Modularity and Expressibility for Nets of Relations

Alexander Rabinovich
Department of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University
Tel Aviv, Israel 69978

Abstract

Modularity reflects the Frege Principle: any two expressions $expr_1$ and $expr_2$ which have the same meaning (semantics) can be replaced by each other in every appropriate context $C[\ ]$ without changing the meaning of the overall expression.

In [18] we identified observable relations and nets of observable relations as appropriate tools for the investigation of dataflow networks over nondeterministic agents. The observable relations are the Input-Output behaviors of (in general nondeterministic) dataflow agents. Moreover, the semantics of nets of observable relations is consistent with the input-output behavior of dataflow agents.

In [18, 19] we showed that the main source of the Brock-Ackerman anomaly [2] is in the semantics of nets of relations. But it turns out that this semantics is not modular.

The central objective of this paper is the characterization of modular classes of relations and hence indirectly the set of dataflow nets without anomalies. Another major theme which plays a technical role in this characterization, but is interesting in its own, is the expressibility for relational nets. The investigation also reveals the interesting role played by *stable* functions introduced by Berry [3].

Contents

1. Introduction

1.1. Modularity, Substitutivity, Compositionality

Modularity reflects the Frege Principle: any two expressions $expr_1$ and $expr_2$ which have the same meaning (semantics) can be replaced by each other in every appropriate context $C[\ ]$ without changing the meaning of the overall expression:

$$sem(expr_1) = sem(expr_2) \text{ implies } sem(C[expr_1]) = sem(C[expr_2])$$

$R-$*substitutivity*, where R is a given a binary relation R in the semantical domain of meanings, is a broader notion. It means that

$$sem(expr_1) R sem(expr_2) \text{ implies } sem(C[expr_1]) R sem(C[expr_2])$$

In particular, R may happen to be an equivalence relation in the semantical domain. For example, a dataflow net may specify a process Pr but one is mainly interested in the input-output behavior $rel(Pr)$ of this process i.e., the relation between the input histories and output histories of Pr. Hence $\equiv_{rel}$-substitutivity (substitutivity of the equivalence $rel(Pr_1) = rel(Pr_2)$) is of fundamental importance. However, in general, R is not necessarily an equivalence relation. As a matter of fact for dataflow nets we consider also the substitutivity of $\leq_{rel}$ i.e. of $rel(Pr_1) \subseteq rel(Pr_2)$.

A conventional syntax (call it TEXTUAL as opposed to a graphical syntax of nets) is based on a signature Σ. Moreover, a complex piece of syntax $expr$ may be *uniquely* decomposed into simpler sub-pieces: $expr = op(expr_1, \cdots expr_k)$, where op is in Σ. This fact is usually referred to as the unique decomposition theorem. If *compositional* semantics is used, then there is a corresponding semantical clause with the format: $sem(expr) =_{def} OP(sem(expr_1), ..., sem(expr_k))$. Here OP is the semantical constructor which corresponds to *op*. In this situation there is a natural and clear notion of context; it is also quite evident that a compositional semantics guarantees modularity. Compositional semantics may be characterized as a homomorphism from the free Σ-algebra of the syntactical domain into the Σ-algebra of the semantical domain.

Typically, a denotational semantics is formulated in a compositional style and hence supports modularity. However, often one starts with an operational semantics which lacks a compositional structure. Then a standard way to prove modularity is to discover a compositional semantics which is equivalent to the given operational one.

In net models of concurrency a syntax is provided by some specific class NN of labeled graphs called *nets* and a semantics is usually defined globally either in an operational style through appropriate firing (enabling) rules or as a solution of a system of equations. For example, the semantics of Petri Nets is defined by the rules which specify the set of enabled transitions and the effect of an execution of an enabled transition.

For a graphical syntax of nets the notions of *context, subnet* and *substitution* make sense and therefore $R-$ *substitutivity* (in particular modularity) may be defined and investigated. Note that the modularity issues for nets can be investigated without introducing any textual syntax for descriptions of nets.

Sometimes operations on nets are defined. Broy [4] considered operations loop and aggregation on nets. He used a textual language NET for describing nets. Different terms of this textual language may describe the same net. Terms t_1 and t_2 of a textual language NET are *graph equivalent* if they describe isomorphic nets. From a compositional semantics for this textual language NET a semantics for nets as graphs can defined in the following way: for every net N choose a term that describes it and call it the standard description of N. The semantics of a net is defined as the semantics of its standard description. The semantics defined in such a way is not necessarily modular. A sufficient condition for its modularity is the requirement that graph equivalent terms of NET will have the same semantics.

1.2. Historical Background

Petri nets and dataflow nets are fundamental paradigms in concurrency. Historically, modularity topics appeared for these paradigms as follows:

1. *Dataflow.* Substitutivity issues for dataflow nets were identified early and in a sharp way. Unlike other theories of concurrency, the main concern in dataflow is 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. The Kahn Principle [7] implies that dataflow nets over deterministic agents are $\equiv_{rel}$-substitutive. In an attempt to generalize Kahn's result to nondeterministic agents, Brock and Ackerman observed in [2] that, $\equiv_{rel}$-substitutivity fails. The celebrated counterexample from [2] illustrates this so called Brock-Ackerman anomaly.

2. *Petri Nets.* For elementary Petri nets modularity was established by Mazurkiewicz [9] following the pattern mentioned above for TEXTUAL syntax. Namely, he discovered a compositional semantics for elementary Petri nets which is equivalent to the original 'token game' semantics. Yet, the novelty is that (unlike the case of textual syntax) there may be *different* decompositions of a net into subnets. In other words, for a textual syntax a compositional semantics is an homomorphism from an Σ-algebra over a system of *free generators*, whereas in the case of nets

the generators obey some nontrivial relations. Clearly, if there exists a homomorphism from a syntactical algebra to a semantical algebra, then all the equations that hold in the syntactical algebra must hold also in the semantical domain. Mazurkiewicz made the fundamental observation that for elementary Petri nets the signature Σ consists of one binary operation to be interpreted as *combination* of nets (at the syntactical level) and *synchronization* of processes (at the semantical level); the nontrivial relations amount to commutativity and associativity of the operation. In the sequel [10] he formulated compositional semantics of this kind also for the more general classes of P/T-nets, but without comparing it with the already existing token game semantics.

These seminal works in dataflow nets and in Petri nets inspired and strongly influenced the research of modularity for net models. [1, 4, 5, 6, 12, 15, 16, 18, 19, 20, 22]. Note that in these works 'modularity' and 'compositionality' are not clearly distinguished.

Modularity issues for models based on the net concept constitute also one of the major goals in our previous papers [5, 18, 19, 20]. In [5] we used the Mazurkiewicz algebraical approach to formulate an alternative compositional semantics for the token game semantics of P/T-nets. On the other hand, in [18, 19] where our main concern was about the phenomena around the Brock-Ackerman anomaly, we did not rely on any specific textual syntax for nets. An important conceptual and technical novelty that we started to use in [19] and developed in [20] is the idea to consider the semantics of nets of relations in addition to the semantics of nets of processes. In [18] we identified observable relations and nets of observable relations as appropriate tools for the investigation of dataflow networks over nondeterministic agents. The observable relations are the I-O behaviors of (in general, nondeterministic) dataflow agents. Moreover, the semantics of nets of observable relations is the unique semantics which is consistent with the least fixed point solution of systems of equations and with the input-output behavior of dataflow agents. However, it turns out that this semantics is not modular. In [18, 19] 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 the semantics over such nets is modular. We say that a subclass C of relations is **modular** if whenever two nets N_1 and N_2 over relations in C have the same meaning and N_1 is a subnet of a net N over relations in C, then by replacing N_1 by N_2 in N we obtain the net N' which has the same meaning as the net N. It appears that in nontrivial cases the following two tasks are reducible to each other [20]:

- Find a class of dataflow agents which are $\equiv_{rel}$-substitutive.
- Find a modular class of observable relations.

In [20] we asked what nonfunctional agents may be used in dataflow nets without producing modularity anomaly?

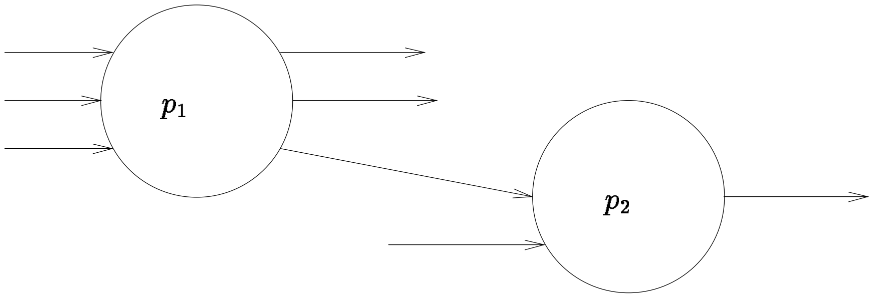

Figure 1: A net N

The central objective of this paper is the characterization of modular classes of relations and hence indirectly the set of dataflow nets without anomalies. Another major theme which plays a technical role in this characterization, but is interesting in its own, is the expressibility for relational nets. Our investigation also reveals the important role played by *stable* functions introduced by Berry [3].

1.3. Overview of Results

1.3.1. Preliminary 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. Here, p_1 and p_2 are names for places 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 a semantics can be defined via an appropriate interpretation of the places. In a relational net the places of the net are interpreted as an input-output relation between values on the input and output channels of the node. The semantics assigns to such a net a relation between values on the input and the output channels of the net; the values on the internal channels are hidden, although in a computation of a global relation one uses values passed through the internal channels. The definition of this semantics (called relational semantics or the semantics of nets of relations) and its justification will be given in section 2. However, few remarks about modularity of net semantics are in order here.

Usually, a textual syntax is used. It is based on a repertoire of operations and every legal syntactical expression has a unique decomposition in the form $op(exp_1,\ exp_2,\ \ldots\ exp_n)$, where op is an operation and exp_i are legal syntactical expressions. Such a syntax suggests the notions of context. Nets are a non standard syntax. Although one may define operations on nets, the

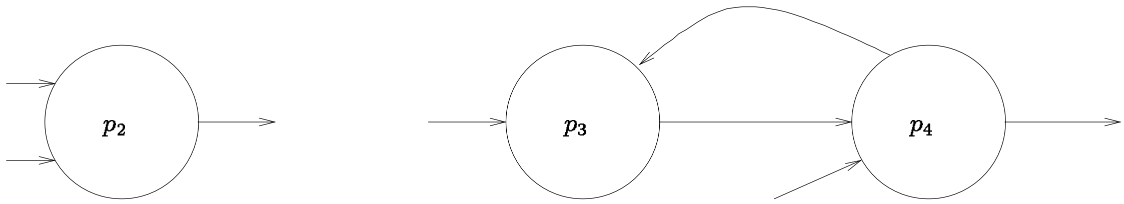

Figure 2: Nets N_1 and N_2

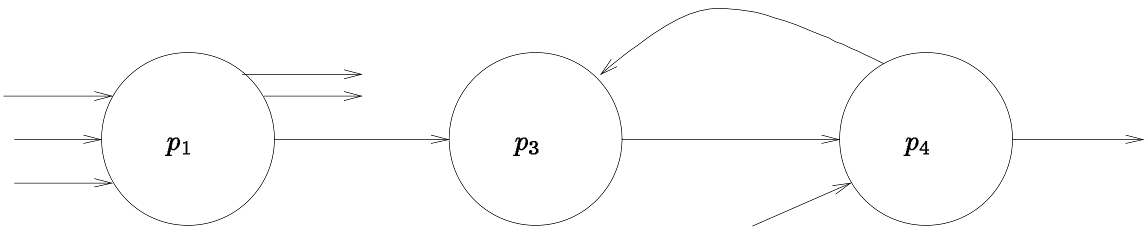

Figure 3: $N' = N[N_2/N_1]$

unique decomposition theorem does not hold. In spite of this, there is a natural notion of a subnet which brings to the notion of context.

Definition 1: *A semantics is called* modular *iff whenever two nets N_1 and N_2 have the same meaning and a net N' is obtained from a net N by replacing the subnet N_1 of N by N_2 then N' and N have the same meaning.*

For example, consider nets N_1 and N_2 in Fig. 2. They have the same type, i.e., the same sets of the input and the 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. 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, relational semantics 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 [16], therefore, relational semantics is of a little interest.

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

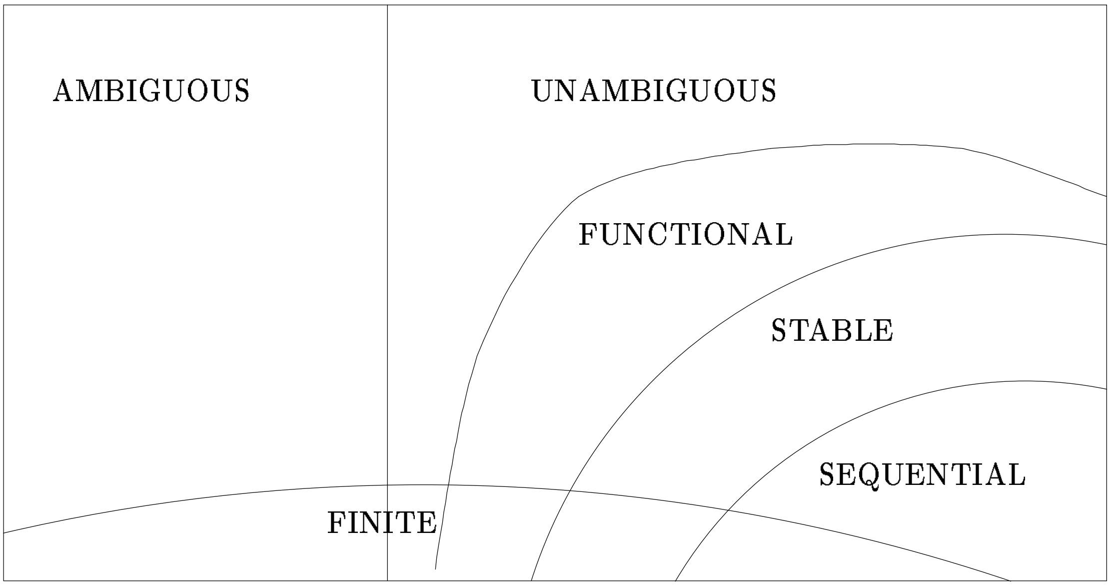

Figure 4: Classification of Relations

We explore in this paper the second alternative.

1.3.2. 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 can 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 with two output channels such that 1 can be produced on the first or on the second channel but not on both. $Unam$ is unambiguous but nonfunctional. It is the 'simplest' nonfunctional relation. Another interesting subclass is the subclass of finite relations. Our classification of relations is given in Fig. 4.

We will deal 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 and the identity function. Second, we require that a class is closed under the net constructs, 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.

Theorem 1: *No modular class can contain an ambiguous relation.*

Theorem 2: *No modular class can contain a nonfunctional relation and an unstable function.*

Theorem 3: *The class of all functional relations is modular. It is a maximal modular class, but there are other maximal modular classes.*

Theorem 4: *The class generated by the stable functions and the relation $Unam$ is modular.*

1.3.3. Plan of the paper In the sequel the exposition is organized as follows:

Section 2 deals with the conceptual background. Here definitions of the semantics are provided and justifications why the semantics of nets of relations is a 'reasonable' semantics are recalled. The exposition here is based on [18, 19, 20].

The semantics of nets of relations is tightly connected with the least fixed point operator (we denote it $loop$) on relations. For functions a simultaneous least fixed point coincides with the corresponding nested fixed points. However, for relations this is not the case and it is a source of the failure of modularity. In section 3 we show that a class C of relations is modular iff for the relations in C the simultaneous and the corresponding nested fixed points are the same. Some laws valid in the algebra of relations are also provided here.

Section 4 deals with a classification of relations.

In section 5 we state expressibility results. In particular, we find the 'simplest' relations for different classes. The expressibility issues are a classical topic of recursion theory. Panangaden et. al. [13, 14] were first to promote this topic for dataflow.

In section 6 we derive theorem 1 and theorem 2 stated above. These theorems were announced in [17].

Section 7 contains the proof that the class generated by the stable functions and the relation $Unam$ is modular. The proof relies on an interesting connection between nets over such relations and nets of functions which 'uniformize' these relations.

Though the dataflow theory is inspired by (and mainly developed for) stream processing there is no reason to focus on specific stream domains. In section 8 we sketch the modification that should be done for more general CPOs which we called in [11, 18, 19] F-domains.

Section 9 discusses some open problems and further work.

2. Background

In this section we introduce the basic notions. Sections 2.1-2.3 deal with nets as syntactical objects. In section 2.1 a full syntax for nets is presented, in section 2.2 substitution for nets is defined and in section 2.3 operations on nets are defined. In section 2.5 observable relations are introduced and their connection with the multi-valued functions is explained. Section 2.7 defines the semantics of nets of functions as solutions of the corresponding systems of equations. In section 2.8 the semantics of nets of observable relations is defined as a natural extension of the semantics of nets of functions. Sections 2.9 and 2.10 describe operations on the observable relations and state their connections with the semantics of nets of relations. This material is based on [18, 19, 20].

2.1. Nets as Syntax

Fig. 5 presents an unabbreviated syntax of the nets. A net is an appropriate labeled graph. Below we provide the requirements on the topology and on the labeling of nets.

Graph: A net is a non-empty bipartite directed graph with nodes of two kinds, pictured as circles and boxes and called respectively places and ports (see Fig. 5). The edges of a net are called channels. If there is a channel ch between port p and place pl then we say that p and pl are adjacent,

We require that in a net:

- Every port is adjacent to at least one place.
- No parallel channels are allowed: for any port and any place in a net there exists at most one channel which connects them.

The ports of a net are partitioned into *input, output* and *internal* ports as follows:

Input ports: ports with no entering channel.
Output ports: ports with no exiting channel.
Internal ports: all the other ports.

A port is an external port of a net N if it is either input or output port of N.

Labeling of channels: A place pl has **arity** $(m;\ n)$ if it has m entering channels and n exiting channels. The entering channels of a place of arity $(m;\ n)$ are injectively labeled by numbers $1, \ldots, m$; The exiting channels are injectively labeled by numbers $1, \ldots, n$.

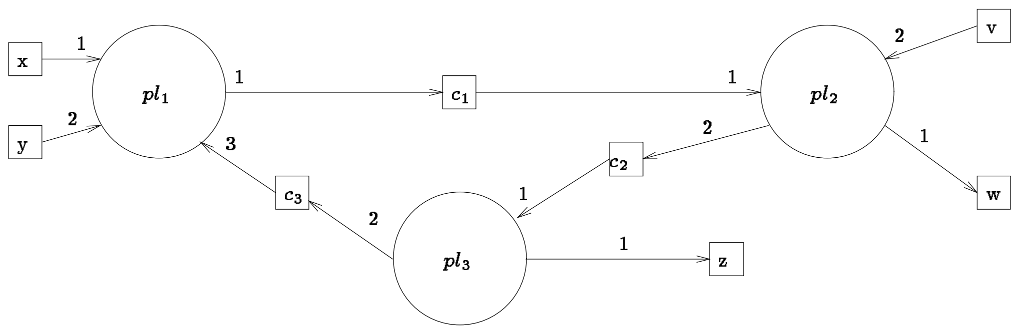

Figure 5: Nets as Syntax

If there is a channel *ch* from port p to the place *pl* (to port p from place *pl*) then p is an input (output) port of *pl*; if number k was assigned to *ch* then p is called k-th input (output) port of *pl*.

Labeling of ports: The ports of a net are labeled by port labels. We require that the external ports of a net have different labels.

The **type of place** *pl* is the pair $\langle P;\ Q\rangle$ where P is the set of labels for input ports of *pl* and Q is the set of labels for output ports of *pl*.

Labeling of places: Places of a net are labeled by place labels. If two places are labeled by the same label l then they have the same arity.

The **type of a net** is the pair $\langle I;\ O\rangle$ where I is the set of labels of its input ports and O is the set of labels of the output ports of the net.

A net is atomic if it has only one place.

Fig. 5 suggests itself. For example, pl_1 has four adjacent ports. Port c_1 is adjacent to places pl_1 and pl_2.

Definition 2: *Two nets are isomorphic if there exists a bijective map ϕ between their nodes such that*

1. *ϕ preserves the labeling of the places.*
2. *ϕ preserves the labeling of the input and the output ports.*

3. *ϕ preserves the adjacency relation, i.e., there exists a channel from place v to port p and the channel has number m if and only if there exists a channel from place $\phi(v)$ to port $\phi(p)$ and number m is assigned to it; similar condition is required for channels entering a place.*

Note that every net is isomorphic to a net in which all ports have different names.

Net Context: The definition of a net-context is similar to the definition of a net except that some places are unlabeled. These places are called the 'holes' of the context. It is required that all the ports adjacent to a hole have different names. It is also required that a context isomorphism preserves the labels of the ports adjacent to the holes.

Dataflow nets: Usually, in dataflow theory, there is a further restriction on the topology of nets: each port has at most one entering and at most one exiting channel. For simplicity, we consider only nets which satisfy this restriction.

Alternative syntax for dataflow nets. For the dataflow nets the following alternative syntax without ports is often used in the literature. It is obtained from our full syntax as follows: (1) delete all ports and channels (2) if a port v with label p is deleted and channel n_1 was entering v from a place pl_1 and channel n_2 was exiting v to place pl_2, then we connect pl_1 and pl_2 by an arc to which the label $\langle n_1, p, n_2\rangle$ is assigned; if an input (output) port with label p is deleted which was connected to a place pl by a channel n then an arc labeled by the $\langle p, n\rangle$ is added which enters (exits) place pl.

We used this alternative syntax in previous pictures and will use it in the sequel.

Parallel channels. In the alternative syntax for dataflow it is sometimes allowed that an arc exits and enters the same place pl. The translation of such a net to our syntax will create parallel channels. Such more general nets can be modeled by the inserting of additional places to our nets which semantically will behave like 'identity' (see e.g., Fig. 6.).

2.2. Substitution

Let pl be a hole of a net context N. We say that a net N_1 is substitutable for a place pl in N if:

1. The sets of the labels of the input and the output ports of N_1 are the same as the sets of the labels of the input and the output ports of pl.

2. N_1 and N do not have common places and ports.

The result of the substitution $N[N_1/pl]$ is the net N_2 defined as follows:

1. $Places(N_2) = Places(N) - \{pl\} \cup Places(N_1)$
2. $Ports(N_2) = (Ports(N) - Ports(pl)) \cup Ports(N_1)$
3. A port p' and a place pl' are connected in N_2 if one of the following cases holds: (a) p' and pl' are connected in N; (b)) p' and pl' are connected in N_1; (c) p' is an external port of N_1 and pl' was connected in N to a port of the hole pl with the label identical to the label of p'.
4. All nodes and channels inherit their labels and all channels inherit their directions.

2.3. Algebra of Nets

Below we consider a set Σ of operations on nets which allow to construct complex nets from more elementary ones. For all these operations (except the renaming operation) all nodes and channels inherit their labels and all channels inherit their directions.

Aggregation: The aggregations of nets N_1 and N_2 is defined if the nets do not have external (i.e. input or output) ports with the same label. The set of nodes in the resulting net is the union of the set of nodes of N_1 and N_2. A port and a place are connected in N if they are connected in N_1 or in N_2. Nodes inherit their labels; channels inherit their directions and labels.

Looping: Looping of an output port labeled y and an input port labeled x in a net N.

The operation $loop(x;\ y)$ in N is defined if x and y are not adjacent to the same place.

Recall that all external ports have different labels. So, in the definition below, we refer to external ports by their labels.

The net $loop(x;\ y)$ **in** N is constructed as follows:

1. Delete x from N.
2. Connect y to the place which was connected to x.

Simultaneous loop: $loop(x_1, x_2 \dots x_n;\ y_1, y_2 \dots y_n)$ in N is defined similarly: first delete $x_1, \dots x_n$ and then connect y_i to the place which was connected to x_i. Sometimes we will use vector notations $loop(\vec{x}; \vec{y})$ in N for a simultaneous loop.

Comments: Note that all looping constructors are only partially defined in order to avoid the creation of nets with parallel channels. Note also that all constructors preserve the number of ports adjacent to a given place. (If parallel channels have been allowed, the looping constructors would be totally defined, but the above invariant would be violated).

Renaming: Let ψ be a one-one function on port labels. We use the notation $N\psi$ for the net obtained by renaming of the ports of N by ψ.

Relying on aggregate, loop and renaming constructors and on some appropriate notations for atomic nets one can formulate a language NET (in the spirit of [4]) for the description of nets. For example, both terms
$\mathbf{loop}(c_1', c_2', c_3'; c_1, c_2, c_3)$ **in** $(pl_1(x, y, c_3'; c_1)$ **ag** $pl_2(c_1', v; c_2, w)$ **ag** $pl_3(c_2'; z, c_3))$ and
$\mathbf{loop}(c_1', c_3'; c_1, c_3)$ **in** $((pl_1(x, y, c_3'; c_1)$ **ag** $\mathbf{loop}(c_2'; c_2)$ **in** $(pl_2(c_1', v; c_2, w)$ **ag** $pl_3(c_2'; z, c_3)))$
describe the net N in Fig. 5.

The set of external ports of a term is defined in a standard way.

If two terms t_1, t_2 of NET describe isomorphic nets, we say that they are graph equivalent and write $t_1 \equiv_{graph} t_2$.

Recall that a net is a dataflow net if each port has at most one entering channel and at most one exiting channel. Recall also that every port in a net is adjacent to some place and that an atomic net is a net with one place.

Lemma 2.1: *[20] The set of dataflow nets is generated from atomic nets by renaming, aggregate and loop constructors.*

Below are equivalences which allow to prove that terms in NET describe the same net:

Lemma 2.2: *(A complete axiomatization of graph equivalence) The following set of equations is sound and complete for proving $\equiv_{graph}$ between terms of NET.*

1. *Aggregation is commutative and associative.*
2. $loop(\vec{x}_1; \vec{y}_1)$ *in* $(loop(\vec{x}_2; \vec{y}_2)$ *in* $t) \equiv_{graph} loop(\vec{x}_1, \vec{x}_2; \vec{y}_1, \vec{y}_2)$ *in* t
3. $loop(x; y)$ *in* $(t_1$ *ag* $t_2) \equiv_{graph} (loop(x; y)$ *in* $t_1)$ *ag* t_2, *provided* y *and* x *are not external ports of* N_2.
4. $t\psi \equiv_{graph} t$ *if* $\psi(c) = c$ *for every external port of* t.
5. *Renaming distributes over aggregation, i.e.,* $(t_1$ *ag* $t_2)\psi \equiv_{graph} (t_1\psi$ *ag* $t_2\psi)$.
6. $(loop(\vec{x}; \vec{y})$ *in* $t)\psi \equiv_{graph} loop(\vec{x'}; \vec{y'})$ *in* $(t\psi)$, *where* $\vec{x'} = \psi(\vec{x})$ *and* $\vec{y'} = \psi(\vec{y})$.
7. *Renaming for atomic nets:* $pl(\vec{x}; \vec{y})\psi \equiv_{graph} pl(\vec{x'}; \vec{y'})$, *where* $\vec{x'} = \psi(\vec{x})$ *and* $\vec{y'} = \psi(\vec{y})$.

We say that a term of language NET is in **normal form** if it was constructed as follows: (1) rename atomic nets; (2) take their aggregation; (3) take simultaneous looping.

A term t' is a normal form of a term t if $t \equiv_{graph} t$ and t' is in normal form.

Lemma 2.3: *(Normal Form) Every term has a normal form.*

2.4. Domains

Definition 3: *Let Δ be an arbitrary set. The stream domain $D = STREAM(\Delta)$ 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 the notation $\sqcup$ for the least upper bound and $\sqcap$ for the greatest lower 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.

It is easy to see that the finite elements of $STREAM(\Delta)$ are finite strings over Δ.

We say that x, y are ***consistent*** elements if they have an upper bound; we say that a set $\{x_i : i \in I\}$ is consistent if it has an upper bound.

2.5. Observable and Approximating Relations

Definition 4: *[18] 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 a finite $y \in D_2$ 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 5: *For a continuous function $f : D_1 \rightarrow D_2$ 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 any approximating relation is an observable relation 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 multi-valued 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 the Hoare power domain of D_2. (2) We do not consider processes (in the process algebraic framework) in this paper. But let us mention that the class of observable relations is the class of I-O behaviors of input-output linear processes [18].

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

2.6. Notational Proviso

Let D be a set of data and P be a set of (port) names. An element x of D^P with $P = \{p_1, ...p_k\}$ is naturally coerced to a tuple $\langle x_{p_1}, ..., x_{p_k} \rangle$ of 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 function $F : D^P \to D^Q$ and the corresponding function $f : D^k \to D^n$. In the sequel we often rely on these coercions, without mentioning them explicitly. For example, we will sometimes talk about the identity function from $D^{\{p\}}$ into $D^{\{q\}}$ and meaning by this that the corresponding coerced function is the identity function on D. Through this abuse of notation we avoid cumbersome expressions in a hopefully, harmless way.

Let P, Q be sets of ports. If R is an observable relation from D^P to D^Q we say that R is an observable relation of type $\langle P;\ Q \rangle$; if f is a function from D^P into D^Q we say that it has type $\langle P;\ Q \rangle$.

2.7. Nets of Functions

We recall definitions from [19]. A net of functions is a pair $\langle N,\ \phi \rangle$, where N is a directed net and ϕ a functional environment which maps the labels of places of N into continuous functions of the same arity. 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.

As it was mentioned before, we consider only nets which have for every port at most one entering channel. Every net is isomorphic to a net in which all ports have different labels and in the definition below we assume that all ports of a net have different labels.

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

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 $p_1, \ldots, p_k$ be input ports of pl. Then $x_q = f_q(x_{p_1}, \ldots x_{p_k})$ 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 the internal ports and get this way a functional G of type $D^{Input} \to D^{Output}$, which by definition is $FUN(N, \ \phi)$.

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

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

2.8. Semantics of Nets of Relations

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

A net of relations is a pair $\langle N, \ \rho \rangle$, where N is a net and ρ is a relational environment which maps the labels of places of N into observable relations of the same type. We say that a relational net $\langle N, \ \rho \rangle$ *approximates* a functional net $\langle N, \ \phi \rangle$ 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 we expect 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, the semantics should assign to $\langle N,\ \rho\rangle$ an observable relation of type $\langle\vec{I};\ \vec{O}\rangle$.

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 $\langle N,\ \rho\rangle$ approximates $\langle N,\ \phi\rangle$ then $sem(N,\ \rho)$ is the approximating relation of the function $FUN(N,\ \phi)$.

Remarks: Recall that the set Obs of observable relations partially ordered by inclusion is a CPO. The ordering of the observable relations induces the pointwise ordering on the set ENV of relational environments: $\rho_1 \leq \rho_2$ if $\rho_1(l) \leq \rho_2(l)$ for every place label l. The second requirement above states that $\lambda\rho.sem(N,\ \rho)$ is a continuous function from ENV into Obs. Also the set of continuous functions from ENV into Obs are ordered pointwise as usual.

Theorem 2.4: *[19]. Among mappings which satisfy (1)-(3) there exists the minimal one.*

Definition 6: *Relational semantics is the minimal mapping which satisfies (1)-(3) above.*

Terminology and Notations: We say that a net of relations $\langle N,\ \rho\rangle$ specifies relation R if R is the relation assigned by the semantics to $\langle N,\ \rho\rangle$. The notation $REL(N,\ \rho)$ is used for the relational semantics. We will use 'semantics of nets of relations' as a synonym for 'relational semantics'.

2.9. Algebra of Observable Relations

The fundamental operations that we define on observable relations are aggregations, loopings and renaming.

Aggregation: If R_i has a type $\langle\vec{I_i},\ \vec{O_i}\rangle$ and $x_i,\ y_i$ are in $Stream^{\vec{I_i}}$, $Stream^{\vec{O_i}}$ respectively then $(R_1 ag R_2)(x_1,\ x_2;\ y_1,\ y_2)$ iff $R_1(x_1;\ y_1)$ and $R_2(x_2;\ y_2)$. Hence, aggregation is conjunction.

Loop: Let R be of the type $\langle\vec{I}\cup\vec{c};\ \vec{O}\cup\vec{c'}\rangle$ and $\vec{x_i},\ \vec{v_i},\ \vec{y_i},\ \vec{w_i}$ be in $Stream^{\vec{I}}$, $Stream^{\vec{c}}$, $Stream^{\vec{O}}$, $Stream^{\vec{c'}}$ respectively. $(loop(\vec{c};\ \vec{c'})\ in\ R)(\vec{x};\ \vec{y})$ iff there is a chain $\langle\vec{x_i},\ \vec{v_i},\ \vec{y_i},\ \vec{w_i}\rangle$ $(i = 1, ...n)$ such that

1. $\langle\vec{x_1},\ \vec{v_1},\ \vec{y_1},\ \vec{w_1}\rangle = \vec{\perp}$.

2. No element of the domain lies between $\langle \vec{x_i},\ \vec{v_i},\ \vec{y_i},\ \vec{w_i} \rangle$ and $\langle \vec{x}_{i+1},\ \vec{v}_{i+1},\ \vec{y}_{i+1},\ \vec{w}_{i+1} \rangle$ for $i = 1, \ldots, n-1$.

3. $\vec{x_n} = \vec{x}$ and $\vec{y_n} = \vec{y}$

4. $\vec{v_i} \leq \vec{w_i}$

5. $R(\vec{x_i},\ \vec{v_i};\ \vec{y_i},\ \vec{w_i})$.

Remark: Note that *loop* does not distribute over union of relations. However, $(loop(\vec{c};\ \vec{c'})\ in\ R_1) \cup (loop(\vec{c};\ \vec{c'})\ in\ R_2) \subseteq loop(\vec{c};\ \vec{c'})\ in\ (R_1 \cup R_2)$.

Renaming: Let ψ be a one-one function on ports. ψ renaming of a relation R of type $\langle P;\ Q \rangle$ (notation $R\psi$) is a relation of type $\langle \psi(P);\ \psi(Q) \rangle$ and it is defined as usual.

Let t be a term in NET and ρ be an environment which maps atoms to observable relations of the appropriate arity. In a compositional way, one can define $Sem(t,\ \rho)$ the meaning of t in environment ρ.

2.10. Semantics of Terms in NET vs Semantics of Nets of Relations

The following theorem from [18] provides a connection between the semantics of nets of relations defined axiomatically in section 2.8 and the semantics of normal terms of NET.

Theorem 2.5: *Let t be a term in normal form which describes a net N and let ρ be an environment. Then $REL(N,\ \rho)$ and $Sem(t,\ \rho)$ denote the same observable relation.*

.

3. Algebra of Relations and Modularity

3.1. Laws

In the following claim we state some equational laws which are valid in the algebra of relations.

Claim 3.1: *(Laws in the algebra of relations)*

1. $R_1 ag R_2 = R_2 ag R_1$

2. $R_1 ag(R_2 ag R_3) = (R_1 ag R_2) ag R_3$

3. $loop(\vec{c};\ \vec{c'})\ in\ (R_1 ag R_2) = R_1 ag(loop(\vec{c};\ \vec{c'})\ in\ R_2)$
   provided $\vec{c}$ *are input ports of* R_2 *and* $\vec{c'}$ *are output ports of* R_2.

4. $loop(\vec{c_1}, \vec{c_2};\ \vec{c'_1}, \vec{c'_2})\ in\ (R_1 ag R_2) = (loop(\vec{c_1};\ \vec{c'_1})\ in\ R_1) ag(loop(\vec{c_2};\ \vec{c'_2})\ in\ R_2)$
   provided that $\vec{c_i}$ *are input ports of* R_i *and* $\vec{c'_i}$ *are output ports of* R_i.

5. *Renaming distributes over aggregation, i.e.,* $(R_1\ ag\ \ R_2)\psi = (R_1\psi\ ag\ \ R_2\psi)$.

6. $(loop(\vec{c};\ \vec{c'})\ in\ R)\psi = loop(\psi(\vec{c});\ \psi(\vec{c'}))\ in\ R\psi$.

Looping on functions is tightly connected with the least fixed point operator. From the fact that the simultaneous least fixed point and the corresponding nested least fixed points coincide, it is easy to show that simultaneous looping on functions (as well as on approximating relations) and the corresponding nested loopings coincide. But, in general, for observable relations simultaneous looping may be different from the corresponding nested loopings and the following equalities do not hold:

Claim 3.2: *(Anomalies). The following equations are not valid.*

1. $loop(c_1,\ c_2;\ c'_1,\ c'_2)\ in\ R = loop(c_1;\ c'_1)\ in\ (loop(c_2;\ c'_2)\ in\ R)$

2. $loop(c_1;\ c'_1)\ in\ (loop(c_2;\ c'_2)\ in\ R) = loop(c_2;\ c'_2)\ in\ (loop(c_1;\ c'_1)\ in\ R)$

Proof: An example which shows that (1) and (2) fail is given below.

Example 2: *Let* $FAIL$ *be the observable relation from* $\boldsymbol{Stream}^2$ *to* $\boldsymbol{Stream}^3$ *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. (Similar conventions are used for the relations described by other tables.) In this example we use* x_i (y_j) *both for a port name and for a value passed over this port; this ambiguity will be resolved by the context.*

$FAIL$

Input		Output		
x_1	x_2	y_1	y_2	y_3
$\perp$	$\perp$	$\perp$	1	$\perp$
$\perp$	1	1	1	$\perp$
1	$\perp$	1	$\perp$	1

$loop(x_2;\ y_2)\ in\ FAIL$

Input	Output	
x_1	y_1	y_3
$\perp$	1	$\perp$
1	1	1

$loop(x_1;\ y_1)\ in\ FAIL$

Input	Output	
x_2	y_2	y_3
$\perp$	1	$\perp$
1	1	$\perp$
1	$\perp$	1

Let $P(y_3)$ be defined as $loop(x_1,\ x_2;\ y_1,\ y_2)\ in\ FAIL$. It is easy to calculate (see section 2.9 for the definition of $loop$) that $P(y_3)$ holds iff $y_3 = \bot$. On the other hand, the corresponding nested looping $loop(x_1;\ y_1)\ in\ (loop(x_2;\ y_2)\ in\ FAIL)$ can produce 1 on its output. Indeed one can check that $R_1(x_1;\ y_1, y_3) \triangleq loop(x_2;\ y_2)\ in\ FAIL$ is described by the table above.

It is easy to see that R_1 is the approximation relation of the function
if $x_1 \geq 1$ then $\langle y_1, y_3\rangle = \langle 1,\ 1\rangle$ else $\langle y_1, y_3\rangle = \langle 1, \bot\rangle$. Let relation $R(y_3)$ be defined as $R(y_3) \triangleq loop(x_1;\ y_1)\ in\ R_1(x_1;\ y_1, y_3)$. It is easy to see that $R(y_3)$ holds iff $y_3 = 1$ or $y_3 = \bot$. Hence $loop(x_1,\ x_2;\ y_1,\ y_2)\ in\ FAIL = P(y_3) \neq R(y_3) = loop(x_1;\ y_1)\ in\ (loop(x_2;\ y_2)\ in\ FAIL)$.

It is also easy to check that the relation $loop(x_1;\ y_1)\ in\ FAIL$ is described by the table above.

Let $Q(y_3) \triangleq loop(x_2;\ y_2)\ in\ (loop(x_1;\ y_1)\ in\ FAIL)$. It is easy to check that $Q(y_3)$ holds iff $y_3 = \bot$. □

Remark: (Algebras of nets vs algebra of relations.) It is instructive to compare lemma 2.2 which gives a complete set of equations for the algebra of nets and claims 3.1 and 3.2. One can see that all identities from lemma 2.2 hold in the algebra of relations except the equality between simultaneous and the corresponding nested loopings.

Though simultaneous looping can be different from the nested loopings, it is easy to show

Claim 3.3: *[18] Simultaneous looping is less than or equal to any of the corresponding nested loopings:* $loop(\vec{c_1},\ \vec{c_2};\ \vec{c'_1},\ \vec{c'_2})\ in\ R \subseteq loop(\vec{c_1};\ \vec{c'_1})\ in\ (loop(\vec{c_2};\ \vec{c'_2})\ in\ R)$.

Remark: (Semantics of nets of relations REL vs semantics Sem of terms in NET) Claims 3.1 and 3.3 and lemma 2.2 imply that if $t_1 \equiv_{graph} t_2$ and t_1 is in normal form, then $Sem(t_1, \rho) \subseteq Sem(t_2, \rho)$. Hence, by theorem 2.5, the semantics of nets of relations can be characterized as follows: $REL(N, \rho) = min\{Sem(t, \rho)\ :\ t \text{ describes } N\}$.

3.2. Looping in Modular Classes

Let C be a set of relations containing the approximation relation for the identity function.

Definition 7: *A set C of observable relations is* closed *under net constructs if for every net N and every environment ρ which maps the places of N into C, the relational semantics of $\langle N,\ \rho\rangle$ is a relation in C.*

Lemma 3.4: *A set of relations which contains the approximation relation for the identity function is closed under net constructs iff it is closed under aggregation and looping.*

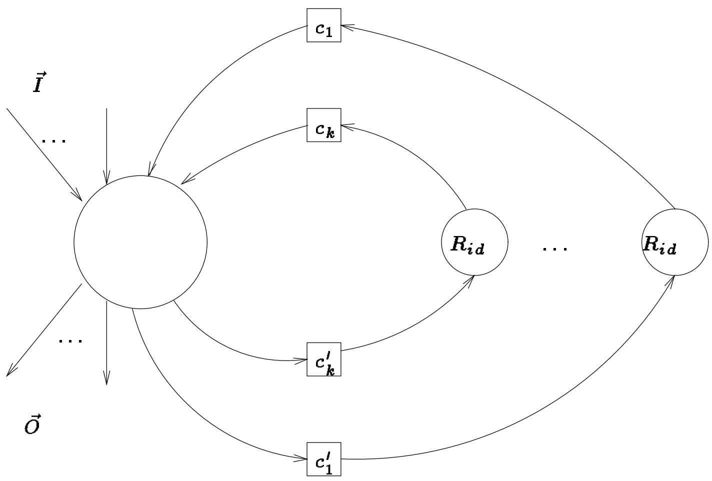

Figure 6: $Con_{loop}[\,]$

Proof: Note that for every dataflow net N there exists a term t in NET that describes a net isomorphic to N. Therefore, the implication from right to left follows from theorem 2.5. Below the other direction is proved. Let R_1 and R_2 be two relations in C of arity $\langle n_1,\ m_1\rangle$ and $\langle n_2,\ m_2\rangle$ respectively. Consider a net N_{aggr} with two places of arity $\langle n_i,\ m_i\rangle$ such that the places have disjoint sets of ports, and the environment ρ which assigns to the first place R_1 and to the second place R_2. Clearly $\langle N_{aggr},\ \rho\rangle$ is the aggregation of R_1 and R_2. Hence, C is closed under aggregation.

In order to show that C is closed under looping consider a net context $Con_{loop}[\,]$ in Fig. 6. It has k place of arity $\langle 1;\ 1\rangle$ to which the approximation relation of the identity function is assigned and a hole. Let ρ be an environment which assigns a relation R to the hole of $Con_{loop}[\,]$. It is easy to show that $\langle Con_{loop}[\,],\ \rho\rangle$ specifies the same relation as $loop(c_1, \ldots c_k;\ c'_1, \ldots c'_k)$ in R. Indeed this follows from theorem 2.5 and the following law in the algebra of observable relations: $loop(x;\ y)\ in\ R(x,\ \vec{u};\ y,\ \vec{v}) = loop(x_1;\ y_1)\ in\ R_1(x_1,\ \vec{u};\ y_1,\ \vec{v})$, where $R_1(x_1,\ \vec{u};\ y_1,\ \vec{v})$ is $loop(x,\ y_2;\ x_2,\ y)\ in\ (R(x,\ \vec{u};\ y,\ \vec{v})\ ag\ R_{id}(x_1;\ x_2)\ ag\ R_{id}(y_2;\ y_1))$.

Hence, if C is closed under the net constructs and contains the approximation relation of the identity function, then it is closed under looping. □

Definition 8: *(Modular sets of relations) A set C of observable relations is* modular *if $REL(N_1,\ \rho) = REL(N_2,\ \rho)$ implies $REL(N[N_1/pl],\ \rho) = REL(N[N_2/pl],\ \rho)$ for every environment ρ that assigns relations from C.*

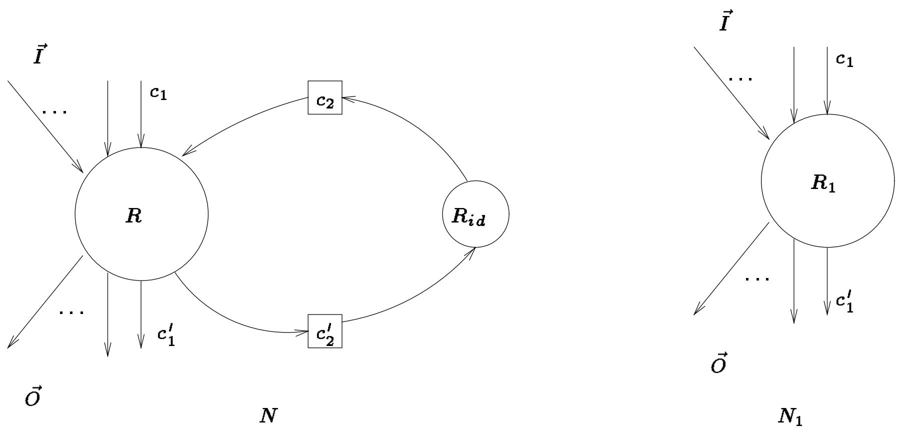

Figure 7: Nets N_1 and N.

Claim 3.2 says that the simultaneous looping of a relation may differ from the corresponding nested loopings. However, if a relation R is in a modular set, then the meanings of simultaneous and nested loopings of R are the same. Moreover,

Claim 3.5: *Let C be a set of observable relations that contains the approximation relation of the identity function and is closed under net constructs. C is modular iff for all the relations in C simultaneous and nested loopings coincide.*

Proof: Assume that C is modular. Let us show that for a relation R in C

$$\textit{loop}(c_1,\ c_2;\ c'_1, c'_2)\ \textit{in}\ R = \textit{loop}(c_1;\ c'_1)\ \textit{in}\ (\textit{loop}(c_2; c'_2)\ \textit{in}\ R) \qquad (*)$$

Consider the nets N and N_1 in Fig. 7, where $R_1 = \textit{loop}(c_2; c'_2)\ \textit{in}\ R$. First, note that R_1 belongs to C, by lemma 3.4. It is clear that both N_1 and N specify the same relation R_1. Consider context $Con[\ ]$ in Fig. 8. Since N and N_1 have the same meaning, it follows that (1) nets $Con[N_1]$ and $Con[N]$ have the same meaning by the modularity of C.

From theorem 2.5, it follows that (2) the meaning of $Con[N_1]$ is the same as the meaning of the normal term $\textit{loop}(c_1;\ c'_1)\ \textit{in}\ R_1$ and this is equal to $\textit{loop}(c_1;\ c'_1)\ \textit{in}\ (\textit{loop}(c_2; c'_2)\ \textit{in}\ R$ by the definition of R_1.

Similarly, we have that (3) the meaning of $Con[N]$ is the same as the meaning of the normal term $\textit{loop}(c_1,\ c_2;\ c'_1,\ c'_2)\ \textit{in}\ R$

The validity of (*) follows from (1), (2) and (3).

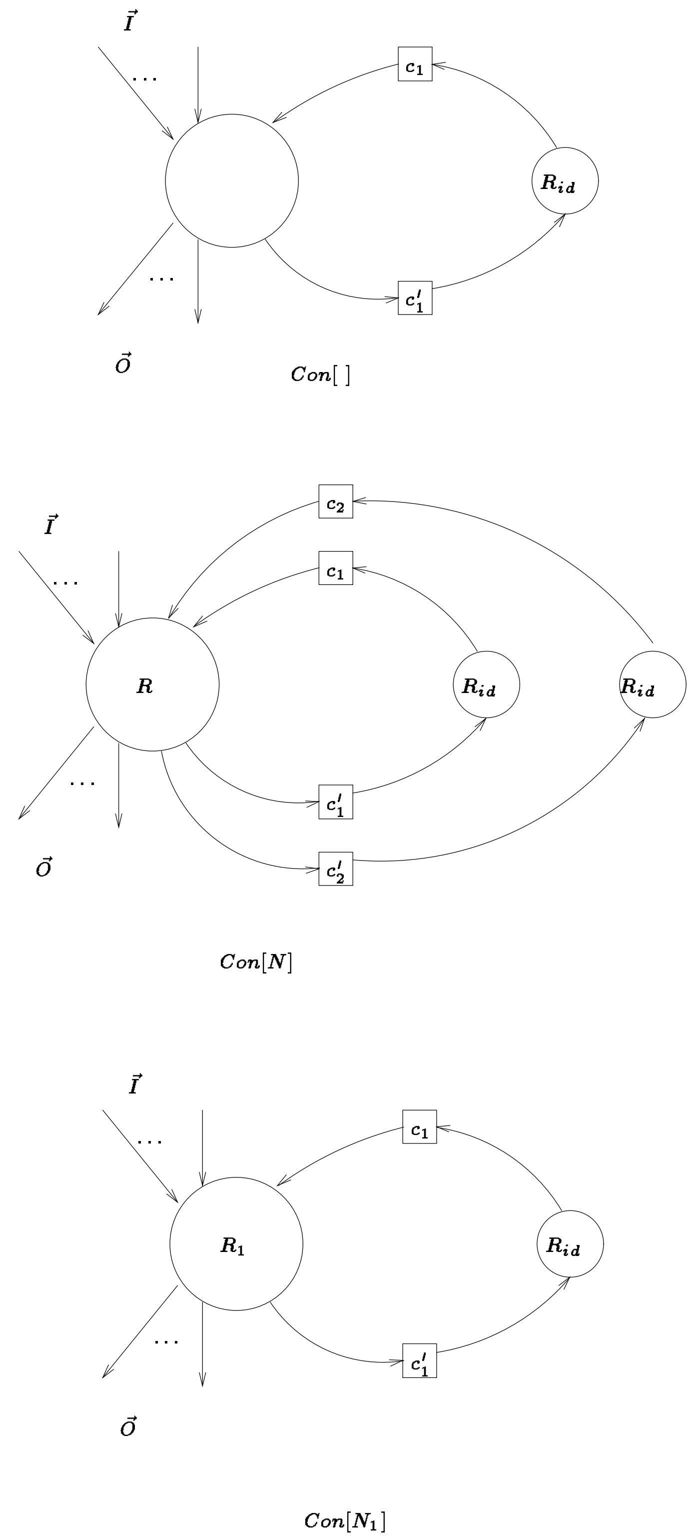

Figure 8: Context $Con[\ ]$ and nets $Con[N_1]$ and $Con[N]$

The proof of equality of simultaneous and nested loopings for the case when more then two channels are connected is similar.

Let us sketch the opposite direction. Note that C is closed under net constructs. Therefore, it is enough to show that for any context $Con[\ ]$ and a net N whose meaning is R the nets $Con[N]$ and $Con[R]$ have the same meaning.

Let R_i be the relations assigned to the places of the context, S_i be the relations assigned to the places of N. (1) Express $Con[N]$ through loop, aggregation and renaming; (2) Express $Con[R]$ through loop, aggregation and renaming; (3) Apply to these expressions the laws from claim 3.1 and the equality of simultaneous and nested loops in order to show that the expressions are equal.

The calculations are simple and we omit them here. $\square$

Later we will use the following

Corollary 3.6: *Let C be a set of relations that is closed under net constructs and contains the approximation relation of the identity function. If C is modular then it does not contain relation FAIL from the example 2 in section 3.1.*

Proof: In example 2 (section 3.1) it was shown that $loop(x_1,\ x_2;\ y_1,\ y_2)\ in\ FAIL$ is different from $loop(x_1;\ y_1)\ in\ loop(x_2;\ y_2)\ in\ FAIL$. Therefore, $FAIL$ cannot belong to a modular set according to claim 3.5. $\square$

4. Classification of Observable Relations

Definition 9: *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 in section 2.5) is unambiguous, but the opposite is wrong (see example 5 below).

Example 3: *The 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.*

Example 4: *Let $Choice_2(;x)$ be the relation without input and with one output port such that only $Choice_2(;\perp)$, $Choice_2(;0)$ and $Choice_2(;1)$ 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. It is clear that $Choice_\omega$ is an ambiguous relation.*

Example 5: *$Unam(;x,\ y)$ is a relation with two output ports and no input ports. 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 10: *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 relations. $Choice_2$ and $Unam$ are finite relations, $Choice_\omega$ is not finite. The relation $\geq$ (see example 3) is functional but not finite.

Let f be a function from $STREAM^n$ into $STREAM^m$. Assume that $f(x_1,\ x_2, ... x_n) = \langle y_1,\ y_2, ... y_m \rangle$. We say that f is ***sequential at*** $\vec{x}$ if for every output port j there is an input port i such that y_j cannot be increased without increasing x_i. Formally, Let $\vec{z} = \langle z_1,\ z_2, \ldots, z_n \rangle$ be greater than $\langle x_1,\ x_2, \ldots, x_n \rangle$. Assume that $f(\vec{z}) = \langle v_1,\ v_2, \ldots, v_n \rangle$. Then, either $z_i > x_i$ or $v_j = y_j$. A function f is ***sequential*** if it is sequential at every $\vec{x}$.

Clearly, every step function is a sequential function. The definition of sequentiality above was introduced by Kahn and Plotkin [8]. The notion of sequentiality can be defined not only for Cartesian products of stream domains, but also for more general ***concrete domains*** [8, 24]. Stable functions, introduced by Berry [3], are a generalization of sequential functions.

Definition 11: *[3] 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 non-sequential stable functions [3].

Example 6: *Let a_1, a_2, b_1 and b_2 be finite elements. Assume that a_i are inconsistent. Then the least upper bound of step functions $a_i \to b_i$ is a sequential function. This remark does not hold for the least upper bound of more then two step functions, even when all a_i are pairwise inconsistent. For example, let $a_1 = \langle 1,\ 0,\ \perp \rangle$, $a_2 = \langle \perp,\ 1,\ 0 \rangle$, $a_3 = \langle 0,\ \perp,\ 1 \rangle$ and $b_i = 0$. Then the least upper bound of $a_i \to b_i$ is stable but non-sequential.*

Example 7: The function Par. *Recall that $STREAM(\Delta)$ is the set of finite and infinite sequences of elements from a set Δ. $STREAM(\Delta)$ is partially ordered by a prefix relation. For $\alpha \in \Delta$ we denote by α the sequence α of length one.*

Let $1 \in \Delta$. The function $Par : STREAM(\Delta)^2 \to STREAM(\Delta)$ is defined as follows:

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

Par is not a stable function, because $Par(\perp,\ 1) = Par(1,\ \perp) = 1 = Par(\perp,\ 1) \sqcap Par(1,\ \perp) \neq \perp = Par(\langle \perp\ , 1\rangle \sqcap \langle 1,\ \perp\rangle)$. Par is finite; it is the least upper bound of $\langle 1,\ \perp\rangle \to 1$ and $\langle \perp,\ 1\rangle \to 1$.

Let S be a set of relations. We say that $\langle N.\ \rho\rangle$ is a net over relations in S if ρ assigns to the places of net N relations in S.

Claim 4.1: *(Closure properties of relations under nets).*

1. *A net over unambiguous relations specifies an unambiguous relation.*
2. *A net over finite relations specifies a finite relation.*
3. *A net over functional relations specifies a functional relation.*
4. *A net over step functions specifies a step function.*
5. *A net over sequential functions specifies a sequential function.*
6. *A net over stable functions specifies a stable function.*

Comments Note that (1), (2), (3) deal with nets of relations; (4), (5), (6) deal with nets of functions. Functions in (4), (5), (6) can be replaced by their approximating relations. For example (4) will be modified as follows: a net over approximating relations of step functions specifies the approximating relation for a step function.

5. Expressibility

For a set of relations S and a relation R we denote by $Exp(S; R)$ the set of relations which are specified by the nets over $S \cup R$. We say that 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 can 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 clear from a context we will say 'expressibility' instead 'S-expressibility', 'degree' instead 'S-degree', etc.

Notation: We use the notation $FinSeq$ for the set of all finite sequential functions.

Claim 5.1: *(Minimal degrees).*

1. $\boldsymbol{Choice_2}$ *is* $\boldsymbol{FinSeq}$*-expressible from any ambiguous relation. It is the minimal* $\boldsymbol{FinSeq}$*-degree among ambiguous relations.*
2. $\boldsymbol{Unam}$ *is* $\boldsymbol{FinSeq}$*-expressible from any unambiguous nonfunctional relation. It is the minimal* $\boldsymbol{FinSeq}$*-degree among unambiguous nonfunctional relations.*
3. $\boldsymbol{Par}$ *is* $\boldsymbol{FinSeq}$*-expressible from any unstable functional relation. It is the minimal* $\boldsymbol{FinSeq}$*-degree among unstable functional relations.* $\boldsymbol{Par}$ *is not expressible from* $\boldsymbol{Unam}$.

For the modularity results, we will need only the following facts
(1) $Choice_2$ is weakly $FinSeq$-expressible from any ambiguous relation.
(2) $Unam$ is weakly $FinSeq$-expressible from any unambiguous nonfunctional relation.
(3) Par is weakly $FinSeq$-expressible from any unstable functional relation.
Proofs of these facts are given below.

Proof: (1) Let R be an ambiguous relation. Then there are consistent x_1 and x_2 such that $R(x_1;\ y_1)$ and $R(x_2;\ y_2)$, but y_1 and y_2 are inconsistent. Recall that observable relations increase on inputs (see definition 4 section 2.5), hence $R(x_1 \sqcup x_2;\ y_1)$ and $R(x_1 \sqcup x_2;\ y_2)$. Consider the net in Fig. 9, where f is the constant function $x_1 \sqcup x_2$ and g is the sequential function $y_1 \to 0 \sqcup y_2 \to 1$. Since $x_1,\ x_2,\ y_1$ and y_2 are finite it follows that f and g are finite functions. Function f is sequential because it is a constant function. Function g is sequential, because y_1 and y_2 are inconsistent.

Easy calculations show that the net N in Fig. 9 specifies $Choice_2$. Hence, $Choise_2$ is weakly expressible from R.

It is also easy to see that for any context $Con[\ ]$, the nets $Con[Choise_2]$ and $Con[N]$ specify the same relations, therefore, $Choice_2$ is strongly expressible from R.

□

Proof: (2) First, note

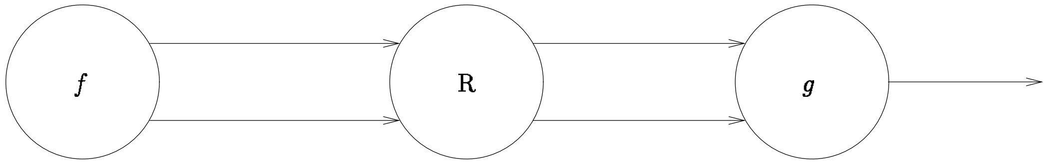

Figure 9: Construction of $Choice_2$ from an ambiguous relation

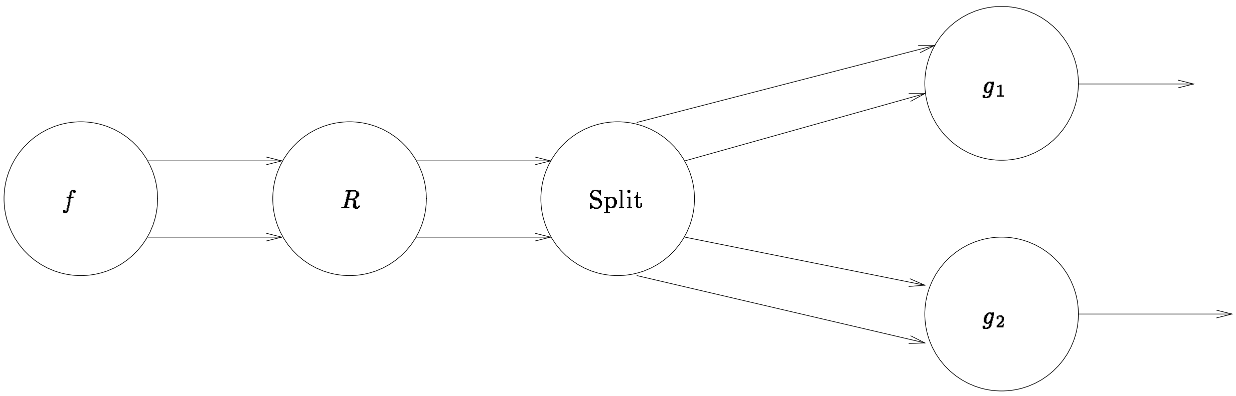

Figure 10: Construction of $Unam$ from a nonfunctional relation

Fact: An unambiguous relation R is a functional relation iff $R(\vec{x};\ \vec{y_1})$ and $R(\vec{x};\ \vec{y_2})$ imply $R(\vec{x};\ \vec{y_1} \sqcup \vec{y_2})$.

Now, given an unambiguous nonfunctional relation R. By the fact above there are $\vec{x},\ \vec{y_1}, \vec{y_2}$ such that $R(\vec{x};\ \vec{y_1})$ and $R(\vec{x};\ \vec{y_2})$ but not $R(\vec{x};\ \vec{y_1} \sqcup \vec{y_2})$. Consider the net in Fig. 10, where (1) f is the constant function which produces $\vec{x}$. (2) $Split =_{def} \lambda\vec{z}.\ let\ \vec{u} = \vec{z} \sqcap (\vec{y_1} \sqcup \vec{y_2})\ in\ \langle\vec{u},\ \vec{u}\rangle$. It makes two copies of the inputs for inputs less than $\vec{y_1} \sqcup \vec{y_2}$. (3) g_1, g_2 are two step functions defined as $g_1 =_{def} \vec{y_1} \rightarrow 1$ and $g_2 =_{def} \vec{y_2} \rightarrow 1$. It is easy to see that the net specifies $Unam$. It can be shown that the net is strongly equivalent to $Unam$. Also note that in our construction we use only approximation relations for finite sequential functions. $\square$

Proof: (3) It is easy to show

Lemma 5.2: *Assume that, for all finite elements x, y, z, consistency of x with y implies that $(f(x) \sqcap f(y) > z) \rightarrow (f(x \sqcap y) > z)$. Then f is stable.*

Now let f be unstable. By the lemma, there are finite x, y, z such that

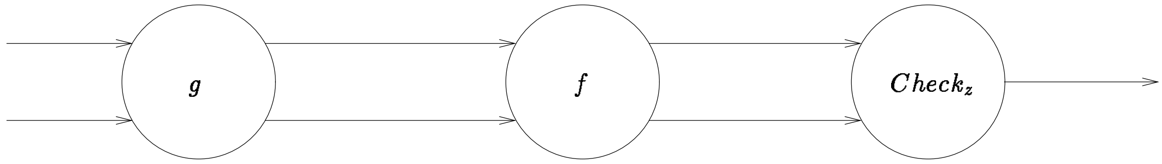

Figure 11: Construction of Par

- x is consistent with y.
- $f(x) \sqcap f(y) > z$ but not $f(x \sqcap y) > z$.

Let g be defined as $\sqcup\{\langle\bot,\ 1\rangle \to x,\ \langle 1,\ \bot\rangle \to y,\ \langle\bot,\ \bot\rangle \to x \sqcap y,\ \langle 1,\ 1\rangle \to (x \sqcup y)\}$. One can show that for finite elements x, y in the domain $Stream^k$ the function g is finite sequential.
Let

$$Check_z(v) = \begin{cases} 1 & \text{if } v > z \\ \bot & \text{otherwise} \end{cases}$$

It is clear that $Check_z$ is finite sequential.

Now $Par = \lambda\langle u, v\rangle.\ Check_z(f(g(\langle u,\ v\rangle))$. The net which corresponds to this expression is pictured in Fig. 11. □

Claim 5.3: *(Weak $FinSeq$-expressibility)*

1. *Any finite relation is weakly expressible from $Choice_2$*
2. *$FAIL$ is weakly expressible from Par and $Unam$.*

Proof: Let R be a finite relation (see section 4). Then for a natural number n there are 2^n step functions $a_i \to b_i$ such that $R = \bigsqcup\{a_i \to b_i\})$. The net in Fig. 12 specifies R. Here, the relation H has in addition to the channels of R one new input channel; H is defined by $H = \bigsqcup(\{\langle i,\ a_i\rangle \to b_i\})$. And it is a finite sequential function.

G has n input channels and one output channel. From every input channel it reads the first element (it is either 0 or 1). G produces on the output a number k if the tuple it reads is the binary representation of k. Clearly, it is a finite sequential function. Other n places of the net are interpreted as $Choice_2$. It is easy to see that the net in Fig. 12 specifies R.

In order to prove (2), consider the net in Fig. 13. It consists of Par, $Unam$ and the finite sequential functions $fork \triangleq 1 \to \langle 1,\ 1\rangle$, $f_1 \triangleq \bigsqcup\{\langle 1,\ \bot,\ \bot\rangle \to \langle\bot,\ 1, \bot\rangle, \langle 1,\ \bot,\ 1\rangle \to \langle 1,\ 1,\ \bot\rangle\}$

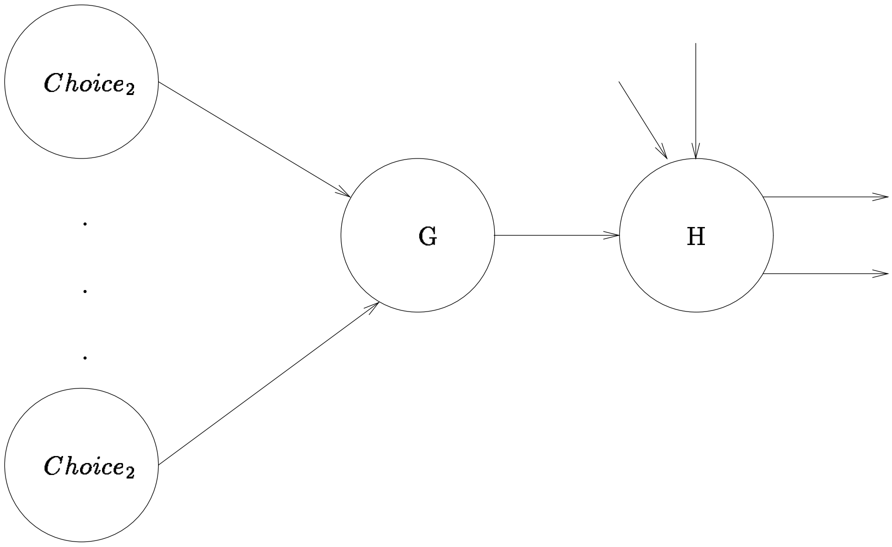

Figure 12: Weak expressibility from $Choice_2$

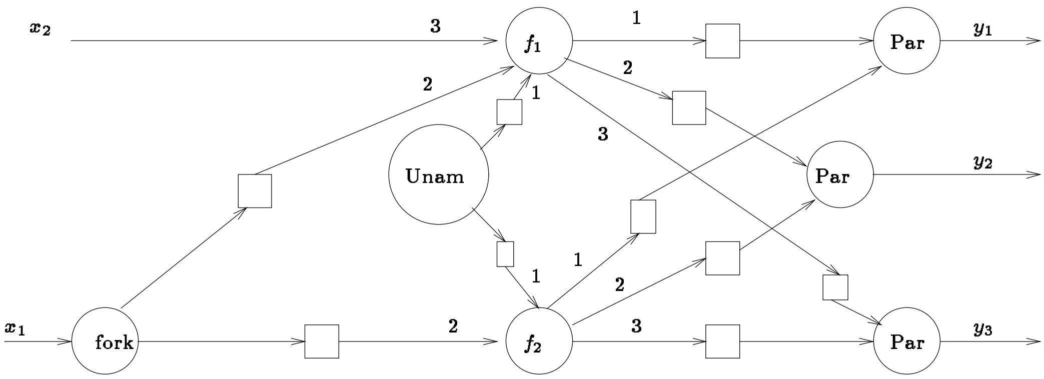

Figure 13: Construction of $FAIL$

and $f_2 \triangleq \langle 1,\ 1\rangle \rightarrow \langle 1,\ \perp,\ 1\rangle$. In Fig. 13 we provided the labels only for the channels entering/exiting places f_1 and f_2; other channels might be labeled arbitrary. It is easy to check that this net specifies the relation $FAIL$ from example 2 (section 3.1). $\square$

Comment: (1) It can be shown that any finite relation is strongly expressible from $Choice_2$. However, for this purpose another construction is needed. (2) The net N in Fig. 13 is strongly equivalent to $FAIL$, i.e., for every context $Con[\ ]$ the nets $Con[N]$ and $Con[FAIL]$ specify the same relations.

6. Non Modular Classes

Definition 12: *A* class *is a set C of relations that has the following properties.*

1. *C has a basic computational power. This amounts to the requirement that it contains the approximating relations of all finite sequential functions and the approximation relation of the identity function.*
2. *C is closed under net constructs, i.e., any net over relations in C specifies a relation in C.*

Recall also the corollary 3.6. It says that no modular class contains the relation $FAIL$.

Since $FAIL$ is a finite relation, it follows by claim 5.3(1) that any class containing $Choice_2$ contains $FAIL$. But $Choice_2$ has the minimal degree among ambiguous relations. Therefore,

Theorem 6.1: *No modular class contains an ambiguous relation.*

Or in other words,

Corollary 6.2: *Any modular class contains only unambiguous relations.*

Claim 5.3 (2) implies that any class containing Par and $Unam$ contains $FAIL$. Par is the minimal degree among approximating relations of unstable functions; $Unam$ is the minimal degree among nonfunctional relations. Therefore,

Theorem 6.3: *No modular class contains both the approximating relation of an unstable function and a nonfunctional relation.*

Corollary 6.4: *If a modular class contains the approximating relation of an unstable function, then it is a subclass of the approximating (functional) relations.*

7. Modular Classes

It is well known that simultaneous and nested least fixed points coincide for continuous functions. Therefore, semantics of nets of functions is modular. Recalling the connection between nets of functions and nets of relations (definition 6 section 2.8) we obtain from claim 3.5

Theorem 7.1: *The set functional relations is modular.*

This class contains unstable functional relations, therefore, applying corollary 6.4, we obtain

Corollary 7.2: *The set of functional relations is a maximal modular class.*

By corollary 6.2, all modular classes contain only unambiguous relations. Moreover, (by corollary 6.4) if a modular class contains a nonfunctional relation it can contain only stable functions. We are going to show

Theorem 7.3: *The class generated[1] by* $Unam$ *and the approximating relations of all stable functions is a modular class.*

Note that this class contains all unambiguous relations without inputs (actually, such relations are expressible from $Unam$). Below we introduce class K of stable relations which contains the class generated by $Unam$ and the approximating relations of all the stable functions and show that K is modular. We do not know, whether this class is a maximal modular class. It may happen that an extension of this class by some unambiguous relations with input is modular.

Definition 13: *A set of functions* $F = \{f_i : i \in I\}$ *is a stable bunch of type* $D \to E$ *if there is a stable function* $g : D_1 \times D \to E$ *and a consistent set* $\{a_i\} \subset D_1$ *such that* $f_i(x) = g(a_i, x)$.

Definition 14: R *is a stable relation if there is a stable bunch* F *such that* $R(x;\ y) \leftrightarrow \exists f \in F.\ f(x) \geq y$.

$Unam$ is a stable relation. Its stable bunch is defined by the identity function $g : Stream^2 \to Stream^2$ and $a_1 = \langle \perp,\ 1\rangle$, $a_2 = \langle 1,\ \perp\rangle$. It is also clear that for every stable function its approximating relation is a stable relation.

Theorem 7.4: *The set* K *of stable relations is a modular class.*

In the rest of this section the proof of the theorem 7.4 is presented. Let us first sketch the structure of our proof.

A relation R in K is represented by a special set F of stable functions which uniformize R (its stable bunch). $R(x;\ y)$ holds iff there is $f \in F$ such that $f(x) \geq y$. Therefore, R is uniquely determined by F, but different (stable) bunches may represent the same relation.

Bunches will have the following property (see lemma 7.9 below):

(*) If F is a stable bunch of R then $\{loop(\vec{c}; \vec{c'})\ in\ f : f \in F\}$ is a stable bunch of $loop(\vec{c}; \vec{c'})\ in\ R$.

[1] We say that class C of relations is generated by a set S of relations if C is the minimal class (see definition 12) of relations which contains S.

Note that in (*) the first occurrence of *loop* denotes the operation on continuous functions and the second occurrence of *loop* denotes the operation on observable relations. This ambiguous notation is also employed in the sequel. We hope that it is clear from the context what version of *loop* is used. Recall also that simultaneous and nested loops on continuous functions are equal.

From (*) modularity of K is proved as follows: First, if F is a bunch of R then $\{loop(\vec{c_1},\ \vec{c_2};\ \vec{c_1}',\ \vec{c_2}')\ in\ f\ :\ f \in F\}$ is a bunch of $loop(\vec{c_1},\ \vec{c_2};\ \vec{c_1}',\ \vec{c_2}')\ in\ R$ and $\{\ loop(\vec{c_1};\ \vec{c_1}')\ in\ loop(\vec{c_2};\ \vec{c_2}')\ in\ f\ :\ f \in F\}$ is a bunch of $loop(\vec{c_1};\ \vec{c_1}')\ in\ loop(\vec{c_2};\ \vec{c_2}')\ in\ R$. Since simultaneous looping on functions and the corresponding nested loopings coincide, $loop(\vec{c_1},\ \vec{c_2};\ \vec{c_1'},\ \vec{c_2'})\ in\ f$ and $loop(\vec{c_1};\ \vec{c_1'})\ in\ loop(\vec{c_2};\ \vec{c_2'})\ in\ f$ coincide. Hence $loop(\vec{c_1},\ \vec{c_2};\ \vec{c_1'},\ \vec{c_2'})\ in\ R$ and $loop(\vec{c_1};\ \vec{c_1'})\ in\ loop(\vec{c_2};\ \vec{c_2'})\ in\ R$ coincide. Then we show that K is closed under net constructs (lemma 7.10), therefore, modular of K will follow by claim 3.5 (section 3.2).

The proof below is organized as follows. First, some technical properties of stable functions are proved. Then, we define and investigate stable bunches and prove property (*) stated above which implies modularity. Finally, we show (lemma 7.10) that the set of stable relations is closed under net constructs.

Recall that a function f is stable iff $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$ for consistent $x_1,\ x_2$.

Note that by claim 4.1(6) and lemma 3.4

Lemma 7.5: *The set of stable functions is closed under aggregation and looping.*

Later we will use the following technical

Lemma 7.6: *If* $f\ :\ D_1 \times D_2 \to E$ *is a stable function and* $a,\ b \in D_1$ *are consistent then:*

$$x_1 < x_2 \ \ and\ \ y_1 < y_2\ \ and\ \ f(a,\ x_1) \geq y_1\ \ and\ \ f(b,\ x_2) \geq y_2$$
$$imply$$
$$f(b,\ x_1) \geq y_1$$

Proof:

$$f(a,\ x_1) \geq y_1\ \ and\ \ f(b,\ x_2) \geq y_2$$

Since f is stable,

$$f(a \sqcap b,\ x_1 \sqcap x_2) \geq y_1 \sqcap y_2$$

Note that by the assumptions $x_1 = x_1 \sqcap x_2$ and $y_1 = y_1 \sqcap y_2$. Therefore

$$f(a \sqcap b,\ x_1) \geq y_1$$

And by the monotonicity of f

$$f(b,\ x_1) \geq y_1$$

$\square$

Lemma 7.7: *Stable bunches are closed under aggregation and looping.*

Proof: Follows directly from the definitions and lemma 7.5. $\square$

Recall that we use notation R_f for the approximation relation of a function f. We also will use R_F for the relation defined by a bunch F.

Lemma 7.8: *For a stable bunch F and its relation R_F*

$$loop\ (\vec{c}; \vec{c}')\ in\ R_F = \bigcup\{loop\ (\vec{c}; \vec{c}')\ in\ R_f :\ f \in F\}$$

Proof: Let $G = loop\ (\vec{c}; \vec{c}')\ in\ R_F$. Since $loop$ is monotonic it follows that $G \supset loop\ (\vec{c}; \vec{c}')\ in\ R_f$. for $f \in F$, therefore $G \supset \bigcup\{loop\ (\vec{c}; \vec{c}')\ in\ R_f :\ f \in F\}$.

The reverse inclusion holds for the following reasons:

Note that by the definition of loop (see section 2.9) and by the requirement (definition 4. section 2.5) that observable relations increase on inputs and decrease on outputs, if $G(x;\ y)$, then there is a chain

$$\perp = v_1 < v_2, < \ldots < v_n$$

such that

$$R_F(v_i,\ x;\ v_{i+1},\ \perp) \quad for\ i = 1, \ldots, n-1 \tag{1}$$

$$R_F(v_n,\ x;\ v_n,\ y) \tag{2}$$

Therefore, there are indices $k(i)$ and functions $f_{k(i)} \in F$ such that

$$\begin{aligned} f_{k(i)}(v_i,\ x) &\ \geq\ \langle v_{i+1},\ \perp\rangle \\ f_{k(n)}(v_n,\ x) &\ \geq\ \langle v_n,\ y\rangle \end{aligned}$$

Recall that F is a stable bunch. Hence there is a stable function g and consistent values $\{a_i\ : i \in I\}$ such that $F = \{f_i\ :\ f_i(v, x) = g(a_i, v, x)\}$. The above in-equations imply

$$\begin{aligned} g(a_{k(i)},\ v_i,\ x) &\ \geq\ \langle v_{i+1},\ \perp\rangle \text{ for } i = 1, \ldots, n-1 \\ g(a_{k(n)},\ v_n,\ x) &\ \geq\ \langle v_n,\ y\rangle \end{aligned}$$

and by lemma 7.6 we obtain

$$g(a_{k(n)},\ v_i,\ x)\ \geq\ \langle v_{i+1},\ \perp\rangle \text{ for } i=1,\ldots n-1$$
$$g(a_{k(n)},\ v_n,\ x)\ \geq\ \langle v_n,\ y\rangle$$

Hence $loop\ R_{f_{k(n)}}(x;\ y)$ and therefore $G \subset \bigcup\{loop\ R_f :\ f \in F\}$. □

Recall that looping of observable relations is consistent with looping of functions; in particular $loop(\vec{c}; \vec{c'})\ in\ R_f$ is the approximation relation for $loop(\vec{c}; \vec{c'})\ in\ f$. Therefore, from lemma 7.7 and lemma 7.8 it follows

Lemma 7.9: *If F is a stable bunch of R then $\{loop(\vec{c}; \vec{c'})\ in\ f\ :\ f \in F\}$ is a stable bunch of $loop(\vec{c}; \vec{c'})\ in\ R$.*

Lemma 7.10: *Any net over stable relations specifies a stable relation.*

Proof: Recall that by theorem 2.5, the relation specified by a net N coincides with the relation assigned to a term t in normal form which describes N. Such a term t can be constructed as follows:

1. Rename the relation assigned to the places of the net.
2. Take aggregation of these relations.
3. Apply simultaneous looping to the relation obtained in step 2 over all internal ports of N.

It is clear that stable relations are closed under aggregation and renaming. By lemma 7.9 they are also closed under looping. □

8. From Streams to F-domains

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. Let us sketch here the modification that should be done for more general CPOs which we called in [11, 18, 19] F-domains. Here are the relevant definitions.

Recall that a **finite** element x is characterized by the following condition:
assume that $x \leq a$, where a is the least upper bound of a sequence $a_1 \leq a_2 \leq \ldots$; then $x \leq a_n$ for some n.

We will say that x_1 immediately precedes x_2, or x_2 covers x_1 if $x_1 < x_2$ and there is no element between x_1 and x_2. A finite chain $s = \{x_i\ :\ i = 1...n\}$ is called *strict* if it $x_1 = \perp$ and x_i immediately precedes x_{i+1} for all $i < n$; we say that this strict chain leads to x_n. x is *finitely reachable* if there is a strict chain leading to x.

Definition 15: *A CPO D is said to be a F-domain if it obeys the following finiteness conditions:*

1. *For each element z in D the set of finitely reachable elements bounded by z is directed and has z as its least upper bound.*
2. *If c, a are finitely reachable and $c < a$ then there is a chain $s = \{x_i \ : \ i = 1 \dots n\}$ such that $c = x_1$, $a = x_n$ and x_i immediately precedes x_{i+1} for all $i < n$.*

Example 8: *(F-domains)*

Multiset Domain*: Let Δ be a set. By a multiset over Δ we mean any mapping $m : \ \Delta \to \omega$. If m is a multiset over Δ and $a \in \Delta$ then $m(a)$ is called the multiplicity of a in m. For any two multisets m_1 and m_2 we say that m_1 is included in m_2 if $m_1(a) \leq m_2(a)$ for any $a \in \Delta$. The multiset domain consists of all multisets ordered by inclusion. It is easy to check that the multiset domain is F-domain.*

Streams with 'holes'*: Sometimes [25] in dataflow theory the domain of stream with holes is considered. This domain consists of all infinite sequence over $\Delta \cup \perp$ and is partially ordered by pointwise ordering. It is isomorphic to the set of all partial functions from the natural numbers into Δ. It is easy to check that the domain of stream with holes is F-domain.*

Also note that every concrete domain [8] is F-domain.

For F-domains all the definitions and results from sections 2, 3 are unchanged. The notion of sequentiality is not well defined for these domains, but the definitions of all the other classes from section 4 are unchanged; we will consider the questions of modularity for classes containing all finite stable functions. The definition of $Choice_2$, $Unam$ and Par should be changed in the following way:

Choice.
Let a and b be finite inconsistent elements of the domain under consideration. $Choice_{a,b}(;x)$ holds if $x \leq a$ or $x \leq b$.

Unam.
Let a and b be finite consistent elements of the domain under consideration such that $a \neq (a \sqcup b) \neq b$. By the definition $Unam_{a,b}(;x)$ holds if $x \leq a$ or $x \leq b$.

Par.
Let a, b, c be finite elements such that:
(1) a is consistent with b
(2) $a < a \sqcup b$

(3) $b < a \sqcup b$

Then

$$Par_{a,\ b,\ c}(x) = \begin{cases} c & \text{if } x \geq a \\ c & \text{if } x \geq b \\ \perp & \text{otherwise} \end{cases}$$

Under these modifications all the results of section 4 hold except for Claim 4.1(5). The results from section 5 also hold if we replace $FinSeq$-expressibility by $FinStab$-expressibility.

The results of sections 6, 7 are also unchanged.

9. Further Research

9.1. The Structure of Degrees

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

Claim 9.1: *Every observable relation is strongly expressible from* $\boldsymbol{Choice}_\omega$ *and the approximation relation of sequential functions. Therefore,* $\boldsymbol{Choice}_\omega$ *is the maximal* $\boldsymbol{Seq}$*-degree.*

As in the case of degrees of parallelism [23, 21] 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 with respect to the finite sequential functions. The structures of degrees with respect to the sequential functions, the stable functions, all functions are also important.

9.2. Other characterizations of non modular classes

Let us recall how we proved that some classes are not modular (Section 6). We proved that these classes contain the specific relation called $FAIL$ which cannot belong to any modular class. We suppose that the following is true.

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.

9.3. Modular classes

We proved modularity of two classes: the class of functional relations and the class generated by $Unam$ and stable functions.

The first of them is a maximal modular class. A natural extension of the second class is the class of relations which have the following stability property.

$$x_1,\ x_2 \text{ are consistent, } R(x_1;\ y_1) \text{ and } R(x_2;\ y_2)$$
$$\text{imply}$$
$$R(x_1 \sqcap x_2;\ y_1 \sqcap y_2))$$

However, this class is not modular. We conjecture that class K is a maximal modular class and that there are only two maximal modular classes.

9.4. Reasoning

The reasoning about relational nets has to be further developed. In section 3.1 only few laws were provided. The fact that simultaneous and nested fixed points are different makes the reasoning about such nets difficult. We are working on this subject and have already made initial steps. Some laws and the connection between relational semantics and conjunction of relations on the one hand and synchronization of concurrent processes on the other hand can be found in [11, 19]. Let us mention here

Claim 9.2: *(Universal context). There is a sequential functional context $U[\ ]$ such that for every pair N_1, N_2 of nets N_1 is replaceable by N_2 in any context iff they are replaceable in U.*

The fact that the nets over functional relations are modular contrasts

Fact: There are two nets N_1, N_2, over nonfunctional relations which specify the same functional relations and a functional context $Con[\ ]$ such that the nets $Con[N_1]$ and $Con[N_2]$ specify different functions.

Acknowledgments

I am very grateful to B. A. Trakhtenbrot for his encouragement while writing this paper; its content was very much influenced by his ideas. I wish also to thank the anonymous referees for their detailed comments and helpful suggestions.

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. Berry, Stable models of typed lambda calculi, in *Proceedings, 5th International Colloquium on Automata, Language and Programming*, pp. 72-89, LNCS, Vol. 62, Springer Verlag.

[4] M. Broy, Semantics of finite and infinite networks of concurrent communicating agents, Distributed Computing (1987) 2:13-31.

[5] J. Hirshfeld, A. Rabinovich, and B. A. Trakhtenbrot. Discerning causality in interleaving behavior. In A. R. Meyer and M. A. Taitsin, editors, *Proceedings of Logic at Botik 89*, volume 363 of *Lect. Notes in Computer Science*. Springer Verlag, 1989.

[6] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proceedings of the 16-th ACM Symposium on Principles of Programming Languages*, 1989.

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

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

[9] A. Mazurkiewicz. Semantics of concurrent systems: A modular fixed point trace approach. In *Advanced in Petri Nets*, volume 188 of *Lect. Notes in Computer Science*. Springer Verlag, 1984.

[10] A. Mazurkiewicz. Concurrency, modularity and synchronization. In *Mathematical Foundation of Computer Science*, volume 379 of *Lect. Notes in Computer Science*. Springer Verlag, 1989.

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

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

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

[14] P. Panangaden, and V. Shanbhogue Expressiveness of Indeterminate Dataflow Networks. Information and Computation, May 1992.

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

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

[17] A. Rabinovich, Non Modularity and expressibility for nets of relations (Extended Abstract). In *the Proceedings of Symposium on Logical Foundations of Computer Science, Lect. Notes in Computer Science.* Springer Verlag, 1992.

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

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

[20] 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.

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

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

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

[24] G. Winskel, Events in Computations, Ph. D. Thesis, CS Dept, University of Edinburgh, 1980.

[25] Wadge, W. Ashcroft, E., LUCID, the Data Flow Programming Language, Academic Press, 1985