out in two stages. First the molecule involved in the reaction makes a *local* copy of its internal structure, giving rise to G_9:

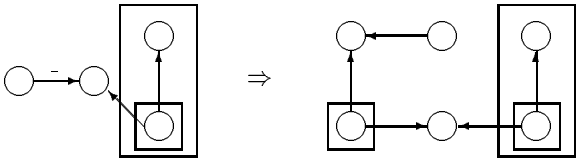
(9)



and then the local copy participates in the reaction. The following is another example of molecular reduction



The molecular reaction rule does not cover the situation where one of the reacting atom lie in a nested molecule. In the following reduction, a local node reacts to an atom inside a molecule inside another molecule:



The internal structure of the molecule was first copied. The internal structure of the smaller molecule in the replica is copied afterwards. Then the reaction happened.

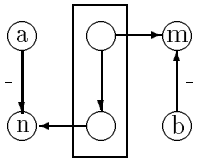
In general a molecular reaction should make successively internal structures of nested molecules until the relevant atom becomes nude. The reaction then follows.

For simplicity, we do *not* consider nested molecular reaction in this paper. We also disallow reactions between two molecular nodes.

A molecular node can not act as a catalyst.

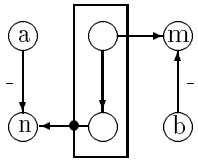
2.5 Guarded Graphs

A molecular node in a reaction graph can react to different atomic nodes via different arrows. In the following reaction graph both the node a and the

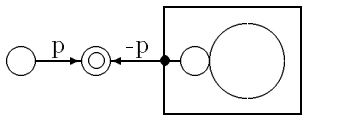


node b can react to a node in a replica of the

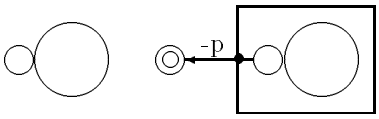
molecule. In many situations it is convenient to have guarded molecules. A guarded molecule is a molecule that has a principal component node that is local. In a graph, every guarded molecule has a principal arrow that starts from the principal component node of the molecule and ends at an atomic node outside the molecule. The molecule in the following graph is guarded.



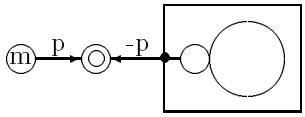
The bottom node in the molecule is the principal node and the principal arrow is indicated by an enlarged dot at the junction of the arrow and the square. The only way an atomic node can react upon a guarded molecule is with the principal node through the principal arrow. For instance there is only one initial reaction in the above graph. By a guarded reaction graph we will refer to either a simple reaction graph or a reaction graph with guarded molecules. The molecular rules are



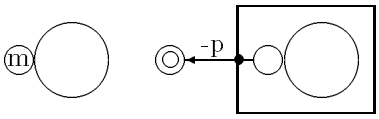
(M1') ⇓



and



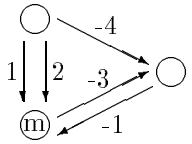
(M2') ⇓



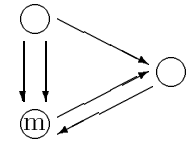
Guarded reaction graphs are introduced because they have better algebraic properties and are easier to implement.

2.6 Classification of Graphs

The labels of the reaction graphs described so far range over the set $\{-,+\}$. Only arrows with opposite labels can react upon each other. We will call such reaction graphs diadic. If instead we let the labels be drawn from the set $\mathcal{Z}$ of integers, we get polyadic reaction graphs. The following is a polyadic reaction graph:



If we go to extreme, we can draw the labels from a singleton set. This is the same as not to label arrows at all. Below is such a graph



We call such graphs monadic reaction graphs[1]. In a polyadic reaction graphs, an arrow labeled by i can react to an arrow labeled by $-i$ if they point to a same atom.

3 Calculi of Graphs

Instead of giving a formal definition of reaction graphs in a set theoretical setting, we define in this section calculi for reaction graphs in the spirit of process algebra ([16, 20, 18]). Two languages will be proposed. One is for the plain reaction graphs. The other is for the guarded reaction graphs. Strictly speaking, the terms of the calculi are more general than the graphs. But the generality is only technical. Graph terms are those terms that correspond precisely to reaction graphs. The non-graph terms can be coded up in graph terms. The encoding is faithful both operationally and algebraically. Similarly one can also identify a class of terms in the calculus of guarded reaction graphs that are in bijective correspondence to guarded graphs. Any term can be transferred to one such term. The transformation also preserves operational and algebraic semantics. We will concentrate on polyadic reaction graphs. All results in this section hold in the diadic and monadic cases as well.

3.1 Calculus of Graphs

The calculus of graphs intends to give term representations of reaction graphs. We first define the set of terms by the following abstract syntax:

$$P := \mathbf{0} \mid m_i[x] \mid (x)P \mid P|P' \mid !P,$$

[1] Our convention on drawing diadic reaction graphs is now causing confusion. Such ambiguity never arises in this paper.

where $i \in \mathcal{Z}$. Here m and x range over the set $\mathcal{N}$ of names. In $(x)P$, x is local, meaning that it can not be seen from outside. We will abbreviate $(x_1)\ldots(x_n)P$ to $(\vec{x})P$. Terms of the from $P|Q$ are composition terms. Terms of the form $m_i[x]$ are called atomic terms whereas those of the form $!P$ molecular terms. In $m_i[n]$, m is subjective and n is objective. It is helpful to think of $m_i[n]$ as describing the situation where the datum n is stored at the i-th port of medium m. We say that an atomic term $m_i[n]$ is free in P if it is a subterm of P and both m and n are global in P. Let $\mathcal{T}$ be the set of all terms. We will adopt the so-called α-convention which says that replacing a local name in a term by a fresh name does not change the syntax of the term. For diadic graphs, the syntax of the calculus is defined as follows:

$$P := \mathbf{0} \mid m[x] \mid \overline{m}[x] \mid (x)P \mid P|P' \mid !P.$$

And for monadic graphs,

$$P := \mathbf{0} \mid m[x] \mid (x)P \mid P|P' \mid !P.$$

Before defining the operational semantics of the calculus, we need to define a structural equivalence on the set of all terms.

Definition 3.1 *The structural relation $=$ is the least congruence on terms that contains:*
(i) $P|\mathbf{0} = P$, $P_1|P_2 = P_2|P_1$, and $P_1|(P_2|P_3) = (P_1|P_2)|P_3$;
(ii) $(x)\mathbf{0} = \mathbf{0}$, $(x)(y)P = (y)(x)P$, $(x)(P|Q) = P|(x)Q$ if $x \notin gn(P)$.

We think of $=$ as a grammatic equality. So $P{=}Q$ means that P and Q are the same term.

A term P is in standard form if it is either $\mathbf{0}$ or of the form,

$$(\vec{x})(A_1|\ldots|A_i|!M_1|\ldots|!M_j),$$

where $i{+}j \geq 1$, $A_1, \ldots, A_i$ are atomic terms and $M_1, \ldots, M_j$ are in standard forms. In the latter case $A_1, \ldots, A_i$ will be called the atomic components of P and $!M_1, \ldots, !M_j$ the molecular components of P.

Lemma 3.2 *For any term P there is a term Q in standard form such that $P{=}Q$.*

In sequel the notation $[y/x]$ stands for an atomic substitution. The result of substituting y for x throughout P is denoted by $P[y/x]$. Local names in P need be renamed to avoid y being captured. A substitution $[y^1/x^1]\ldots[y^n/x^n]$ is a concatenation of atomic substitutions. The effect of a substitution on a term is defined as follows:

$$P[] \stackrel{\text{def}}{=} P$$
$$P[y^1/x^1]\ldots[y^n/x^n] \stackrel{\text{def}}{=} (\ldots P[y^1/x^1]\ldots)[y^n/x^n]$$

where $[]$ is the empty substitution. Substitutions will be ranged over by σ.

The operational semantics for the calculus of graphs is given in a reductional style rather than by a labeled transition system. It consists of the following reduction rules.

Atomic Reduction

$$(x)(m_i[x]|m_{-i}[y]|P) \rightarrow (x)(P[y/x])$$

Molecular Reduction

$$m_i[x]|!(y)(m_{-i}[y]|P) \rightarrow P[x/y]|!(y)(m_{-i}[y]|P)$$

Structural Rule

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \frac{P \rightarrow P'}{(x)P \rightarrow (x)P'}$$

Let $\rightarrow^+$ ($\rightarrow^*$) be the (reflexive and) transitive closure of $\rightarrow$.

Lemma 3.3 *If $P \rightarrow P'$ then $P[y/x] \rightarrow P'[y/x]$.*

We now investigate the algebraic theory of the language. Suppose a is a global name in term P. We say that a is an *immediate barb* of P, notation $P{\downarrow}a$, if

- $P = a_i[x]$ for some $i \in \mathcal{Z}$;
- $P = P_1|P_2$ and either $P_1{\downarrow}a$ or $P_2{\downarrow}a$;
- $P = (x)P'$ and $P'{\downarrow}a$ and $x \neq a$.

Write $P{\downarrow}Q$ if $\forall a \in \mathcal{N}.P{\downarrow}a \leftrightarrow Q{\downarrow}a$. We say that a is a barb of P, notation $P{\Downarrow}a$, if Q exists such that $P \rightarrow^* Q{\downarrow}a$. For two terms P and Q, write $P{\Downarrow}Q$ if $\forall a \in \mathcal{N}.P{\Downarrow}a \Leftrightarrow Q{\Downarrow}a$. A binary relation $\mathcal{R}$ is barbed if $P\mathcal{R}Q$ implies $P{\Downarrow}Q$. Our definition of barb is non-standard in that we completely ignore the internal structure of molecules. It is reasonable for the version of barbed bisimilarity introduced below.

Lemma 3.4 *If $P{=}Q$ then $P{\downarrow}Q$ and $P{\Downarrow}Q$.*

Lemma 3.5 *If $P \rightarrow Q$ and $Q{\Downarrow}a$ then $P{\Downarrow}a$.*

There are many ways to define an equivalence relation on processes ([23, 16, 26, 27, 25]). What seems most suitable to the terms of the calculus of graphs is the barbed bisimilarity introduced in [21]. Barbed bisimulations are defined in terms of a reduction semantics and an observation predicate. The definition below is closer to the version of Honda and Yoshida ([13]).

Definition 3.6 *Let $\mathcal{R}$ be a subset of $\mathcal{T}\times\mathcal{T}$. The relation $\mathcal{R}$ is a barbed bisimulation if it is barbed and whenever $P\mathcal{R}Q$ then for any term R and any sequence $\vec{x}$ of names, it holds that*
(i) if $(\vec{x})(P|R) \rightarrow P'$, then there exists some Q' such that $(\vec{x})(Q|R) \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(Q|R) \rightarrow Q'$, then there exists some P' such that $(\vec{x})(P|R) \rightarrow^* P'$ and $P'\mathcal{R}Q'$.
The barbed bisimilarity $\approx$ is the largest barbed bisimulation.

According to the definition, one has that $!!P \approx \mathbf{0}$ and $!m_i[x] \approx \mathbf{0}$.

Definition 3.7 *Let $\mathcal{R}$ be a subset of $\mathcal{T}\times\mathcal{T}$. The relation $\mathcal{R}$ is a barbed bisimulation up to $\approx$ if it is barbed and whenever $P\mathcal{R}Q$ then for any term R and any sequence $\vec{x}$ of names, it holds that*
(i) if $(\vec{x})(P|R) \rightarrow^* P'$, then there exists some Q' such that $(\vec{x})(Q|R) \rightarrow^* Q'$ and $P' \approx \mathcal{R} \approx Q'$;
(ii) if $(\vec{x})(Q|R) \rightarrow^* Q'$, then there exists some P' such that $(\vec{x})(P|R) \rightarrow^* P'$ and $P' \approx \mathcal{R} \approx Q'$.

Care should be given to proofs using bisimulation up to technique ([29]).

Lemma 3.8 *If $\mathcal{R}$ is a barbed bisimulation up to $\approx$, then $\mathcal{R} \subseteq \approx$.*

Theorem 3.9 *Suppose $P \approx Q$. Then*
(i) $P|R \approx Q|R$ for any $R \in \mathcal{T}$;
(ii) $(x)P \approx (x)Q$.

PROOF: (i) If $P \approx Q$ then $P|O \approx Q|O$. Consider $\mathcal{R} \stackrel{\text{def}}{=} \{(P|O, Q|O) | P \approx Q \wedge O \in \mathcal{T}\} \cup \approx$. Suppose $(\vec{x})(P|O|R) \rightarrow P'$. Then by definition some Q' exists such that $(\vec{x})(Q|O|R) \rightarrow^* Q'$, $P' \approx Q'$ and $P' \Downarrow Q'$. Hence $\mathcal{R}$ is a barbed bisimulation.
(ii) If $P \approx Q$ then $(y)P \approx (y)Q$. This can be proved by showing that $\mathcal{R} \stackrel{\text{def}}{=} \{((\vec{y})P, (\vec{y})Q) | P \approx Q\}$ is a barbed bisimulation. Suppose $(\vec{x})((\vec{y})P|R) \rightarrow P'$. Then $(\vec{x})(\vec{y})(P|R) \rightarrow P'$. By definition, $(\vec{x})(\vec{y})(Q|R) \rightarrow^* Q'$ such that $P' \approx Q'$ and $P' \Downarrow Q'$. Therefore $\mathcal{R}$ is a barbed bisimulation. □

Unfortunately $\approx$ is not a congruence relation. For a counter example, notice that

$$(x)m_i[x] \approx (x)(a)(a_i[x]|(b)(b_0[a]|b_0[m])).$$

But $!(x)(a)(a_i[x]|(b)(b_0[a]|b_0[m]))$ is clearly not bisimilar to $!(x)m_i[x]$ as the former is barbed bisimilar to $\mathbf{0}$ whereas the latter is not. This is part of the reason to introduce guarded molecules.

Before ending this section, we prove an important technical lemma.

Lemma 3.10 *If $P \approx Q$, then $P[y/x] \approx Q[y/x]$.*

PROOF: Suppose $P \approx Q$ and $P[y/x] \rightarrow P'$. By theorem 3.9, $(x)(P|a_0[y]|a_0[x]) \approx (x)(P|a_0[y]|a_0[x])$ for a fresh name a. Assuming $x \neq y$, one has

$$(x)(P|a_0[y]|a_0[x]) \rightarrow P[y/x] \rightarrow P'$$

By definition, $(x)(Q|a_0[y]|a_0[x]) \rightarrow^* Q'$ such that $P' \approx Q'$ and $P' \Downarrow Q'$. So the communication between $a_0[x]$ and $a_0[y]$ must have happened before reaching Q'. It can be easily seen from lemma 3.3 that the communication through a can happen in the very first place:

$$\begin{array}{rcl} (x)(Q|a_0[y]|a_0[x]) & \rightarrow & Q[y/x] \\ & \rightarrow^* & Q'. \end{array}$$

So $P[y/x] \rightarrow P'$ is matched by $Q[y/x] \rightarrow^* Q'$. □

3.2 Calculus of Guarded Graphs

The reaction graphs are simpler than the guarded reaction graphs. But the latter enjoys a better algebraic theory, as will be shown below. The terms of the calculus of guarded reaction graphs are defined as follows:

$$P := \mathbf{0} \mid m_i[x] \mid (x)P \mid P|P' \mid m_i(x)*P,$$

where $i \in \mathcal{Z}$. The operational semantics of this calculus is defined similarly to the one for the calculus of graphs. Only the molecular rule need be redefined.

Molecular Reduction

$$m_i[x]|m_{-i}(y)*P \rightarrow P[x/y]|m_{-i}(y)*P.$$

The algebraic theory of this language can be studied in completely the same way as we have done with the calculus of graphs. All results in section 3.1 also hold for the calculus of unguarded graphs. However the bisimilarity for the terms of the calculus of guarded graphs enjoys much better algebraic properties as the following theorem shows.

Theorem 3.11 *The barbed bisimilarity is a congruence equivalence.*

PROOF: We only have to prove that if $P \approx Q$ then $m_i(x)*P \approx m_i(x)*Q$. Let $\mathcal{R}$ be the following relation

$$\left\{ ((\vec{x})(R|m_i(x)*P), (\vec{x})(R|m_i(x)*Q)) \;\middle|\; \begin{array}{l} P \approx Q \\ R \in \mathcal{T} \\ \vec{x} \in \mathcal{N} \end{array} \right\}.$$

Suppose $(\vec{x})(R|m_i(x)*P) \rightarrow P'$. There are two cases:

- $(\vec{x})(R|m_i(x)*P) \rightarrow P'$ is caused by a communication within R. P' must be of the form $(\vec{x'})(R'|(m_i(x)*P)\sigma)$. Then obviously

  $$(\vec{x})(R|m_i(x)*Q) \rightarrow (\vec{x'})(R'|(m_i(x)*Q)\sigma).$$

  By lemma 3.10,

  $$(\vec{x'})(R'|(m_i(x)*P)\sigma)\mathcal{R}(\vec{x'})(R'|(m_i(x)*Q)\sigma).$$

- $(\vec{x})(R|m_i(x)*P) \rightarrow P'$ is caused by a communication between R and $m_i(x)*P$. Then P' is of the form $(\vec{x'})(R'|P[y/x]|m_i(x)*P)$. Similarly

  $$(\vec{x})(R|m_i(x)*Q) \rightarrow (\vec{x'})(R'|Q[y/x]|m_i(x)*Q).$$

  Now

  $$\begin{array}{rl} & (\vec{x'})(R'|P[y/x]|m_i(x)*P) \\ \approx \mathcal{R} \approx & (\vec{x'})(R'|Q[y/x]|m_i(x)*Q) \end{array}$$

  by theorem 3.9 and lemma 3.10.

Conclude that $\mathcal{R}$ is a bisimulation up to $\approx$. □

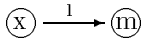
3.3 Graph Terms

A graph term P is a term such that none of its molecular subterms contains any free atomic components. Graph terms correspond to reaction graphs.

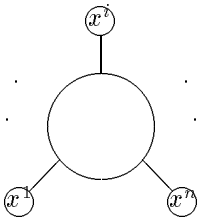
Lemma 3.12 *If $P \rightarrow Q$ and P is a graph term, then Q is a graph term.*

An inductive definition of a translation g from graph terms to reaction graphs goes as follows:

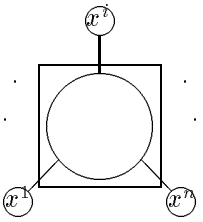
- $\mathsf{g}(\mathbf{0})$ is the empty reaction graph;
- $\mathsf{g}(m_i[x])$ is the following reaction graph:

  which is a basic building block in the construction of reaction graphs;
- $\mathsf{g}((x)P)$ is the reaction graph $\mathsf{g}(P)\backslash x$;
- $\mathsf{g}(P|Q)$ is the reaction graph $\mathsf{g}(P)|\mathsf{g}(Q)$;
- if $\mathsf{g}(P)$ is the reaction graph

  where $x^1, \ldots, x^n$ are the global names in P, then $\mathsf{g}(!P)$ is the following reaction graph:

This completes the definition of g. Suppose P is a graph term. The following properties are obvious: (i) if $P \rightarrow Q$, then $\mathsf{g}(P) \rightarrow \mathsf{g}(Q)$; (ii) if $\mathsf{g}(P) \rightarrow G$, then Q exists such that $P \rightarrow Q$ and $\mathsf{g}(Q)$ is G. On the other hand, it is clear that every reaction graph can be translated into a graph term. The two translations are in an obvious sense reversible to each other.

How about the non-graph terms? The fact of the matter is that the difference between graph terms and non-graph terms is only technical. There is a translation from terms to graph terms that preserves both operational and algebraic semantics. First we define an auxiliary translation $(\_)_\psi$, where ψ is a finite set of names:

$$\begin{array}{rcl} (\mathbf{0})_\psi & \stackrel{\text{def}}{=} & \mathbf{0} \\ (P|Q)_\psi & \stackrel{\text{def}}{=} & (P)_\psi|(Q)_\psi \\ ((x)P)_\psi & \stackrel{\text{def}}{=} & (x)(P)_{\psi\cup\{x\}} \\ (!P)_\psi & \stackrel{\text{def}}{=} & !P \\ (m_i[x])_\psi & \stackrel{\text{def}}{=} & m_i[x], \text{ when } \{m,x\} \cap \psi \neq \emptyset \\ (m_i[x])_\psi & \stackrel{\text{def}}{=} & (a)(a_i[x]|(b)(b_0[a]|b_0[m])), \\ & & \text{when } \{m,x\} \cap \psi = \emptyset. \end{array}$$

Lemma 3.13 *For a term P in the calculus of graphs and a finite set of names ψ, $(P)_\psi \rightarrow^* P$ and $(P)_\psi \approx P$.*

The translation can then be defined as follows:

$$\begin{array}{rcl} \mathbf{0}^\circ & \stackrel{\text{def}}{=} & \mathbf{0} \\ (m_i[x])^\circ & \stackrel{\text{def}}{=} & m_i[x] \\ (P|Q)^\circ & \stackrel{\text{def}}{=} & P^\circ|Q^\circ \\ ((x)P)^\circ & \stackrel{\text{def}}{=} & (x)P^\circ \\ (!P)^\circ & \stackrel{\text{def}}{=} & !(P^\circ)_\emptyset. \end{array}$$

It is clear that if P is a term then P° is a graph term. The following two theorems guarantee that we do not lose any expressive power by restricting our attention to graph terms.

Theorem 3.14 *Suppose P and Q are terms in the calculus of graphs. Then*
(i) if $P \rightarrow Q$, then $P^\circ \rightarrow^+ Q^\circ$;
(ii) if $P^\circ \rightarrow P'$, then P_1 exists such that $P \rightarrow P_1$, $P' \rightarrow^+ P_1^\circ$ and $P' \approx P_1^\circ$.

PROOF: (i) If $P \rightarrow Q$ is caused by an atomic communication, then clearly $P^\circ \rightarrow Q^\circ$. If $P \rightarrow Q$ is caused by a molecular communication, then use lemma 3.13.
(ii) Equally simple. □

Theorem 3.15 *$P \approx P^\circ$ for any term in the calculus of graphs.*

PROOF: The relation $\{(P, P^\circ) \mid P \text{ a term}\}$ is a bisimulation up to $\approx$. □

Corollary 3.16 *$P \approx Q$ if and only if $P^\circ \approx Q^\circ$.*

Next we take a look at the graph terms in the calculus of guarded graphs. In the calculus of guarded graphs, the graph terms are also strong enough to code up all the non-graph terms. The following simple translation is part of the encoding to be defined later.

$$\begin{array}{rcl}
\overline{\mathbf{0}} & \stackrel{\text{def}}{=} & \mathbf{0} \\
\overline{m_i[x]} & \stackrel{\text{def}}{=} & (a)(a_i[x]|(b)(b_0[a]|b_0[m])) \\
\overline{P|Q} & \stackrel{\text{def}}{=} & \overline{P}|\overline{Q} \\
\overline{(x)P} & \stackrel{\text{def}}{=} & (x)\overline{P} \\
\overline{m(x)*P} & \stackrel{\text{def}}{=} & m(x)*P.
\end{array}$$

Lemma 3.17 *For a term P in the calculus of guarded graphs, $\overline{P} \rightarrow^* P$ and $\overline{P} \approx P$.*

Now the encoding:

$$\begin{array}{rcl}
\widehat{\mathbf{0}} & \stackrel{\text{def}}{=} & \mathbf{0} \\
\widehat{m_i[x]} & \stackrel{\text{def}}{=} & m_i[x] \\
\widehat{P|Q} & \stackrel{\text{def}}{=} & \widehat{P}|\widehat{Q} \\
\widehat{(x)P} & \stackrel{\text{def}}{=} & (x)\widehat{P} \\
\widehat{m_i(x)*P} & \stackrel{\text{def}}{=} & m_i(x)*\overline{\widehat{P}}.
\end{array}$$

The proofs of the next three results are similar to those for the corresponding results in the calculus of graphs.

Theorem 3.18 *Suppose P and Q are terms of the calculus of guarded graphs. Then*
(i) if $P \rightarrow Q$ then $\widehat{P} \rightarrow^+ \widehat{Q}$;
(ii) if $\widehat{P} \rightarrow P'$ then there exists some P_1 such that $P \rightarrow P_1$, $P' \rightarrow \widehat{P_1}$ and $P' \approx \widehat{P_1}$.

Theorem 3.19 *For a term P in the calculus of guarded graphs, $P \approx \widehat{P}$.*

Corollary 3.20 *For terms P and Q in the calculus of guarded graphs, $P \approx Q$ if and only if $\widehat{P} \approx \widehat{Q}$.*

Results in this section substantiate our claim that the calculus of (guarded) graphs is a formal language for (guarded) reaction graphs. If something can be coded up in the calculus, it can also be coded up in the reaction graphs and vice versa.

3.4 Equivalence of Calculi

The terms of the guarded reaction graphs are better behaved. On the other hand the terms of the unguarded reaction graphs have more concise graphic representations. The two languages are however equivalent. The following translation from the guarded terms to the terms should look familiar:

$$\begin{array}{rcl}
\mathbf{0}^! & \stackrel{\text{def}}{=} & \mathbf{0} \\
(m_i[x])^! & \stackrel{\text{def}}{=} & m_i[x] \\
(P|Q)^! & \stackrel{\text{def}}{=} & (P)^!|(Q)^! \\
((x)P)^! & \stackrel{\text{def}}{=} & (x)P^! \\
(m_i(x)*P)^! & \stackrel{\text{def}}{=} & !(x)(m_i[x]|(\overline{P})^!).
\end{array}$$

A reverse translation from the terms to the guarded terms can be defined as follows:

$$\begin{array}{rcl}
\mathbf{0}^\star & \stackrel{\text{def}}{=} & \mathbf{0} \\
(m_i[x])^\star & \stackrel{\text{def}}{=} & m_i[x] \\
(P|Q)^\star & \stackrel{\text{def}}{=} & P^\star|Q^\star \\
((x)P)^\star & \stackrel{\text{def}}{=} & (x)P^\star \\
(!P)^\star & \stackrel{\text{def}}{=} & m_{i_1}^1(x_1)*P_1^\star|\ldots|m_{i_j}^j(x_j)*P_j^\star,
\end{array}$$

where $m_{i_1}^1 \neq x_1, \ldots, m_{i_j}^j \neq x_j$ for some $j \geq 0$ and

$$P = (x_1)(m_{i_1}^1[x_1]|P_1) = \cdots = (x_j)(m_{i_j}^j[x_j]|P_j).$$

Here $m_{i_1}^1[x_1], \ldots, m_{i_j}^j[x_j]$ are all the atomic components of P whose subjects are not local in P and whose objects are local in P. Notice that when j=0, $m_{i_1}^1(x_1)*P_1^\star|\ldots|m_{i_j}^j(x_j)*P_j^\star$ is $\mathbf{0}$.

Let's see an example:

$$\begin{array}{rl}
 & (!(x)(y)(m_0[x]|n_0[y]|n_0[y]|!Q))^\star \\
= & m_0(x)*(n_0[y]|n_0[y]|(!Q)^\star) \\
 & |n_0(y)*(m_0[x]|n_0[y]|(!Q)^\star) \\
 & |n_0(y)*(m_0[x]|n_0[y]|(!Q)^\star).
\end{array}$$

The proof of the next theorem is similar to that of theorem 3.14.

Theorem 3.21 *Suppose P and Q are terms of the calculus of graphs. Then*
(i) if $P \rightarrow Q$ then $P^\star \rightarrow^+ Q^\star$;

(ii) if $P^\star \to P'$ then P_1 exists such that $P \to P_1$, $P' \to (P_1)^\star$ and $P' \approx P_1^\star$.
Suppose P and Q are terms of the calculus of guarded graphs. Then
(i) if $P \to Q$ then $P^! \to^+ Q^!$;
(ii) if $P^! \to P'$ then P_1 exists such that $P \to P_1$, $P' \to (P_1)^!$ and $P' \approx P_1^!$.

The composition of the two translations is not an identity function in either way. But they are inverse to each other up to bisimilarity in the sense of the following theorem.

Theorem 3.22 *The following properties hold:*
(i) if P is a term in the calculus of guarded graphs, then $(P^!)^\star \approx P$;
(ii) if P is a term in the calculus of graphs, then $(P^\star)^! \approx P$.

Proof: (i) The proof is carried out by induction on the structures of terms. The only non-trivial case is for molecular terms:

$$\begin{aligned}((m_i(x)*P)^!)^\star &= (!(x)(m_i[x]|(\overline{P})^!))^\star \\ &= m_i(x)*((\overline{P})^!)^\star.\end{aligned}$$

By lemma 3.17, $\overline{P} \approx P$. By induction hypothesis, $((\overline{P})^!)^\star \approx \overline{P}$ and $(P^!)^\star \approx P$. Hence by theorem 3.11,

$$\begin{aligned}((m_i(x)*P)^!)^\star &= m_i(x)*((\overline{P})^!)^\star \\ &\approx m_i(x)*\overline{P} \\ &\approx m_i(x)*P \\ &\approx m_i(x)*(P^!)^\star.\end{aligned}$$

This completes the proof of (i).
(ii) Let $\mathcal{R}$ be the following relation:

$$\left\{ ((\vec{x})(R|P), (\vec{x})(R|(P^\star)^!)) \;\middle|\; \begin{array}{l} P \in \mathcal{T} \\ R \in \mathcal{T} \\ \vec{x} \text{ names} \end{array} \right\}.$$

We show that $\mathcal{R}$ is a barbed bisimulation up to $\approx$. The only nontrivial case is when P is a molecular term. So assume

$$(!P)^\star = m_{i_1}^1(x^1)*P_1^\star | \ldots | m_{i_j}^j(x^j)*P_j^\star.$$

Then $((!P)^\star)^!$ is

$$!(x)(m_{i_1}^1[x^1]|(\overline{P_1^\star})^!)| \ldots |!(x)(m_{i_j}^j[x^j]|(\overline{P_j^\star})^!).$$

Suppose R is a term and $\vec{x}$ is a sequence of names. Suppose further that $(\vec{x})(R|P) \to P'$. There are two cases:

- If $(\vec{x})(R|!P) \to P'$ is induced by a communication within R, then P' must be of the form $(\vec{x'})(R'|P\sigma)$. On the other hand,

$$(\vec{x})(R|((!P)^\star)^!) \to (\vec{x'})(R'|((!P)^\star)^!\sigma).$$

That is

$$(\vec{x})(R|((!P)^\star)^!) \to (\vec{x'})(R'|((!(P\sigma))^\star)^!).$$

- If $(\vec{x})(R|!P) \to P'$ is caused by a communication between R and $!P$, then P' must be of the form $(\vec{x})(R'|P_{i_k}[a/y^{i_k}]|!P)$ for some a and $1 \le k \le j$. Correspondingly

$$\begin{aligned}&(\vec{x})(R|((!P)^\star)^!) \\ \to\ &(\vec{x})(R'|(\overline{(P_{i_k}[a/y^{i_k}])^\star})^!|((!P)^\star)^!).\end{aligned}$$

Clearly,

$$(\overline{(P_{i_k}[a/y^{i_k}])^\star})^! \approx ((P_{i_k}[a/y^{i_k}])^\star)^!.$$

By induction hypothesis,

$$\begin{aligned}(\overline{(P_{i_k}[a/y^{i_k}])^\star})^! &\approx ((P_{i_k}[a/y^{i_k}])^\star)^! \\ &\approx P_{i_k}[a/y^{i_k}].\end{aligned}$$

So by theorem 3.9,

$$\begin{aligned}&(\vec{x})(R'|(\overline{(P_{i_k}[a/y^{i_k}])^\star})^!|((!P)^\star)^!) \\ \approx\ &(\vec{x})(R'|P_{i_k}[a/y^{i_k}]|((!P)^\star)^!).\end{aligned}$$

It follows that

$$\begin{aligned}&(\vec{x})(R'|P_{i_k}[a/y^{i_k}]|!P) \\ \approx \mathcal{R} \approx\ &(\vec{x})(R'|(\overline{(P_{i_k}[a/y^{i_k}])^\star})^!|((!P)^\star)^!).\end{aligned}$$

So $\mathcal{R}$ is a bisimulation up to $\approx$. □

For the sake of the next proof, let's annotate the barbed bisimilarity in the calculus of guarded graphs by a subscript g.

Theorem 3.23 *(i) Suppose P and Q are terms in the calculus of guarded graphs. Then $P \approx_g Q$ if and only if $P^! \approx Q^!$.*
(ii) Suppose P and Q are terms in the calculus of graphs. Then $P \approx Q$ if and only if $P^\star \approx_g Q^\star$.

Proof: (i) We first show that if $P \approx_g Q$ then $P^! \approx Q^!$. Consider the relation

$$\mathcal{S} \stackrel{\text{def}}{=} \{(P^!, Q^!) \mid P \approx_g Q\}.$$

Suppose R is a term, $\vec{x}$ a sequence of names and $(\vec{x})(R|P^!) \to P_1$. Then $(\vec{x})(R^\star|(P^!)^\star) \to^+ P_1^\star$ by theorem 3.21. By assumption and theorem 3.22, $(P^!)^\star \approx (Q^!)^\star$. So $(\vec{x})(R^\star|(Q^!)^\star) \to^* Q_2$ for some Q_2 such that $P_1^\star \approx_g Q_2$. Hence by theorem 3.21,

$$(\vec{x})((R^\star)^!|((Q^!)^\star)^!) \to^* Q_2^!.$$

But

$$(\vec{x})((R^\star)^!|((Q^!)^\star)^!) \approx (\vec{x})(R|Q^!)$$

according to theorem 3.22. So Q_1 exists such that $Q_2^! \approx Q_1$ and $(\vec{x})(R|Q^!) \rightarrow^* Q_1$. It follows that $\mathcal{S}$ is a bisimulation up to $\approx$.
(ii) Similarly we can show that if $P \approx Q$ then $P^\star \approx_g Q^\star$.
(iii) For guarded terms P and Q, if $P^! \approx Q^!$, then by theorem 3.22 and (ii), $P \approx_g (P^!)^\star \approx_g (Q^!)^\star \approx_g Q$.
(iv) The proof is similar to (iii). □

The calculus of guarded graphs appears to have the edge over the calculus of graphs. But theorems 3.21, 3.22 and 3.23 establish the equivalence of the two languages. These results justify the decision to take reaction graphs as the basic computing agents.

3.5 Graphs as Models

Suppose we have two calculi $\mathcal{L}^i$, $i = 1, 2$. $\mathcal{L}_i$ is equipped with a reduction relation $\rightarrow_i$ and an observational equivalence $\approx_i$. The former defines the operational semantics of $\mathcal{L}_i$ while the latter characterizes its denotational semantics. A translation $\mathcal{M}$ from $\mathcal{L}_1$ to $\mathcal{L}_2$ should preserve at least the operational semantics of $\mathcal{L}_1$, meaning that the following two complementary properties are satisfied:

- if $S \rightarrow_1 T$, then $\mathcal{M}S \rightarrow_2 \mathcal{M}T$;
- if $\mathcal{M}S \rightarrow_2 T'$, then T exists such that $S \rightarrow_1 T$ and $T' \rightarrow_2^* \mathcal{M}T$

The first condition guarantees the soundness of the translation and the second ensures the faithfulness. As for the preservation of the observational equivalence, a natural requirement is

(1) $\qquad S \approx_1 T$ implies $\mathcal{M}S \approx_2 \mathcal{M}T$.

If $\mathcal{M}$ is not meant to be part of a comparison of $\mathcal{L}_1$ against $\mathcal{L}_2$, but acts as a translation that interpret $\mathcal{L}_1$ in the target language $\mathcal{L}_2$, then (1) is too strong a requirement. $\mathcal{L}_2$ might have far more discriminating power than $\mathcal{L}_1$. For example suppose $\mathcal{L}_1$ is a high level programming language and $\mathcal{L}_2$ is an assembly language. Normally we are not interested in the behaviour of two programmes $\mathcal{M}S$ and $\mathcal{M}T$ in the presence of an arbitrary piece of assembly code. All we care about is how they act in the presence of $\mathcal{M}P$ where P is a programme in $\mathcal{L}^1$. The next definition to be introduced shortly come with these considerations. We first introduce the notion of contexts in the calculus of graphs, which can be inductively defined as follows:

- $[]$ is a context;
- if $C[]$ is a context, then $C[]|P$, $(x)C[]$ are context.

Definition 3.24 *Suppose $\mathcal{C}$ is a set of contexts in the calculus of graphs and $\mathcal{R}$ is a subset of $\mathcal{T}\times\mathcal{T}$. The relation $\mathcal{R}$ is a barbed bisimulation with respect to $\mathcal{C}$ if it is barbed and whenever $P\mathcal{R}Q$ then for each $C[] \in \mathcal{C}$ it holds that*
(i) if $C[P] \rightarrow P'$, then there exists some Q' such that $C[Q] \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $C[Q] \rightarrow Q'$, then there exists some P' such that $C[P] \rightarrow^* P'$ and $P'\mathcal{R}Q'$.
The barbed bisimilarity with respect to $\mathcal{C}$, notation $\approx_{\mathcal{C}}$, is the largest barbed bisimulation with respect to $\mathcal{C}$.

When $\mathcal{C}$ includes all contexts, $\approx_{\mathcal{C}}$ coincides with $\approx$.

In next section we show how the calculus of graphs act as a target language.

4 Applying Reaction Graphs

In the end, a computational model has to be judged in practice. This section is devoted to demonstrations of the operational adequacy of reaction graphs. We first explain how to regard a multiplicative linear logic proof as a reaction graph. The graph theoretical interpretation explicates that reactions in graphs are cut eliminations in an abstract setting. It is obvious that some reactions in a reaction graph can happen in parallel. The fact that sequential computations can be modeled in reaction graphs should indicate that graphs are concurrent computational agents. We will elaborate this point by giving a translation of the mini mobile processes to reaction graphs. By composing this translation with Milner's translation of the lazy λ-calculus ([4, 1, 22]), we get immediately a translation of the latter in the calculus of graphs. A fallout of this fact is that reaction graphs possess Turing computability.

4.1 Proofs as Polyadic Graphs

Linear logic was proposed as a modification of classical logic ([11, 12, 33]). A computational treatment of the proofs of the classical linear logic is initiated in [2]. The idea is further explored in the setting of process algebras; see for example [6]. In this approach, a sequent proof is annotated with variables. A process interpreting the proof has free variables occurring in the conclusion of the proof as its free names through which it communicates with outside world. The Abramsky approach generalizes the proof-as-term paradigm to a proof-as-process paradigm. Cut eliminations are interpreted as process communications. Two things are noteworthy with this interpretation. First in order to model cut eliminations high up in a derivation tree, one needs reduction under prefixing. In [6], unguarded

prefixes are introduced to achieve that. Second the linear logic proofs are more symmetric than the π-calculus can capture. The mismatch is due essentially to the functional nature of the communications in π-calculus.

Another important point of the proof-as-process interpretation is that it tries to code up the underlying tree structures of the derivations of proofs. This is especially true for the multiplicative part of the linear logic. However in the process interpretation, the tree structures are buried in the syntax of the process calculus.

In this section we take a look at Abramsky's translation of the multiplicative linear logic in our graph theoretical setting. We will see that the graph interpretation is more symmetric and reflects more faithfully the tree structures of proofs.

There are four rules for the multiplicative linear logic, ignoring those rules for units:

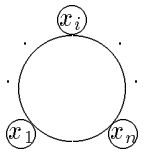
$$\frac{}{x : A, y : A^{\perp}}\, Axiom$$

$$\frac{\vdots F \qquad \vdots G}{\vec{w} : \Gamma, x : A \qquad \vec{v} : \Delta, y : B}{\vec{w} : \Gamma, \vec{v} : \Delta, z : A \otimes B}\otimes$$

$$\frac{\vdots F}{\vec{w} : \Gamma, x : A, y : B}{\vec{w} : \Gamma, z : A \wp B}\wp$$

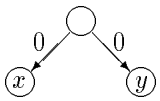
$$\frac{\vdots F \qquad \vdots G}{\vec{w} : \Gamma, x : C \qquad \vec{v} : \Delta, y : C^{\perp}}{\vec{w} : \Gamma, \vec{v} : \Delta}\, Cut$$

The basic idea of the interpretation is that a proof of the sequent $x_1 : A_1, \ldots, x_n : A_n$ is interpreted by a reaction graph with n global nodes labeled respectively by $x_1, \ldots, x_n$, as is shown below:



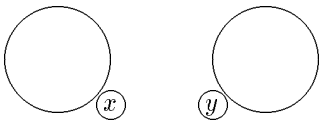
In the sequel, only relevant global nodes are displayed. The translation is inductively defined as follows:

- The axiom is interpreted by the following reaction graph:

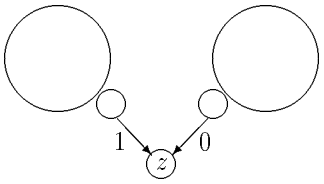


  Formally the axiom is modeled in the calculus of graphs by the term $(a)(x_0[a]|y_0[a])$. Notice the symmetric roles of x and y in this term.

- If the premises of the $\otimes$-rule are interpreted respectively as the following reaction graphs:

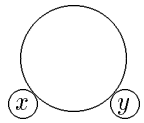


  then the conclusion of the rule is interpreted by the reaction graph:

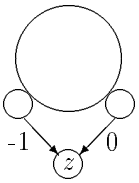


  Algebraically the interpretation is the term $(x)(y)(z_1[x]|z_0[y]|F|G)$.

- If the premise of the $\wp$-rule is interpreted by the following reaction graph:

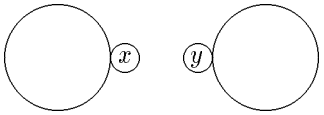


  then the conclusion of the rule is then interpreted by the reaction graph:

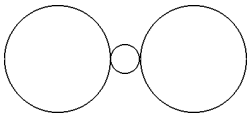


  In the calculus of graphs, it is expressed as $(x)(y)(z_{-1}[x]|z_0[y]|F)$.

- If the two premises of the cut rules are interpreted by the following reaction graphs:



  then the conclusion of the rule is interpreted by the following reaction graph:



More formally, the interpretation is the following term $(z)(F[z/x]|G[z/y])$.

This completes the definition of the interpretation.

Cut eliminations for the multiplicative linear logic correspond to reductions in reaction graphs. A precise statement of this fact is omitted. Confer [6].

4.2 Processes as Diadic Graphs

Conceptually reactions that are not causally related can happen in parallel. Meanwhile reactions can be ordered in terms of enablement and the recursion mechanism. It is in this sense reaction graph rewriting captures concurrent computations. This section explains how to think of mobile processes as reaction graphs. The processes considered here are the so-called mini π-processes ([17]):

$$P := \mathbf{0} \mid m(x).P \mid \overline{m}x.P \mid (x)P \mid P|P' \mid m(x)*P.$$

We will impose a syntactical equality on the set of all mini π-processes in the same way as we have done with the terms of the calculus of graphs.

Definition 4.1 *The structural relation $=$ is the least congruence on terms that contains:*
(i) $P|\mathbf{0} = P$, $P_1|P_2 = P_2|P_1$, *and* $P_1|(P_2|P_3) = (P_1|P_2)|P_3$;
(ii) $(x)\mathbf{0} = \mathbf{0}$, $(x)(y)P = (y)(x)P$, $(x)(P|Q) = P|(x)Q$ *if* $x \notin gn(P)$.

The operational semantics of the calculus is defined by the following two reduction rules:

$$m(x).P|\overline{m}y.Q \rightarrow P[y/x]|Q$$

$$m(x)*P|\overline{m}y.Q \rightarrow m(x)*P|P[y/x]|Q$$

and the following three structural rules:

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \frac{P \rightarrow P'}{(x)P \rightarrow (x)P'}$$

We now define a translation from the π-processes to reaction graphs. In view of the results in section 3, it suffices to define a translation from the processes to the calculus of guarded graphs:

$$\begin{aligned}
\mathbf{0}^\dagger &\stackrel{\text{def}}{=} \mathbf{0} \\
(m(x).P)^\dagger &\stackrel{\text{def}}{=} (a)(m[a]|a(x)\star(a[a]|P^\dagger)) \\
(\overline{m}x.P)^\dagger &\stackrel{\text{def}}{=} (a)(\overline{m}[a]|\overline{a}[x]|\overline{a}(b)\star Q^\dagger) \\
(P|Q)^\dagger &\stackrel{\text{def}}{=} P^\dagger|Q^\dagger \\
((x)P)^\dagger &\stackrel{\text{def}}{=} (x)P^\dagger \\
(m(x)*P)^\dagger &\stackrel{\text{def}}{=} m(x)*P^\dagger.
\end{aligned}$$

Theorem 4.2 *Suppose P is a π-process.*
(i) if $P \rightarrow Q$ then $P^\dagger \rightarrow Q^\dagger$;
(ii) if $P^\dagger \rightarrow P'$, then Q exists such that $\rightarrow Q$, $P' \rightarrow^* Q^\dagger$ and $P' \approx Q^\dagger$.

The algebraic properties of the π-calculus have been intensively studied ([30, 20, 31, 25, 26, 27, 32]). What is relevant to us in this paper is the barbed bisimilarity. Let's mention that a is an *immediate barb* of P, notation $P\downarrow a$, if

- $P \equiv a(x).P$ or $P \equiv \overline{a}x.P$;
- $P \equiv P_1|P_2$ and either $P_1\downarrow a$ or $P_2\downarrow a$;
- $P \equiv (x)P'$ and $P'\downarrow a$ and $x \neq a$.

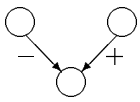
The barbed bisimulation (up to bisimilarity) can be defined for the mini π-processes in completely the same way as for the terms of the calculus of graphs. The barbed bisimilarity is preserved by all but the input prefixing operator.

Let $\mathcal{C}_\pi$ be the set of all mini π-process contexts. The graph interpretations of the mini π-processes preserve the algebraic equality in the sense of the theorem below.

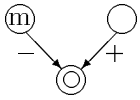
Theorem 4.3 *Let Π be the set $\{C[]^\dagger \mid C[] \in \mathcal{C}_\pi\}$. If P and Q are two mini π-processes and $P \approx Q$, then $P^\dagger \approx_\Pi Q^\dagger$.*

5 Conclusion

The operational aspects of computations encapsulated in reaction graphs are important enough to justify their introduction. This is reinforced by the simplicity of the structures of the reaction graphs. Reactions of graphs are cut elimination reincarnated in a graph theoretical setting. It is intended that reaction graphs abstract common geometric structures from processes and proofs while reactions capture the dynamic reconfigurations of the structures typically found in concurrency theory and proof theory. More specifically, a configuration of the form



in a reaction graph is a cut. The reconfiguration of the cut by reaction eliminates the cut. On the other hand, a subgraph of the shape



describes the situation in which m is ready to output through a channel which can also act as a channel for an input action. The reaction of the subgraph is a communication which instantiates the local name by m.

We have emphasized that reaction graphs are computational objects on their own. It is however also worthwhile to see them as implementations of computational objects.

Coming back to the view that reaction graphs are underlying structures of an abstract proofs, we can see that the calculus of graphs and the encodings of mini π-processes as graph terms are activities in a process-as-proof paradigm. The proof theoretical interpretation gives rise to a nonstandard understanding of mobile processes. Such an understanding demands a further stay-away from the familiar functional view of computations.

Rewriting of reaction graphs reminds one of chemical reactions ([5]), which explains our choice of terminology. What comes natural in a chemical scenario is that things happen in parallel. We can also think of the calculus of graphs as a model for parallel computations. If two reactions do not result in conflict, they might as well proceed in parallel. To simplify the operational semantics for this parallel model, it is better to ban the rules (A2) and (M2). The simplification does not decrease the expressive power of the model though. From another viewpoint, elimination of (A2) and (M2) push the reaction graphs closer to proofs. The situation draw a resemblance to that where attention is paid to a sublanguage πI of the π-calculus ([28]).

By adding the sequentiality combinator to the calculus of graphs, we get a process calculus, χ-calculus, that has been studied in [7, 8]. The syntax of the calculus also motivates a symmetric presentation of the π-calculus ([10]).

References

[1] S. Abramsky. The Lazy Lambda Calculus. *Declarative Programming*, ed. D. Turner, 65–116, Addison-Wesley, 1988.

[2] S. Abramsky. Computational Interpretations of Linear Logic. *Theoretical Computer Science*, **111**: 3–57, North Holland, 1993.

[3] S. Abramsky. Proofs as Processes. *Theoretical Computer Science*, **135**: 5–9, North-Holland, 1994.

[4] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and Foundations of Mathematics, North-Holland, 1984.

[5] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, **96**: 217–248, North Holland, 1992.

[6] G. Bellin and P. Scott. On the π-Calculus and Linear Logic. *Theoretical Computer Science*, **135**: 11–65, North-Holland, 1994.

[7] Y. Fu. The χ-Calculus. *Proceedings of the International Conference on Advances in Parallel and Distributed Computing*, 74-81, 19-21 March, Shanghai, China, IEEE Computer Society Press, 1997.

[8] Y. Fu. A Proof Theoretical Approach to Communications. *ICALP'97*, 7-11 July, Bologna, Italy, Lecture Notes in Computer Science 1256, Springer Verlag, 1997.

[9] Y. Fu. Reaction Graphs. *International Workshop on Formal Model of Programming and Its Applications*, Beijing, 17-20 September, 1997.

[10] Y. Fu. Symmetric π-Calculus. *Journal of Computer Science and Technology*, **13**: 202–208, 1998.

[11] J. Girard. Linear Logic. *Journal of Theoretical Computer Science*, **50**: 1–102, North-Holland, 1987.

[12] J. Girard. Towards a Geometry of Interaction. *Categories in Computer Science and Logic*, ed. J.W. Gray and A. Scedrov, Contemporary Mathematics, **92**, 69–108, AMS, 1989.

[13] Honda and Yoshida. On Reduction Based Process Semantics. *Theoretical Computer Science*, **152**(2):437–486, North-Holland, 1995.

[14] Y. Lafont. Interaction Nets. *POPL'90*, 95–108, ACM, 1990.

[15] Y. Lafont. Interaction Combinators. *Information and Computation*, **137**: 69–101.

[16] R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.

[17] R. Milner. Functions as Processes. *Journal of Mathematical Structures in Computer Science* **2**: 167–180, 1992.

[18] R. Milner. The Polyadic π-Calculus: a Tutorial. Technical Report, Department of Computer Science , University of Edinburgh, 1992.

[19] R. Milner. π-Nets: a Graphical Form of π-Calculus. *ESOP'94*, Lecture Notes in Computer Science 788, 26–42, Springer Verlag, 1994.

[20] R. Milner, J. Parrow and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, **100**: 1–40 (Part I), 41–77 (Part II), Academic Press, 1992.

[21] R. Milner and D. Sangiorgi. Barbed Bisimulation, *ICALP'92*, Lecture Notes in Computer Science 685–695, Springer Verlag, 1992.

[22] L. Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College of Science and Technology, University of London, 1988.

[23] D. Park. Concurrency and Automata on Infinite Sequences. Lecture Notes in Computer Science 154, 561–572, Springer Verlag, 1981.

[24] J. Parrow. Interaction Diagrams. *A Decade of Concurrency*, Lecture Notes in Computer Science 803, Springer Verlag, 1993.

[25] J. Parrow and D. Sangiorgi. Algebraic Theories for Name-Passing Calculi. *Information and Computation*, **120**, Academic Press, 1995.

[26] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1993.

[27] D. Sangiorgi. A Theory of Bisimulation for π-Calculus. *CONCUR'93*, Lecture Notes in Computer Science 715, Springer Verlag, 1993.

[28] D. Sangiorgi. πI: A Symmetric Calculus Based on Internal Mobility, *ICALP'95*, Lecture Notes in Computer Science, Springer Verlag, 1995.

[29] D. Sangiorgi and R. Milner. Techniques of "Weak Bisimulation Up To". *CONCUR'92*, Lecture Notes in Computer Science, Springer Verlag, 1992.

[30] B. Thomsen. A Calculus of Higher Order Communicating Systems. *POPL'89*, 143–154, Austin, Texas, USA, 1989.

[31] B. Thomsen. Plain CHOCS—A Second Generation Calculus for Higher Order Processes. *Acta Informatica*, **30**: 1–59, Springer Verlag, 1993.

[32] B. Thomsen. A Theory of Higher Order Communicating Systems. *Information and Computation*, **116**: 38–57, Academic Press, 1995.

[33] A. Troelstra. Lectures on Linear Logic. CSLI Lecture Notes, 1992.