Non Modularity and Expressibility for Nets of Relations (Extended Abstract)

Alexander Rabinovich
IBM Research Center
P.O. Box 218, Yorktown Heights, NY 10598
e.mail: *alik@watson.ibm.com*

1 Introduction

1.1 Motivation and Objectives

Data flow deals with nets of concurrently communicating agents. But unlike other theories of concurrency, the main concern here is about the I(nput)-O(utput) behavior of a system. The Kahn principle states that for special deterministic agents the Input-Output behavior of a net can be obtained from the I-O behaviors of its components. The I-O behaviors of Kahn agents are *sequential* functions and the I-O behavior of a net over such agents is obtained as the solution of an appropriate system of equations constructed from the functions computed by its components. One of important consequences of the Kahn principle and the properties of corresponding systems of equations is that nets over Kahn agents are *I-O-substitutive*, i.e. if a subnet N_1 of a given net N is replaced by a net with the same I-O behavior then the I-O behavior of the overall net remains unchanged. In an attempt to generalize Kahn's results [3] to *nondeterministic* agents, Brock and Ackerman observed that for such nets subtitutivity fails. This fact was such a surprise when it was first observed in [2] that it is known as Brock Ackerman anomaly.

Much research in dataflow centers around the problems raised by the pioneering works of Kahn and Brock and Ackerman [1, 6, 11, 12, 15]. In [11] we identified observable relations and nets of observable relations as appropriate tools for investigation of dataflow networks over nondeterministic agents. Observable relations are behaviors of (in general nondeterministic) dataflow agents. Moreover, semantics of nets of observable relations is the unique semantics which is consistent with the least fixed point solution of system of equations and with the input-output behavior of dataflow agents. But it turns out that this semantics is not modular, i.e. there are two nets of observable relations N_1, N_2 with the same meaning and a context $Con[\,]$ such that $Con[N_1]$ and $Con[N_2]$ have different meanings.

In [11, 12] we showed that the main source of the Brock-Ackerman anomaly is in the semantics of nets of relations. If one considers nets over a subclass of observable relations, it may happen that semantics over such nets is modular. We say that a subclass C of relations is **modular** if nets over relations in C with the same meaning are replaceable by each other in any context over relations in C. It appears that in nontrivial cases the following two tasks are reducible to each other [13]:

- Find a class of dataflow agents which are substitutive.
- Find a modular class of relations.

The central objective of this paper is the characterization of modular classes of relations and hence indirectly of I-O-substitutivity for dataflow agents. Another major theme which plays a technical role in this characterization, but is interesting in its own is expressibility for relational nets. The investigation also reveals the interesting role played by *stable* functions.

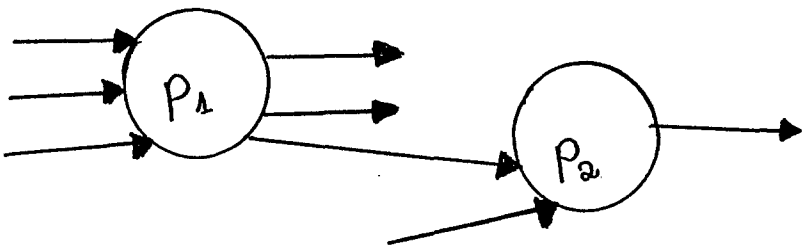

Figure 1: A net N

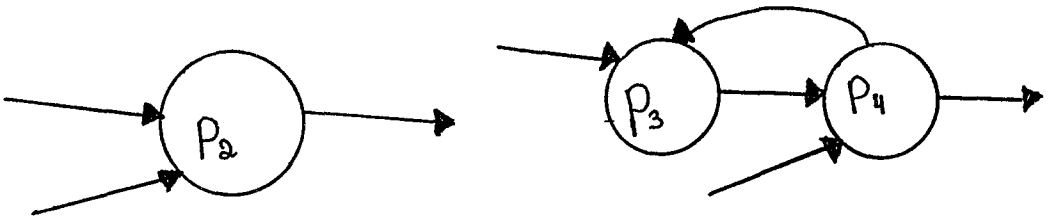

Figure 2: Nets N_1 and N_2

1.2 Nets of Relations

For simplicity we consider only finite nets. Let N be a net; its arcs (channels) are divided into input, output and internal channels.

Fig. 1 suggests itself. p_1, p_2 are names for nodes of the appropriate types. For example, p_1 has three input channels (all of them are input channels of the net) and three output channels (two of them are output channels of the net). Actually, such a net is a piece of syntax and one can impose a semantics on a net via an appropriate interpretation of the nodes. In a relational net the nodes of the net are interpreted as an input-output relation between values on the input and output channels of the node. A semantics assigns to such a net a relation between values on input and output channels of the net; the values on internal channels are hidden, although in computation of a global relation one uses values on internal channels. The definition of this semantics and its justification will be given in an appendix. However, few remarks about modularity of net semantics are in order here.

Consider nets N_1 and N_2 in Fig. 2. They have the same type, i.e., the same sets of input and output channels. N_1 is a subnet of the net N in Fig. 1. By replacing N_1 by N_2 in N we obtain the net N' in Fig. 3.

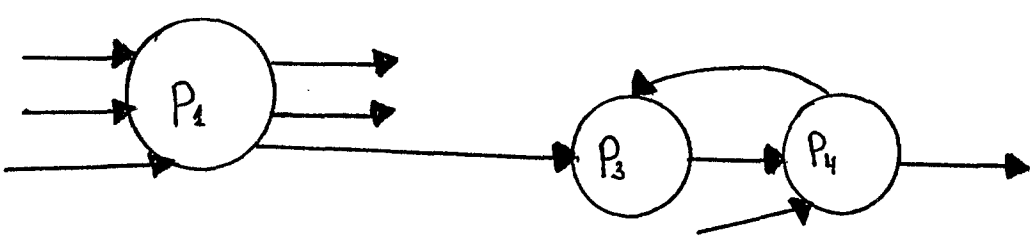

Figure 3: $N' = N[N_2/N_1]$

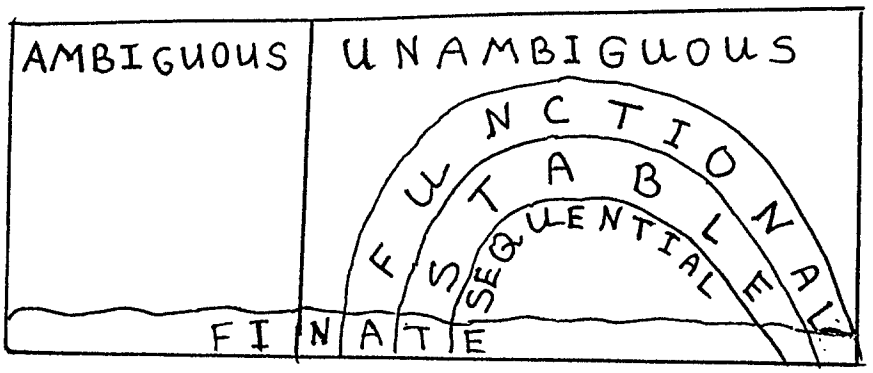

Figure 4: Classification of Relations

Definition 1: *A semantics is called* modular *iff two nets with the same meaning are replaceable by each other in every context.*

Modularity of relational semantics would imply that if N_1 and N_2 specify equal input-output relations, then N and N' specify equal relations. As we noted earlier, the semantics for relational nets is not modular. Modularity reflects Frege principle and is a crucial property of semantics, therefore, two alternatives are possible:

1. Relations are a wrong structure to apply net composition to [9], therefore, relational semantics is of a little interest.

2. Relations are widely used in programming. The source of Brock-Ackerman anomaly is in the semantics of relational nets. It is important to find out for which classes of relations the semantics is modular.

We explore in this paper the second alternative.

1.3 Overview of Results

We consider relations over stream domains and classify them into functional and nonfunctional. A continuous function f is represented by its approximating relation R_f; $R_f(x; y) \leftrightarrow_{def} f(x) \geq y$. Functions also are subdivided into different classes: Sequential $\subset$ Stable $\subset$ Continuous. This classification is standard and precise definitions will be given later. Relations are classified into ambiguous and unambiguous. Roughly speaking, R is an ambiguous relation if consistent (in the domain) inputs may produce inconsistent outputs. All the other relations are unambiguous. The simplest ambiguous relation is a relation without inputs and with one output channel which may produce on the output either 0 or 1. All functional relations are unambiguous but there are unambiguous nonfunctional relations. For example, let $Unam$ be the relation without inputs and two output channels such that 1 may be produced on the first or on the second channel but not on both. R is unambiguous but nonfunctional. It is the 'simplest' nonfunctional relation. Another interesting subclass is a subclass of finite relations. Classification of relations is given in Fig. 4. We will deal only with classes of relations which have the following properties: First, a class should have a basic computational power. This amounts to the requirement that a class contains all finite sequential functions. Second, we require that a class is closed under net construct, i.e., any net over relations in the class specifies a relation in the class.

We are now ready to state some results of the paper.

1. The class of all functional relations is modular. It is a maximal modular class, but there are other maximal modular classes.

2. No modular class can contain an ambiguous relation.

3. No modular class can contain a nonfunctional relation and an unstable function.

To contrast these results let us note that the class generated by the stable functions and the relation *Unam* is modular [10].

In the sequel the exposition is organized as follows:

Section 2 provides a classification of relations. In section 3 we state expressibility results; we find the 'simplest' relations in different classes. The expressibility issues are a clasical topic of recursion theory. Panangaden et. al. [8] were first to promote the topic for dataflow.

In section 4 we demonstrate that classes which contain some simple relations cannot be modular. Together with expressibility results of sections 3 it gives general theorems about the failure of modularity.

Section 5 discusses some open problems and further work. The definitions of the semantics of nets of relations is provided in the appendix.

2 Relations

In this section we introduce basic notions. In section 2.2 observable relations are defined and their connection with multivalued functions is explained. Section 2.3 provides many examples and our classification of relations.

2.1 Stream Domains

Definition 2: *Let Δ be an arbitrary set. The stream domain $Stream(\Delta)$ over Δ consists of all finite and infinite strings over Δ, including the empty string and is partially ordered by the relation 'x is a prefix of y'.*

Obviously the set of streams ordered as above is a CPO. We use $\sqcup$ for the least upper bound and $\sqcap$ for the greatest lower bound. We say that x, y are **consistent** elements if they have an upper bound.

Let D be a CPO. Recall that an element x of D is called **finite** if it satisfies the following condition: assume that $x \leq a$, where a is the least upper bound (lub) of a sequence $a_1 \leq a_2 \leq \ldots$; then $x \leq a_n$ for some n.

In the sequel D, D_1, D_2 range over the stream domain or Catresian products of stream domains. But most results hold for more general CPO.

2.2 Observable and Approximating Relations

Definition 3: *[11] Let D_1, D_2 be domains. A relation R between elements of D_1 and D_2 is said to be an observable I(nput)-O(utput) relation from D_1 to D_2 iff*

1. *R may hold only for finite elements*

2. *For every finite $x \in D_1$ there exists $y \in D_2$ such that $R(x;\ y)$.*

3. *R increases on inputs and decreases on outputs, i.e. for finite x, y, x_1, y_1*
   $R(x;\ y)$, $x \leq x_1$, $y_1 \leq y$ imply $R(x_1;\ y_1)$

Definition 4: *For a continuous function $f : D_1 \to D_2$ its* approximating relation R_f *is defined as* $R_f(x;\ y) =_{def} f(x) \geq y$ *and x, y are finite. A relation R is an* approximating *(or functional) relation if it is R_f for some f.*

It is easy to see that approximating relations are observable relations and that $f = g$ iff $R_f = R_g$.

Terminology. The terms 'approximating relations' and 'functional relations' are used as synonyms in the sequel.

Comments (1) Observable relations play the same role for multivalued functions as approximating relations for continuous functions. More exactly: there is an isomorphism between observable relations from D_1 to D_2 and continuous functions from D_1 into Hoare power domain of D_2. (2) We did not consider processes in this paper. But let us mention that the class of observable relations is the class of relations implemented by input-output linear processes [11].

Observable relations are ordered by inclusion. Under this ordering the set of observable relations from D_1 to D_2 is a CPO.

2.3 Classification of Observable Relations

Definition 5: *A relation R is* unambiguous *if the following holds: x and x_1 are consistent, $R(x\ ; y)$ and $R(x_1;\ y_1)$ imply that y and y_1 are consistent. A relation is* ambiguous *if it is not unambiguous.*

Clearly, every approximating (functional) relation (see definition 4) is unambiguous, but the opposite is wrong (see examples below).

Example 1: *An approximating relation for the identity function $\lambda x.x$ is the relation $\geq (x;\ y)$ which holds iff $x \geq y$ and x, y are finite.*

Let $Choice_2(;x)$ be the relation from $Stream^0$ to $Stream$ such that only $Choice_2(;\perp)$, $Choice_2(;0)$, $Choice_2(;1)$ hold. It is an ambiguous relation. $Choice_\omega(;x)$ is defined in a similar way. It holds if $x = \perp$ or x is any natural number. Clear $Choice_\omega$ is an ambiguous relation.

$Unam(;x,\ y)$ is a relation from $Stream^0$ to $Srteam^2$. It holds when $x = y = \perp$ or $x = \perp$ and $y = 1$ or $x = 1$ and $y = \perp$. It is an unambiguous, but not a functional relation.

We classify approximating relations according to the properties of their corresponding functions.

Definition 6: *Let a and b be elements of a CPO. The step function $a \to b$ is defined by*

$$(a \to b)x = \begin{cases} b & \text{if } a \leq x \\ \perp & \text{otherwise} \end{cases}$$

Note that $a \to b$ is a finite iff a and b are finite elements. We say that a relation is **finite** if it is the least upper bound of a finite set of finite step functions. The finite relations are the finite elements in the domain of observable relations. $Choice_2$ and $Unam$ are finite relations, $Choice_\omega$ is not finite. The relation $\geq$ is functional but not finite.

Assume that $f(x_1,\ x_2, \ldots x_n) = < y_1,\ y_2, \ldots y_m >$. A function f is **sequential at** $\vec{x}$ if for every j there is i such that y_j cannot be increased without increasing x_i. Formally, let $\vec{z} = < z_1,\ z_2, \ldots z_n >$ be greater than $< x_1,\ x_2, \ldots x_n >$. Assume that $f(\vec{z}) = < v_1,\ v_2, \ldots v_n >$. Then either $z_i > x_i$ or $v_j = y_j$. A function f is **sequential** if it is sequential at every $\vec{x}$ [4].

Clearly, every step function is a sequential function. The definition of sequentiality above is not preserved under domain isomorphisms. It is only well defined for *concrete* domains (these are domains which have notions of argument places [4, 17]). Stable functions were introduced by Berry and are a generalization of sequential functions and the notion of stability is invariant under domain isomorphisms.

Definition 7: *A function f is stable iff $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$ for consistent x_1, x_2.*

Every sequential function is stable, but there are nonsequential stable functions.

Example 2: *$Par : Stream^2 \rightarrow Stream$ is defined as follows:*

$$Par(x,y) = \begin{cases} 1 & \text{if } x \geq 1 \\ 1 & \text{if } y \geq 1 \\ \bot & \text{otherwise} \end{cases}$$

Par is not a stable function, because $Par(\bot,\ 1) = Par(1,\ \bot) = 1 = Par(\bot,\ 1) \sqcap Par(1,\ \bot) \neq \bot = Par(< \bot, 1 > \sqcap < 1,\ \bot >)$. Par is finite; it is the least upper bound of $< 1,\ \bot > \rightarrow 1$ and $< \bot,\ 1 > \rightarrow 1$.

The relation $FAIL$ given in the following example plays an important role for our nonmodularity results.

Example 3: *Let $FAIL$ be a relation from $Stream^2$ to $Stream^3$ which is defined by the table below as follows: if a tuple $< a_1,\ a_2,\ b_1,\ b_2,\ b_3 >$ appears in the table, and $a_i' \geq a_i$ and $b_j' \geq b_j$ then $FAIL(a_1',\ a_2';\ b_1',\ b_2,\ b_3')$ holds.*

Input		*Output*		
x_1	x_2	y_1	y_2	y_3
$\bot$	$\bot$	$\bot$	1	$\bot$
$\bot$	1	1	1	$\bot$
1	$\bot$	1	$\bot$	1

It is clear that $FAIL$ is a finite unambiguous relation. $FAIL$ is not an approximating relation, since $FAIL(1, \bot;\ 1, \bot, 1)$ and $FAIL(1, \bot;\ \bot, 1, \bot)$, but not $FAIL(1, \bot;\ 1, 1, 1)$.

3 Expressibility

A relational net is a net together with an environment which maps the nodes of the net into observable relations. The semantics of nets of relations is defined in the appendix. We say that a net of relations N *specifies* a relation R if R is the relation assigned to N by the semantics.

For a set of relations S and a relation R we denote by $Exp(S; R)$ the set of relations which are specified by nets over $S \cup R$; R' is weakly S-expressible from R iff $R' \in Exp(S; R)$. We say that R' is strongly S-expressible (or just S-expressible) from R iff there is a net N over relations in $R \cup S$ such that for every context $C[\,]$ the nets $C[R']$ and $C[N]$ specify the same relations.

Strong S-expressibility defines preorder $\leq_S$ on relations. $R' \leq_S R$ if R' is strongly S- expressible from R. We say that R and R' have the same S-degree iff $R \leq_S R'$ and $R' \leq_S R$. Since the semantics is not modular, weak S-expressibility does not define preorder. It may happen that R' is weakly S-expressible from R and R'' is weakly S-expressible from R' but R'' is not weakly S-expressible from R.

When S is the class of approximating relations of finite sequential functions we will say 'expressibility' instead 'S-expressibility', 'degree' instead 'S-degree', etc..

Claim 3.1: *(Minimal degrees).*

1. *$Choice_2$ is expressible from any ambiguous relation. It is the minimal degree among ambiguous relations.*

2. *Unam is expressible from any unambiguous nonfunctional relation. It is the minimal degree among unambiguous nonfunctional relations.*

3. *Par is expressible from any unstable functional relation. It is the minimal degree among unstable functional relations. Par is not expressible from Unam.*

We will use the following lemmas in the next section.

Lemma 3.2: *$Choice_2$ has the maximal degrees among the finite relations.*

Lemma 3.3: *The degree of $FAIL$ is less than the least upper bound of the degrees of Par and Unam.*

4 Non Modular Classes

In the appendix A5 we construct two nets N_1 and N_2 over relation $FAIL$ (see Example 3), approximating relations of finite sequential functions and a context $Con[\ ]$ over approximating relations of finite sequential functions such that N_1 and N_2 specify the same relation but $Con[N_1]$ and $Con[N_2]$ specify different relations.

Recall that a set C of relations is called *class* if it has the following properties:

1. C has a basic computational power. This amounts to the requirement that it contains all the approximating relations of the finite sequential functions.

2. C is closed under net construct, i.e., any net over relations in the class specifies a relation in the class.

Therefore it follows

Lemma 4.1: *No modular class contains relation $FAIL$.*

Recall that $FAIL$ is a finite relation, therefore, by lemma 3.2 it is expressible from $Choice_2$. Hence by claim 3.1 (1) we get

Theorem 4.2: *No modular class contains an ambiguous relation.*

Par is the minimal degree among unstable functions, *Unam* is the minimal degree among nonfunctional relations. Hence by 3.3 and 4.1

Theorem 4.3: *No modular class contains both an unstable function and a nonfunctional relation.*

Corollary 4.4: *If a modular class contains an unstable function, then it is a subclass of functional relations.*

Since the class of functional relation is modular [11] it follows

Theorem 4.5: *The class of functional relations is a maximal modular class.*

The class of stable functions plays an unexpected role when the above results are contrasted with the following theorem which will be proved in [10].

Theorem 4.6: *The class* generated *by the stable functions and Unam is modular.*

5 Conclusion and Further Work

5.1 The Structure of Degrees

It is interesting to investigate the structures of degrees. It is clear that for any two relations there exists the least upper bound of their degrees. One can also show

Claim 5.1: *$Choice_2$, Unam and $\perp$ are the only degrees for the finite relations without inputs.*

As in the case of degrees of parallelism [14, 16] there are infinite decreasing chains of degrees. Many questions about the structure of degrees have not yet been investigated. For example, we do not know whether there are infinite increasing chains of degrees. We considered degrees wrt finite sequential functions. The structures of degrees wrt the sequential functions, the stable functions and all functions are also important.

5.2 Non Modular Classes

We proved that the specific relation $FAIL$ cannot belong to any modular class. We have the following **Conjecture:** There is a finite set of relations $R_1 \ldots R_k$ such that class C is modular iff it does not contain any of R_i.

5.3 Reasoning

The reasoning about relational nets has to be further developed. We are working on this subject and have already made some initial steps. Some laws and the connection between relational semantics and conjunction of relations, the least fixed point operator on functions and synchronization of concurrent processes can be found in [5, 12, 13].

5.4 From Streams to more General CPOs

Though the theory which we are investigating is inspired by (and mainly developed for) stream processing, there is no reason to focus on specific stream domains. A generalization of the results in the paper for more general CPOs is given in [10].

Acknowledgments

I am very grateful to B. A. Trakhtenbrot for his encouragements while writing this paper; its contents were very much influenced by his ideas.

References

[1] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In *Mathematical Foundations of Programming Languages Semantics*, volume 442 of *LNCS*, Springer Verlag, 1990.

[2] J. D. Brock and W. B. Ackerman. Scenarios: A model of non-determinate computation. In *Formalization of Programming Concepts*, volume 107 of *LNCS*, Springer Verlag, 1981.

[3] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*. North Holland Publ. Co., 1974.

[4] G. Kahn and G. Plotkin, Concrete Domains, Technical Report 1978.

[5] A. Mazurkiewicz, A. Rabinovich, and B. A. Trakhtenbrot. Connectedness and synchronization. In Images of Programming. North Holland Publ. Co., 1991.

[6] J. Misra. Equational reasoning about nondeterministic processes. In *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, 1989.

[7] D. Park. The fairness problem and nondeterministic computing networks. In *Proceedings, 4th Advanced Cource on Theoretical Computer Science.* Mathematisch Centrum, 1982.

[8] P. Panangaden, and V. Shanbhogue Mccarthy's amb cannot implement fair merge. In *Proceedings, 8th FSTTSC Conference,* 1988.

[9] V. R. Pratt. e.mail correspondence, 1988.

[10] A. Rabinovich, Modularity and Expressibility for Nets of Relations. To appear in *An International Symposium on Theoretical Computer Science in honor of B. A. Trakhtenbrot*

[11] A. Rabinovich and B. A. Trakhtenbrot. Nets and data flow interpreters. In the Proceedings of the Fourth Symposium on Logic in Computer Science, 1989.

[12] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In ICALP90, volume 443 of LNCS. Springer Verlag, 1990.

[13] A. Rabinovich and B. A. Trakhtenbrot. On Nets, Algebras and Modularity. In *International Conference on Theoretical Aspects of Computer Software*, volume 526 of *LNCS.* Springer Verlag, 1991.

[14] V. Yu. Sazonov, Expressibility of Functions in Scott's LCF Language, Algebra and Logic, Vol. 15, No 3, 1976.

[15] E. W. Stark. A simple generalization of Kahn's principle to indeterminate dataflow networks. In *Semantics for concurrency*, Workshops in Computing. Springer Verlag, 1990.

[16] M. B. Trakhtenbrot, Relationship between Classes of Monotonic Functions, Theoretical Computer Science, 1976.

[17] G. Winskel, Events in Computations, Ph. D. Thesis, CS Dept, Edinburgh Univ., 1980.

A. Appendix

The appendix contains background material about nets. In section A.1 we consider nets as syntactical objects. Section A.3 defines nets of functions as a solution of the corresponding systems of equations. In section A.4 semantics of nets of observable relations is defined as a natural extension of semantics of nets of functions. This material is based on [11, 12]. Section A.5 shows that no modular class contains relation $FAIL$.

A.1 Nets as Syntax

A net is a bipartite directed graph with nodes of two kinds, pictured as circles and boxes which are called respectively places and ports (see Fig. 5). The edges of a net are called channels. If there is a channel from port p to place pl then p is called an input port of pl. If there is a channel to port p from place pl then p is called an output port of pl. The type of place pl is the pair $< P; Q >$ where P is the set of input ports of pl and Q is the set of output ports of pl.

The ports of a net are partitioned into *input* (ports with no entering channel), *output* (ports with no exiting channel) and *internal* (all the other ports). The type of a net is the pair $< I; O >$ where I

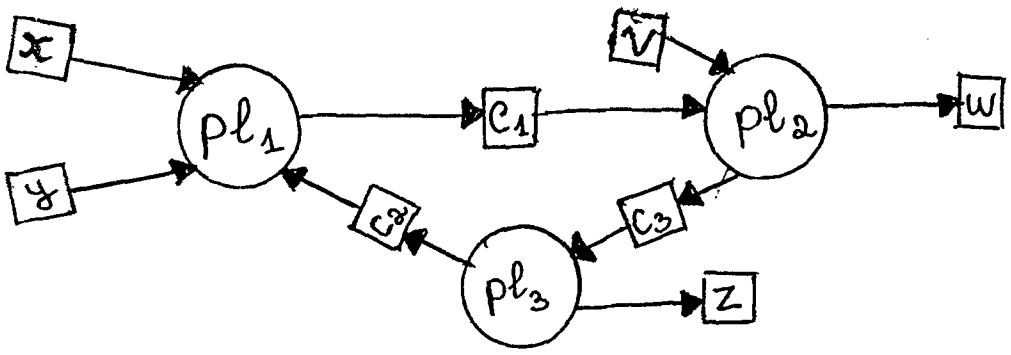

Figure 5:

is the set of input ports and O is the set of output ports of the net. In dataflow there is a further restriction on the topology of nets: each port has at most one incoming and at most one outcoming edge. For simplicity, we consider only nets which satisfy such a restriction. In this case one can use the following shorthand for nets: delete all ports and assign their names to channels which touch them. These shorthands were used in the previous figures.

A.2 Notations

Let D be a set of data and P be a set of (ports) names. An element x of D^P with $P = \{p_1, ...p_k\}$ is naturally coerced to a tuple $< x_{p_1}, ..., x_{p_k} >$ with elements of D indexed by the corresponding port names. Accordingly, a relation $R \subset D^{\{p_1,...,p_k\}}$ is uniquely associated with a relation $r \subset D^k$ such that $\eta \in R \leftrightarrow r(\eta(p_1), ...\eta(p_k))$; the same remarks hold for the mutual coercion between a port function $F : D^P \to D^Q$ and the corresponding function $f : D^k \to D^n$. In the sequel we rely often on these coercions without mentioning them explicitly. Through this abuse of notations we avoid cumbersome expressions, hopefully, in a harmless way. Let P, Q be sets of ports. We say that a function f is a function of *type* $< P; Q >$ if it maps D^P into D^Q; we say that a relation is of type $< P; Q >$ if it is an observable relation from D^P to D^Q.

A.3 Nets of Functions

We recall definitions from [12]. A net of functions is a pair $< N, \phi >$ where N is a net and ϕ a functional environment which maps places of N into continuous functions of the same type. We define below the semantics $FUN(N, \phi)$ as the solution of an appropriate system of equations $\Sigma(N, \phi)$. This definition is quite standard but note the *hiding of internal ports* in Step 3 below.

We associate a function $f : D^P \to D^{\{q_1,...,q_m\}}$ with the set of m functions $f_{q_i} : D^P \to D^{q_i}$ $i = 1, ...m$.

Step 1. Constructing the system $\Sigma(N, \phi)$. To every local (i.e., internal or output) port q there corresponds an equation in the system $\Sigma(N, \phi)$. Let q be a local port. Assume that the *unique* place pl from which there is a channel to q is mapped to f. Let $p_1, ..., p_k$ be input ports of pl. Then $x_q = f_q(x_{p_1}, ...x_{p_k})$ is the equation which corresponds to the port q.

Step 2. Taking the minimal solution of $\Sigma(N, \phi)$ *for fixed inputs.*

In this way, abstracting from the inputs, one defines in a routine way a function F of type $D^{Input} \to D^{Local}$.

Step 3. Hiding internals. Since the semantics of the net should have the same type as the net we hide in F internal ports and obtain the functional G of type $D^{Input} \to D^{Output}$, which by definition is $FUN(N, \phi)$.

Example 4: *Let ϕ assigns f^1, f^2, f^3 to the places pl_1, pl_2, pl_3 of the net in Fig. 5. Below is the corresponding system of equations:*

$$\begin{cases} c_1 = f^1(x,\ y,\ c_2) \\ c_2 = f^3_{c_2}(c_3) \\ c_3 = f^2_{c_2}(c_1,\ v) \\ z = f^3_z(c_3) \\ w = f^2_w(c_1,\ v) \end{cases}$$

A.4 Semantics for Nets of Relations

In this section we define semantics for nets of relations in an axiomatic way.

A net of relations is a pair $< N,\ \rho >$ where N is a net and ρ is a relational environment which maps places of N into observable relations of the same type. We say that a relational net $< N,\ \rho >$ *approximates* a functional net $< N,\ \phi >$ if for every place pl of N the relation assigned to pl by ρ is the approximating relation of the function assigned to pl by ϕ.

Relational semantics is a mapping *sem* from nets of relations into observable relations. Let us formulate some properties that are expected from relational semantics.

1. *sem respects type,* i.e, for a net N with input ports $\vec{I}$ and output ports $\vec{O}$ and for every relational environment ρ semantics should assign to $< N,\ \rho >$ an observable relation of type $< \vec{I};\ \vec{O} >$.
2. *sem is continuous on the CPO of observable relations,* i.e. $\lambda\rho.sem(N,\ \rho)$ is continuous for every N.
3. *sem is consistent with semantics of nets of functions,* i.e. if $< N,\ \rho >$ approximates $< N,\ \phi >$ then $sem(N,\ \rho)$ is the approximating relation of the function $FUN(N,\ \phi)$.

Theorem A.1: *[12]. Among mappings which satisfy (1)-(3) there exists the minimal mapping.*

Definition 8: *Relational semantics is the minimal mapping which satisfies (1)-(3) above.*

A.5 No Momular Class Contains Relation *FAIL*

In this section we show that no modular class contains relation $FAIL$ from example 3 (section 2). To demonstrate this fact, we consider net N_1 in Fig. 6 and find the relation R' which is specified by it; N_2 (see Fig 6) is the net with one place to which the environment assigns this very relation R'. Then we substitute N_1 and N_2 in the context $Con[\ ]$ in Fig. 7 and show that the relations assigned to the nets $Con[N_1]$ and $Con[N_2]$ are different observable relations. Therefore, modularity fails for a class which contains $FAIL$.

Below we compute the relations specified by the nets N_1, $Con[N_1]$ and $Con[N_2]$.

Definition 8 provides an axiomatic characterization of semantics for nets of relations. In order to find the relations assigned by the semantics to the nets in Fig 6 and in Fig 7 we need a more constructive definition of semantics for nets of relations which is provided in theorem A.2 below.

First, let us introduce some notations. Recall that we write $R_1 \leq R_2$ if relation R_1 is a subset of relation R_2. We also denote by $\leq$ the extensional order of the continuous functions. For a function f and an observable relation R we write $f \leq R$ if the approximating relation R_f of f is a subset of R. Similarly, for a functional environment ϕ and a relational environment ρ we write $\phi \leq \rho$ if $\phi(pl) \leq \rho(pl)$ for every place pl. We use $REL(N, \rho)$ $(FUN(N, \phi))$ for the relation (function) assigned by the relational (functional) semantics to the net of relations (functions) $< N,\ \rho >$ $(< N,\ \phi >)$. Using the above notation we state

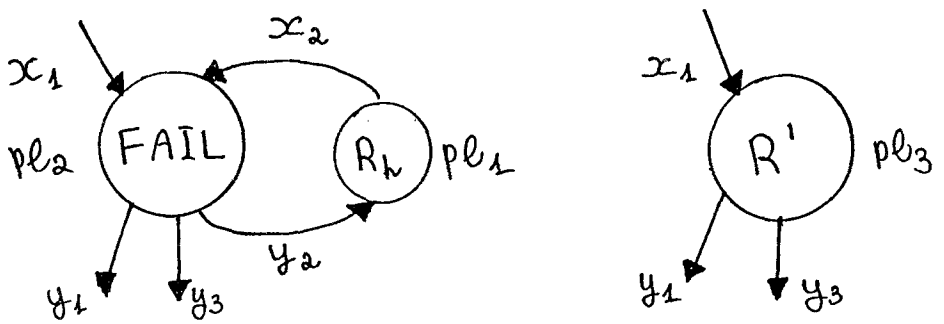

Figure 6: Nets N_1, N_2; the relation R_h is the approximating relation of $1 \rightarrow 1$

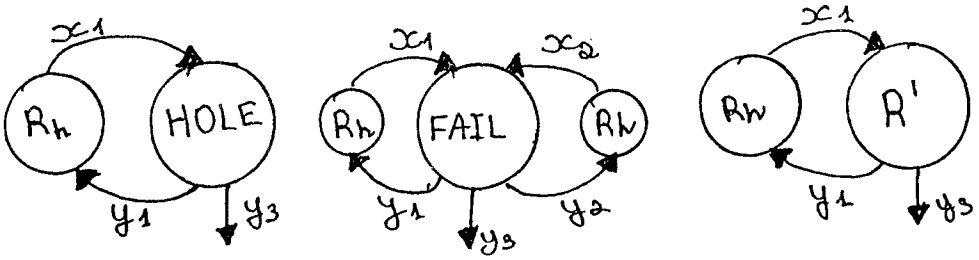

Figure 7: $Con[\,]$ and nets $Con[N_1]$, $Con[N_2]$.

Theorem A.2: $REL(N, \rho) = \cup\{R_g : g = FUN(N, \phi)$ *and* $\phi \leq \rho\}$.

Let h_1 and h_2 be functions from $STREAM^{\{x_1, x_2\}}$ into $STREAM^{\{y_1, y_2, y_3\}}$ defined as follows:

$$h_1 =< \perp, \perp >\rightarrow< \perp, 1, \perp > \sqcup < \perp, 1 >\rightarrow< 1, 1, \perp >$$
$$h_2 =< 1, \perp >\rightarrow< 1, \perp, 1 >$$

Relation $FAIL$ from example 3 (section 2) is the least upper bound of the approximating relations for h_1 and h_2. Moreover, one can prove

Fact A.3: *If* f *is a function and* $f \leq FAIL$ *then* $f \leq h_1$ *or* $f \leq h_2$.

Let us first find the relation $R'(x_1; y_1, y_3)$ which is specified by net N_1 in Fig 6. (1) From monotonicity of semantics, theorem A.2 and fact A.3 it follows that $R' = R_{g_1} \cup R_{g_2}$, where $g_1 = FUN(N_1, \phi_1)$, $g_2 = FUN(N_1, \phi_2)$, both ϕ_1 and ϕ_2 assign function $h = 1 \rightarrow 1$ to the place pl_1 of N_1, and ϕ_1 assigns h_1 to the place pl_2 and ϕ_2 assigns h_2 to the place pl_2. (2) Using the definitions of semantics for nets of functions (section A3) one can find that $g_1 = \perp \rightarrow< 1, \perp >$ and $g_2 = 1 \rightarrow< 1, 1 >$. (3) Note that $R_{g_1} \cup R_{g_2}$ is the approximating relation of the function $m = \perp \rightarrow< 1, \perp > \sqcup 1 \rightarrow< 1, 1 >$. Hence R' is the approximating relation of m.

Similar arguments show that net $Con[N_1]$ specifies relation $Q_1(; y_3)$ which holds iff $y_3 = \perp$.

Note that the environment assigns the approximating relations to all places of net $Con[N_2]$. The corresponding net of functions specifies constant function $l() = 1$. Therefore, the relation $Q_2(; y_3)$ which is specified by $Con[N_2]$ is the approximating relation of l. Hence $Q_2(; y_3)$ iff $y_3 = 1$ or $y_3 = \perp$.

To summarize, our example shows that the semantics assigns to N_1 and N_2 the same relations. However, the relations assigned to nets $Con[N_1]$ and $Con[N_2]$ are different. The relations used in the above example are the approximating relations of finite sequential functions and relation $FAIL$. Therefore, no modular class can contain $FAIL$.