On Nets, Algebras and Modularity

Alexander Rabinovich
IBM Research Division
T.J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598, USA
alik@watson.ibm.com

Boris A. Trakhtenbrot*
MIT Lab. for Computer Science
boris@theory.lcs.mit.edu
and
Dep. of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University, Tel Aviv, Israel 69978
trakhte@taurus.bitnet

Abstract

We aim at a unified and coherent presentation of net models for concurrency like Petri nets and dataflow networks from the perspective of modularity and substitutivity. The major goal is to achieve a better understanding of the links between modularity issues for nets and laws (or anomalies) in algebras of processes and algebras of relations. To this end we develop Mazurkiewicz's compositional approach which requires a careful analysis of homomorphisms from algebras of nets into algebras of processes and relations.

0 Introduction

0.1 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

*Supported by NSF Grant No. 9002826–CCR.

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

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

A conventional syntax (call it TEXTUAL as opposed to NET-syntax) is based on a signature Σ. Morever, a complex piece of syntax *expr* may be *uniquely* decomposed into simpler subpieces: $expr = op(expr_1, \cdots expr_k)$, where op is in Σ. 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 compositional semantics guarantees modularity. Compositional semantics may be characterized as a homomorphism from the Σ-algebra of the syntactical domain into the Σ-algebra of the semantical domain.

Typically, denotational semantics is formulated in compositional style and hence supports modularity. However, often one starts with an operational semantics which lacks 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 syntax is provided by some specific class NN of labelled graphs called *nets.* On the other hand, semantics is usually defined in an operational style through appropriate firing (enabling) rules. Though NN is not necessarily equipped with a signature Σ of operations (i.e. no algebra of nets must be assumed) the notions of *context, subnet* and *substitution* may make sense and therefore $R - substitutivity$ (in particular modularity) may be defined and investigated.

0.2 Historical Background

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

1. *Dataflow.* Substitutivity issues for dataflow nets were identified early and in a sharp way. The Kahn Principle [7] implies that dataflow nets over functional agents are $\equiv_{rel}$-substitutive . On the other hand, as Brock and Ackerman observed, $\equiv_{rel}$-substitutivity in general fails if nonfunctional agents (like MERGE) are also allowed. The celebrated counterexample from [3] illustrates this so called Brock-Ackerman anomaly.

2. *Petri Nets.* For elementary Petri nets modularity was established by Mazurkiewicz [8] following the pattern mentioned above wrt 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 textual syntax 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 in order to support homomorphism, these relations must hold also in the semantical domain as well. 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 leval) and *synchronization* of processes (at the semantical level); the nontrivial relations amount to commutativity and associativity of the operation. In the sequel [9] he formulated compositional semantics of this kind also for the more general classes of P/T-nets but without to compare 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. [2, 4, 5, 6, 11, 12, 13, 15, 16, 17, 18]. 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, 15, 16, 17]. In [5] we used the Mazurkiewicz algebraical approach to formulate an alternative compositional semantics for the token game semantics of P/T-nets. In this way we established modularity for this, more general class of nets. On the other hand, in [15, 16] where our main concern was about the phenomena around the Brock-Ackerman-anomaly, we did not rely on any specific algebraical arguments. An important conceptual and technical novelty we started in [16] and developed in [17] is the idea to consider semantics of nets of relations in addition to semantics of nets of processes. As a result of a careful comparison of these two kinds of nets we came very close to answering the following question: what nonfunctional agents may be used in dataflow nets without to produce anomalies?

As shown in [16], if such nontrivial agents exist they may implement only so called unambiguous relations. This seems to be too a strong restriction which cannot offer much to practice. However, the full answer to the question is an exciting challenge and we have more to say about that in the sequel.

0.3 Goals of the Paper

They are better explained after some preliminary comments about the conceptual and notational framework we are going to use.

A possible formalization of the models we consider is through triples:

$MM =_{def} < NN, DD, SEM >$, where
NN is the syntactical domain, a class of 'nets',

DD is the semantical domain, usually a class of 'processes',
SEM is a function from $NN \times ENV$ into DD. Here ENV is an appropriately defined class of 'environments'.

Syntax. A great diversity of nets is actually used in the literature. Roughly speaking the nets we consider are *bipartite* graphs as in the theory of Petri nets with the additional requirement that the set of all transitions (we call them *ports*) is divided into the set of visible ports and the set of hidden ports. It may happen that for the whole class of nets (we denote it as NN_1) modularity cannot be guaranteed. In order to regain modularity one has to consider subclasses of NN_1, which reflect reasonable restrictions on the topology of the net or on the status of visible/hidden nodes. We find the following restrictions enough representative: NN_2-nets without hiding, NN_4-nets without loops, NN_3-nets with exactly the internal ports hidden. In most of dataflow papers (including our [16]) one prefers to deal with simpler (nonbipartite) graphs in which edges do not necessarily have nodes, or may have nodes of different kinds. It is easy to see that these kinds of nets are shorthands of our bipartite nets and in particular of nets in NN_3. Summarizing we believe that our approach to nets is quite general.

Semantical domains. Here our choice is very specific and debatable. We consider only processes which are prefix closed sets of finite runs. This may be too a strong restriction. Yet it still allows to explain many phenomena concerning modularity and anomalies. But note that in addition to processes we consider also the semantical domain of (connected) relations, which are the behaviors of processes. These objects are interesting in their own (see [10, 16, 17]) but we use them here also for the explanation of $\equiv_{rel}$-substitutivity and anomalies.

Semantics. Our starting point is an operational semantics we call SEM_{proc} which provides meanings for nets of processes in the style of firing (enabling) rules. A semantics SEM_{rel} for nets of relations is derived from SEM_{proc}. In [16, 17] we analyzed different possible approaches and they provide evidence to the naturalness of the semantics SEM_{rel}.

In this paper we pursue two major interrelated goals.

The first is to develop our previous results to a level which presents in an unified and coherent way the status of different models from the perspective of modularity. To be more concrete one can imagine a table with 4 rows (corresponding to our four kinds of nets) and with 3 columns (corresponding to modularity for processes, modularity for relations and $\equiv_{rel}$-substitutivity). At each of the 12 intersections we would expect the characterization of those classes of processes or relations (if any!) which support the required version of modularity/substitutivity wrt the class of nets under consideration.

Our second goal is to achieve a better understanding of the links between modularity/substitutivity issues for nets and laws (or 'anomalies') in algebras of processes and relations. To this end, following Mazurkiewicz, we aim at a careful analysis of homomorphisms from algebras of nets into algebras of processes or relations. This may be illustrated by the following comparison with [8]. There, for nets without hiding, modularity is argued by the fact that process synchronization obeys the laws of commutativity and associativity. For other models we expect to discover in a similar way appropriate laws which support modularity (in particular - their violation spoils modularity).

0.4 Survey of Contributions

Let us now proceed with the survey of the paper and its main contributions.

Section 1 presents nets as syntax and also the concept of modular net semantics. This material is mainly folklore, but note the accurate definition of substitution (for nets some of whose nodes may be hidden!) and of substitutional classes of nets.

Sections 2-4 contain the definitions of input processes, input relations and of the semantical functions SEM_{proc}, SEM_{rel}. In general, processes are dealt with in a more or less routine way. However our treatment of relations is not routine and heavily relies on the notion of *kernel* which transfers on relations the idea of least fixed points. This notion appears in [17, 10] and comes close to Misra's 'smooth solution' [11].

The conceptual framework covered in sections 1-4 suffices for the formulation of the modularity (relational substitutivity) problems we investigate in this paper. It suffices also to formulate most of the facts (though not their proofs) according to the 4×3 - table we mentioned above. Not surprisingly (though we never met this fact in the literature) modularity for processes holds for all nets and all processes (Claim 2.1). The real problems arise with modularity for relations and for $\equiv_{rel}$-substitutivity; both fail if all nets are allowed. Moreover, they fail even for trivial subdomains of processes or relations. Here is where restrictions on the class of nets have to be considered. If hiding is not allowed (the case of class NN_2) or if the nets under consideration don't contain loops (the case of class NN_4) no anomalies appear for relations. This analysis shows that the real challenge is with nets which allow both hiding and loops but are still tractable. In our classification the appropriate candidate is just the class NN_3 which requires exactly the hiding of the internal ports of the net (additional fanin and fanout restrictions are imposed only to make the exposition readable). Note, that in most of works on dataflow such nets (or more precisely - their shorthands) are considered. Earlier in [16] we also investigated these nets; it appears that for them the two following tasks are reducible to each other:
Task 1. Find a class of processes which is relational substitutive.
Task 2. Find a class of relations which is modular.

Sections 5-8 contain the algebraical part of the paper.

In Section 5 the classes NN_1, NN_2, NN_3, NN_4 of nets are characterized as algebras with appropriate signatures. Moreover the corresponding nontrivial relations for their generators are explicitly formulated. Among them the most prominent are: the existential quantifier law - 5.2(4) and the looping law - 5.2(5).

Section 6 deals with algebra of processes to an extent which exceeds the direct necessities of our modularity issues (in particular we consider the union operation which is not in the signature of the net algebras). Nevertheless, we included claim 6.1 which shows that well known logical laws hold in the algebra of processes. Beyond of being a generally stimulating observation, this fact may be also useful for other related applications (e.g., for the proof of the generalized Kahn Principle as in [17]).

Section 7 deals briefly with the algebra of relations. Again we notice similarities of the operations in this algebra with logical operations, but unlike for processes these similarities are much more limited and exhibit anomalies.

Section 8 contains the main technical result (claim 8.3) which establishes the links between the algebras investigated in the previous sections and their relationship to the original semantical functions SEM_{proc} and SEM_{rel}. It extends the Mazurkiewicz compositional approach to a broad class of net models and paves the way to the discovery of modular models or to the prediction that they are impossible under given circumstances. Actually, that is how most of the claims in sections 1-4 may be proved. In particular a model $< NN_3,\ RR,\ SEM_{rel} >$ is modular iff the looping law holds in the class RR of relations. Recalling the connection between modularity for nets of relations and $\equiv_{rel}$-substitutivity (see Task 1 and Task 2 on the previous page) we can see that this fact opens the way to the full characterization of nonfunctional agents which avoid the Brock-Ackerman anomaly. However, the *explicit* description of *all* classes of relations which obey the looping rule appears to be a subtle task and will be the subject of a separate paper [14].

1 Nets

1.1 General Definitions

A *net* is an appropriately labeled bipartite directed graph with nodes of two kinds, pictured as circles and boxes and called respectively **places** and **ports**. The edges of the net are called **channels**. If there is a channel between port p and place pl they are said to be adjacent. 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. Channels connecting place pl to its input ports and output ports are numbered. This allows to refer to the first input channel of pl, to its second input channel, $\cdots$, first output channel etc. The difference between ports and places is relevant for the notion of subnet.

Definition 1 *A subgraph N_1 of N is considered to be a subnet of N if the set of its nodes consists of some places and all ports and channels adjacent to these places.*

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.

Output and internal ports are called **local** ports.

In the sequel we consider marked nets i.e. nets in which some ports are declared as visible ports; all the other ports are said to be hidden.

Labeling. Ports are labeled by port names. Different ports of a net are labeled by different names. Places are labeled by identifiers together with pair of natural numbers (rank). An identifier assigned to a place *pl* with n input ports and m output ports should have the rank $(n; m)$

The type of place *pl* is the set of names of port adjacent to it. We use the notation *Port(pl)* for the type of *pl*. A net with only one place and any number of ports is called **atomic**. $At(a_1, \cdots a_n; b_1, \cdots b_m)$ is a typical notation for an atomic net with place labeled by $At(n; m)$, with input ports labeled by $a_1, \cdots a_n$ and output ports labeled by $b_1, \cdots b_m$. Different ports of a net have different labels. Hence we may identify ports with their labels. We always assume in the sequel that no parallel channels are allowed in the net: given an arbitrary place and an arbitrary port in the net there may be no more than one channel which connects them. Therefore, in any atomic subnet $At_1(a_1, \cdots a_n; b_1, \cdots b_m)$ all the port labels $a_1, \cdots a_n, b_1, \cdots b_m$ are different.

Fig. 1 suggest itself. For example in N_2 place pl_1 has two adjacent ports. Port a is adjacent to places pl_1, pl_2, pl_3. As usual nets are to be considered up to isomorphism. Two nets are isomorphic if there is a bijection between them which preserves adjacency, visibility status of ports and also the labeling.

Here are some possible restrictions concerning hiding and the topology of directed nets:

1. *No hiding at all*
2. *No Confluence* - For every port there is at most one channel entering it.
3. *No Forks* - For every port there is at most one channel exiting it.
4. All internal ports are hidden.
5. *No Loops* - No directed cycles in the net.

1.2 Substitution

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

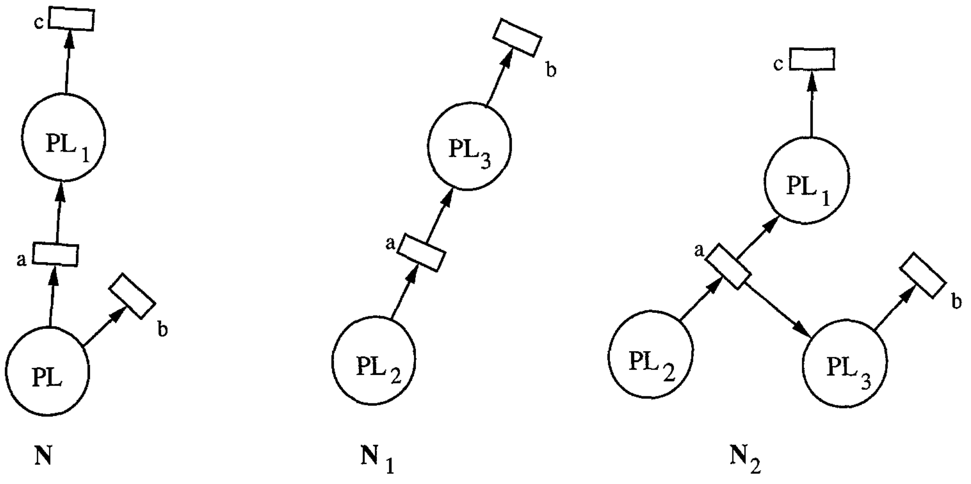

Figure 1: $N_2 = N[N_1/pl]$

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

2. No hidden port of N_1 is a port of N.

3. N_1 and N do not have common places.

The result of 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. $Ports(N_2) = Ports(N) \cup Ports(N_1)$

3. A port and a place are connected in N_2 if they are connected in N or in N_1 and the edges preserves their direction.

4. A port is visible in N_2 if it is visible in N.

5. All nodes inherit their labelling.

For example in Fig. 1 $N_2 = N[N_1/pl]$.

A class of nets is called **substitutional** if it is closed under substitutions.

Notations. In the sequel we will refer to some specific substitutional classes of nets and denote them as follows:

- NN_1 - all nets
- NN_2 - all nets with only visible ports

- NN_3 - all nets without forks, without confluences and exactly internal ports are hidden.
- NN_4 - all nets without loops.

NN_3 is a subclass of what we would be more interested in, namely the class of all nets with exactly the internal ports hidden. However, we impose the additional restriction for NN_3 in order to simplify the exposition.

If for classes NN_1, NN_2, NN_4 we require also that no confluences are allowed, then we obtain non substitutional classes.

Sometimes in the literature the class of nets without hiding, without confluences and without forks is considered. This class is not substitutional. See Fig. 1 in which $N_2 = N[N_1/pl]$, nets N, N_1 are in this class, but N_2 is not.

On the other hand the class without forks, without confluence and with all internal ports hidden is substitutional.

1.3 Modular Net Semantics

A model of net semantics is a triple $< NN, D, SEM >$ where: NN is a substitutional class of nets, D is an 'appropriate' semantical domain we do not specify here (In the sequel classes of relations or classes of processes are intended mainly). SEM is a function from $NN \times ENV$ into D. Here ENV is the set of environments; each environment is a mapping from (all) atomic nets into D which respects types and renaming (see definition 3 below). An *interpreted net* is a net together with an environment. Notations $< N, env >$ and $SEM(N, env)$ are used for an interpreted net and its semantics. Since the places of a net are uniquely identified with its atomic subnets we refer (by abuse of notation) to env in $< N, env >$ also as to a function from the places of N into D. As always a *net context* is a net with partial environment (an environment which assigns value not to all places of N). $N[pl]$ is a typical notation for the net with one hole pl.

Definition 2 (Modularity) *We say that model* $< NN, D, SEM >$ *is modular (or briefly - that semantics* SEM *is modular) iff* $SEM(N_1, env) = SEM(N_2, env)$ *implies that for arbitrary context* $N[pl]$

$$SEM(N[N_1/pl], env) = SEM(N[N_2/pl], env)$$

From the particular case when N_2 is atomic it follows that a modular semantics SEM has the following

Property: Assume that pl is a place in N and $SEM(N_1, env) = env(pl)$; then $SEM(N, env) = SEM(N[N_1/pl], env)$.

It is easy to see that if NN contains all atomic nets then this property is equivalent to modularity.

2 Processes

2.1 Basic Definitions

Let P be a set of ports and Δ be a fixed data set. A communication event over P is a pair $< port,\ d >$ with $port \in P$ and $d \in \Delta$. A linear run over P is a finite string of communications over P. A linear process of type P is a pair $(T,\ P)$, where T is a prefix closed set of runs over P. Note, that processes of different types might contain the same set of string; such processes are different. The type of a process is its important attribute; we use notations $ports(Pr)$ for the type of process Pr. On processes of a given type one considers the subset preorder: $Pr_1 \leq Pr_2$ iff every run of Pr_1 is a run of Pr_2.

Example 1 Buffer. Usually under 'buffer' one has in mind an automaton with one input port one output port; it reads values and outputs them according to the FIFO discipline. As a linear process a buffer with input port p and output port q (notation - $buf(p \rightarrow q)$) consists of all strings s which obey the condition: in every prefix of s the sequence of data communicated through q is a prefix of the sequence of data communicated through p.

Example 2 Labeled transition systems and linear processes A Labeled Transition System (LTS) of type P is an automaton whose alphabet (set of actions) is the set of communications over P and the special invisible action τ. It consists of:

- Set of states Q.
- Initial state $q_0 \in Q$.
- Transition Relation: a subset of $Q \times Alphabet \times Q$.

We use $q \xrightarrow{<p,d>} q'$ as a notation for a transition from state q via communication $< p, d >$ to state q'; we say that $< p, d >$ is enabled at state q if there is a transition $q \xrightarrow{<p,d>} q'$ for some q'.

An alternating sequense $q_0,\ a_0,\ q_1,\ a_1, \cdots a_{n-1},\ q_n$ of states of LTS T and actions of T is an *execution sequence* of T if q_0 is the initial state of T and $q_i \xrightarrow{a_i} q_{i+1}$ are transitions of T for $i = 0 \cdots n-1$. A run of T is the sequence of communications which is obtained from an execution sequence by deleting the states of T and τ actions. For every LTS T the process of the same type as T is assigned. This process consists of the runs of T. It is clear that the set of runs of T is a prefix closed set of strings. It is also clear that for every process Pr there corresponds a LTS whose set of runs consists of the strings of Pr.

2.2 Operational Semantics Sem_{proc} for nets of processes

Let us consider first operational semantics for a net of LTS.

Let N be a net with n places and let ρ be a function which assigns to every place pl_i of N a LTS of the same type as pl. N and ρ define the LTS T as follows:

- States of T are the tuples $(q_1, \cdots q_n)$, where q_i is a state of $\rho(pl_i)$.
- The initial state of T is the tuple of the initial states of $\rho(pl_i)$.
- The transitions of T are defined as follows:
  1. If $q_i \xrightarrow{\tau} q'_i$ is a transitions of $\rho(pl_i)$ then $(q_1, \cdots q_{i-1}q_i, q_{i+1}, \cdots q_n) \xrightarrow{\tau} (q_1, \cdots q_{i-1}q'_i, q_{i+1}, \cdots q_n)$ is a transitions of T.
  2. If in each of the places $pl_{i_1}, \cdots pl_{i_k}$ which are adjacent to the port p the communication $< p, d >$ is enabled at state $(q_1, \cdots q_{i-1}q_i, q_{i+1}, \cdots q_n)$, and $q_{i_1} \xrightarrow{<p,d>} q'_{i_1} \cdots q_{i_k} \xrightarrow{<p,d>} q'_{i_k}$ are transitions of $\rho(pl_{i_1}) \cdots \rho(pl_{i_k})$ then $(q_1, \cdots q_{i-1}q_i, q_{i+1}, \cdots q_n) \xrightarrow{\tau} (q_1, \cdots q'_{i_1}, \cdots q'_{i_k} \cdots q_n)$ is a transitions of T if p is a hidden port of N and $(q_1, \cdots q_{i-1}q_i, q_{i+1}, \cdots q_n) \xrightarrow{<p,d>} (q_1, \cdots q'_{i_1}, \cdots q'_{i_k} \cdots q_n)$ is a transitions of T if p is a visible port of N.

Definition 3 *A process* **environment** *pp is a mapping from atomic nets into processes which*

- *Respects types:* $pp(At(a_1, \cdots a_n; b_1, \cdots b_m))$ *has the same type as* $At(a_1, \cdots a_n; b_1, \cdots b_m)$.
- *Respects renaming: the processes* $pp(At(a_1, \cdots a_n; b_1, \cdots b_m))$ *and* $pp(At(a'_1, \cdots a'_n; b'_1, \cdots b'_m))$ *are the same up to appropriate renaming of their ports.*

Semantics of a net of processes Let $< N, pp >$ be an interpreted net of processes and let ρ be a function which maps the places of N into labeled transition systems such that the sets of runs of $\rho(pl_i)$ is the same as the process $pp(pl_i)$. The process semantics Sem_{proc} of $< N, pp >$ is the process assigned to the LTS for $< N, \rho >$.

It is easy to see that $Sem_{proc}(N, pp)$ does not depend on the choice of ρ. Therefore process semantics is well defined.

2.3 Input Processes

Definition 4 *Pr is an* **input process** *if its ports are divided into input ports and local ports with the only demand that if p is declared as an input port it should be 'input buffered' in the following sense: Assume s in Pr; then Pr contains also all strings one can construct via the following operations:*

- *Input extension. Extend s appending to the right arbitrary many communications through p.*
- *Input anticipation. If a communication $< p, d >$ follows immediately after a communication through a port different from p, permute them.*

The following remarks explain the intuition behind these conditions. Let Pr' be a process of type $p' \cup P$. Consider the process Pr specified by the net N with hidden port p' and two places: one for Pr' and another for $buf(p \longrightarrow p')$ (here p is not a port of Pr). Then port p in Pr satisfies input extension and input anticipation conditions.

If a process is obtained by the construction above, we say that its port p contains a buffer. It is easy to check that a process is input buffered at ports $p_1, \cdots p_k$ iff it contains buffers at these ports. Additional remarks about input bufferness will be given in section 6.2 when we consider operations on processes.

Example 3 *$buf(p \to q)$ is a linear process with input port p and local port q.*

Example 4 *(Rudimentary Processes [15]). Start with an arbitrary run s over ports $P \cup Q$. Let $Prefix(s)$ be the closure of s under prefixes. Finally, close $Prefix(s)$ under input extension and input anticipation wrt ports in P. The resulting process $Rudim(s, P, Q)$ is called the rudimentary process generated by s, P, Q. It is an input process with input ports P and local ports Q.*

From now on when we refer to an interpreted net of processes we will have in mind that its environment assigns to the atoms input processes and to an atom At with input ports $p_1, \cdots, p_k$ and output ports $q_1, \cdots, q_m$ the environment assigns a process with input ports $p_1, \cdots, p_k$ and local ports $q_1, \cdots, q_m$. It is easy to check that the process Pr specified by such an interpreted net is an input process wrt to the set of visible inputs in N. Hence, the general notion of semantics for a net of processes consistently restricts to semantics for nets of input processes.

Claim 2.1 *(Modularity of SEM_{proc}). Let PP_1 be the the class of all input processes. Then $< NN_1, PP_1, SEM_{proc} >$ is a modular model.*

For the proof see later section 8.3. As a straightforward consequence we mention:

Corollary 2.2 *Let $< NN, PP, SEM_{proc} >$ be a model with arbitrary substitutional set NN and arbitrary set PP of input processes; then this model is modular.*

3 Implementing Relations

3.1 Basic Definitions

Let D be a domain and P be a set of (port) names. A **port** relation R of type P over D is a subset of D^P. We will designate the type P of R as $ports(R)$. Below we will consider port relations over stream domains.

Definition 5 *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.

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 $a_1 \leq a_2 \leq ...$; then $x \leq a_n$ for some n.

For a finite set of ports P, the finite elements of $STREAM(\Delta)^P$ are functions which map ports into finite streams. Let s be a run of process Pr. The behavior of run s at port p is the stream of data communicated through p in s. Therefore, to each run there corresponds a function from ports to $STREAM(\Delta)$. And to a process Pr of type P there corresponds a port relation of type P which we denote by $rel(Pr)$. We say that process Pr **implements** this relation. We say that processes Pr_1, Pr_2 are relationally equivalent (notation $Pr_1 \equiv_{rel} Pr_2$) if $rel(Pr_1) = rel(Pr_2)$. Among the processes which implement a relation R there is a maximal process (i.e. each other process implementing R is its subset). This maximal process is said to be **fat** and is denoted by $fat(R)$. We also introduce a preorder $\leq_{rel}$ on processes: $Pr_1 \leq_{rel} Pr_2$ if $rel(Pr_1)$ is a subset of $rel(Pr_2)$.

3.2 About $\equiv_{rel}$-substitutivity issues for SEM_{proc}

Consider a model $< NN, PP, SEM_{proc} >$ where NN is a substitutional set of nets and PP is a set of processes. We know already what it means that such a model is modular. Say that it respects $\equiv_{rel}$ (or that it is $\equiv_{rel}$ substitutive) if the following holds: Assume that two interpreted nets $< N_1, pp_1 >$ and $< N_2, pp_2 >$ in this model specify processes Pr_1, Pr_2 which implement the same relation (i.e. $rel(Pr_1) = rel(Pr_2)$) and that they are both substitutable in some context. Then they are replaceable by each other without changing the relation of the overall net. Similarily one defines 'respecting $\leq_{rel}$' (or $\leq_{rel}$ substitutivity): require that if ($Pr_1 \leq_{rel} Pr_2$ then replacing $< N_1, pp_1 >$ by $< N_2, pp_2 >$ may only increase the relation of the overall net. Clearly $\leq_{rel}$ substitutivity implies $\equiv_{rel}$ substitutivity. The Brock-Ackerman example (Brock-Ackerman anomaly) is a warning that substitutive reasoning of this kind is generally impossible; nevertheless, it still does not exclude specific cases when this is possible.

Given a substitutional set of nets NN and a set of processes PP the closure of PP under NN consists of all processes which can be specified by nets from NN over processes from PP. Say that PP is modular wrt NN if $< NN, closure(PP), SEM_{proc} >$ is a modular model. In a similar way we refer to PP as being $\leq_{rel}$-substitutive wrt NN.

Looking for $\leq_{rel}$ substitutive sets of processes we prefer to deal with sets PP of processes which have enough computational power [15]. The formalization is in terms of powerful sets. PP is said to be a powerful set if it contains at least all the rudimentary processes (see example 4 in 2.3).

Here is a slightly rephrased version of our result in [16], adapted to the notations of this paper:
Assume that PP is a powerful set of processes which is closed under NN_3. Then the model $< NN_3, PP, SEM_{proc} >$ is $\leq_{rel}$ substitutive iff all the processes in PP are fat.

One direction of this claim is easy. Note (1) SEM_{proc} is monotonic wrt inclusion of processes. (2) for fat processes, $Pr_1 \subseteq Pr_2$ iff $rel(Pr_1) \subseteq rel(Pr_2)$. Hence, if all processes in PP are fat then the model $< NN_3, PP, SEM_{proc} >$ is $\leq_{rel}$ substitutive. The second direction that a modular powerful set of processes contains only fat processes is more subtle and its proof is based on full abstractness.

What powerful sets of processes are $\leq_{rel}$-substitutive wrt NN_1, NN_2, NN_3, NN_4?

Claim 3.1 *1. NN_1. No powerful set is $\leq_{rel}$-substitutive wrt NN_1.*

2. NN_2. A powerful set is $\leq_{rel}$-substitutive wrt NN_2 iff it consists of only fat processes.

3. NN_3. A powerful set is $\leq_{rel}$-substitutive wrt NN_3 iff its closure under NN_3 consists of only fat processes.

4. NN_4. Each set of processes is $\leq_{rel}$-substitutive wrt NN_4.

Comment. (Comparing classes NN_2 and NN_3.) If a set PP consists of only fat processes then its closure under NN_2 will also consist of only fat processes. That is not the case for NN_3. Hence, it is easy to give examples of $\leq_{rel}$-subsitutive (and powerful) sets for NN_2; just take all fat input buffered processes. On the other hand, it is not even simple to check that the closure of the rudimentary processes under NN_3 consists only of fat processes. Therefore, the construction of all powerful $\leq_{rel}$-substitutive sets is a difficult problem. This issue is better handled in connection with modularity for relations (see 4.3).

4 Connected Relations

4.1 Basic Definitions

Since processes are prefix closed their relations may not be arbitrary.

We are going to characterize briefly this particular kind of relations, we call *connected relations* (see [17, 10]).

Definition 6 *We will write $x_1 \ll x_2$ (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 begins with $\perp$ and $x_i \ll x_{i+1}$ for all $i < n$.*

Let R be a subset of D. *chain*(R) denotes the set of all strict chains contained in R. The kernel of R (denoted $Kern(R)$) is the subset of R such that x is in $Kern(R)$ if it belongs to a chain in *chain(R)*.

Definition 7 *A relation R is called* **connected** *if $R = Kern(R)$.*

Obviously, $Kern(R)$ is the maximal connected subset of R. Every connected relation over a stream domain consists only of finite elements.

Example 5 *(kernel vs least fixed point) Consider the relations: $S =_{def} \{y = f(x,y)\}$ and $S' =_{def} \{y < f(x,y)\}$. Assume that f is the constant function which returns the stream 00. Then S consists of all pairs $< x, 00 >$ and its kernel is obviously empty. On the other hand for arbitrary continuous f: $Kern(S')$ consists of all finite x, y such that $y < h(x)$, where $h(x) =_{def} lfp.\lambda y.f(x,y)$.*

Definition 8 *Given a relation R of type P (i.e., $R \subset STREAM(\Delta)^P$) we say that R increases at port p if the following holds: Assume that x, y are finite elements in $STREAM(\Delta)^P$) which differ only on p and moreover $x(p) \leq y(p)$. Then $x \in R$ implies that $y \in R$.*

Similarly one defines 'R decreases in p'. We will refer to a relation R as to an input relation if its ports are divided (someway!) into input ports and local ports with the only requirement that R increases on each of its input ports. Notations like $R(\vec{x}; \vec{y})$ are used to point on the vector $\vec{x}$ of input ports and on the vector $\vec{y}$ of local ports.

Example 6 *$buf(p \rightarrow q)$ implements the relation R we designate as $p \geq q$. It contains only finite elements and $x \in R \Leftrightarrow x(p) \geq x(q)$. Note that this relation increases in p and decreases in q.*

It is easily seen that if p is an input port of Pr then $rel(Pr)$ increases on this port. Hence $rel(Pr)$ may be considered as an input relation with the same inputs as Pr.

Fact 4.1 *1. R is a connected relation iff it is implemented by a linear process.*

2. R is an input relation with input ports P and local ports Q iff it is implemented by input process with input ports P and local ports Q.

4.2 Nets of Relations and their Semantics

Relational environments are defined similarly to process environments. Let rr be a relational environment. Given the interpreted net $< N, rr >$ choose a process environment pp such that for each place pl in N the process $pp(pl)$ implements the relation $rr(pl)$. Now consider the relation S implemented by the process $SEM_{proc}(N, pp)$. Since a relation may be implemented by different processes neither pp nor S are uniquely determined by $< N, rr >$

Fact 4.2 *[16, 17] There is an extreme environment pp which returns the maximal among all possible S; namely, this is the environment which assigns to each pl the fat implementation of rr(pl).*

Definition 9 *The maximal relation S achievable in this way is called the relational semantics of the net and is denoted by* $SEM_{rel}(N, rr)$.

Hence, $SEM_{rel}(N,\ rr) = rel(SEM_{proc}(N,\ fat(rr)))$.

4.3 Modularity of Sem_{rel} and $\leq_{rel}$ substitutivity of Sem_{proc}

Given a substitutional class of nets NN and a set of connected relations RR. We define 'RR is modular wrt NN' in the same way as for processes in 3.2. A set RR of relations is said to be powerful if it contains all the rudimentary relations, i.e. those relations which are implemented by rudimentary processes (see example 4 in section 2.3).

There is a simple relationship between modularity for relations and $\equiv_{rel}$-substitutivity for processes.

Claim 4.3 *Let RR and PP be corresponding sets of relations and fat processes, i.e.* $Pr \in PP$ *iff* $Pr = fat(R)$ *for R in RR. Then PP is* $\equiv_{rel}$*-substitutive wrt* NN_3 *iff RR is modular wrt* NN_3.

This claim is the starting point for improvements which show that problems about rel-substitutivity may be reduced to problems about modularity for relations.

What sets RR of relations are modular wrt NN_1, NN_2, NN_3, NN_4?

Claim 4.4

1. NN_1. *No powerful set RR is modular wrt* NN_1.
2. NN_2. *Every set RR is modular wrt* NN_2.
3. NN_3. *A powerful set RR is modular wrt* NN_3 *iff the corresponding set of processes* $fat(RR)$ *is* $\leq_{rel}$*-substitutive wrt* NN_3.

4. NN_4. *Every set RR is modular wrt* NN_4.

Comment. Claims 3.1.3 and 4.4.3 provide the reductions between the following tasks:

1. Find powerful sets of processes which are $\leq_{rel}$-substitutive wrt NN_3.
2. Find powerful sets of relations which are modular wrt NN_3.(Such sets will be directly characterized later through claim 8.4)

Indeed, if RR is a modular and powerful set of relations, then according to claim 4.4.3, the set $fat(RR)$ of processes is powerful and $\leq_{rel}$-substitutive. On the other hand, if PP is a powerful and $\leq_{rel}$-substitutive set of processes then by claim 3.1.3 it consists only of fat processes. Therefore, PP coincides with $fat(rel(PP))$ and is $\leq_{rel}$-substitutive. Hence, by claim 4.4.3, $rel(PP)$ is modular.

5 Algebra of Nets

5.1 Net Constructors

Below we consider a set Σ of operations on nets which allow to construct complex nets from more elementary ones. For all these operations labelling of nodes is unchanged.

Combination. N_1 and N_2 may be combined if they do not have a hidden port with the same name. The set of nodes in the resulting net is 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. Ports inherit their visibility status; edges inherit their directions and numbering.

Aggregation: is combination of nets which do not have common port names. (neither hidden, nor visible).

Sequential composition (notation *seq*) is combination of two nets N_1, N_2 such that every common port name is the name of a visible local port in N_1 and the name of a visible input port in N_2.

Hiding. If p is a visible port in N it becomes hidden in $\exists p.N$.

Note that for all operations above the set of atomic subnets of resulting net is the union of the sets of atomic subnets of components. The following operations do not possess this property.

LOOPing of a local port y and an input port x which are visible in a net N.

The operation $LOOP(y \to x)$ **in** N is defined as following:

1. Delete x from N.
2. Connect y to all places which were connected to x.

3. The visibility status of all ports is unchanged.

looping (note the low case spelling). $loop(y \to x)$ in N is defined as $LOOP(y \to x)$ in N, but the status of y changes from visible to hidden.

Simultaneous $LOOP(\vec{y} \to \vec{x})$ in N and $loop(\vec{y} \to \vec{x})$ in N are defined in a similar way.

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).

Relying on the signature Σ 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 $(At(;a,b) comb At_1(a;c))$ and $(LOOP(a \to a')$ in $(At(;a,b) aggr At_1(a';c)))$ describe the net N in Fig. 1. If two terms t_1, t_2 of NET describe the same net, we say that they are graph equivalent and write $t_1 \equiv_{graph} t_2$.

5.2 Equivalences in NET

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

1. combination is commutative and associative.
2. aggregation is commutative and associative.
3. $\exists p \exists q.N = \exists q \exists p.N$
4. $\exists p.\ (N_1 comb N_2) = (\exists p.\ N_1) comb N_2$, provided p is not visible in N_2.
5. $loop(\vec{y}_1 \to \vec{x}_1)$ in ($loop(\vec{y}_2 \to \vec{x}_2)$ in N) $= loop(\vec{y}_1,\ \vec{y}_2 \to \vec{x}_1,\ \vec{x}_2)$ in N
6. $loop(y \to x)$ in $(N_1 aggr N_2) = (loop(y \to x)$ in $N_1) aggr N_2$, provided y and x are not visible ports of N_2.

5.3 Constructor sets for specific classes of nets

Say that the class NN of nets is generated by the subsignature $\Sigma' \subset \Sigma$ if it contains exactly the nets generated from atomic nets by the operations in Σ' (in other words - the nets expressible in the language NET with the use of only Σ')

Claim 5.1 *1. The classes NN_i below are generated as follows:*

(a) (All nets.) NN_1 is generated by comb and hide.

(b) *(All nets with only visible ports.)* NN_2 *is generated by comb.*

(c) *(All nets without forks, without confluences and exactly internal ports hidden.)* NN_3 *is generated by aggr and loop.*

(d) *(All nets without loops.)* NN_4 *is generated by aggr, seq and hide.*

2. *(Standard systems of equivalences.) For each of the classes* NN_i *above and their corresponding constructor set* Σ_i *there is a standard system of equivalences from which all other equivalences are provable by equational reasoning.*

   (a) *For* NN_1*: equivalences 1,3,4;*

   (b) *For* NN_2*: equivalences 1;*

   (c) *For* NN_3*: equivalences 2,5,6;*

   (d) *For* NN_4*: omitted;*

6 Algebra of Processes

6.1 Preliminary Remarks

We consider below the special interpretation of Σ (the signature of net constructors) wrt processes (see 6.2). Σ_{proc} will designate the set of these operations on processes.

We preserve the terminology and notations used wrt nets except for combination, to which there corresponds synchronization ($||$) of processes. All the definitions implicitly include an appropriate classification of the ports (in the result of the operation) into input and local ports exactly as for the corresponding constructors. It is easy to check that the ports declared as input ports indeed obey the input buffering condition. We consider also union of processes.

As an immediate consequence of the interpretation Σ_{proc} one can use the syntax of NET for specification of processes.

6.2 Operations on Processes

First we consider operations on processes which correspond to the signature Σ of the net constructs.

Synchronization (notations: $||$)

$$ports(Pr_1||Pr_2) = ports(Pr_1) \cup ports(Pr_2)$$
$$s \in Pr_1||Pr_2 \text{ iff for } i = 1,2$$
$$s|ports(Pr_i) \in Pr_i$$

where $ports(Pr)$ is the type of process Pr and $s|A$ is the notation for the string one gets from s by deleting all events which are not on ports A.

Aggregation. In the case when Pr_1 and Pr_2 do not have common ports, their synchronization is called aggregation.

Hiding. $\exists\, p.\ Pr$ results in the process of type $ports(Pr) - p$; its strings are obtained from the strings of Pr by deleting all occurrences of communications on p.

Next we consider two versions of the looping operation. Note that we use for them upper cases notations (when the local port is not hidden) and lower case notation (when the local port is hidden).

LOOPing of a local port y and an input port x of process Pr.

$LOOP(y \to x)$ in $Pr =_{def} \exists x.(Pr \| buf(y \to x))$

looping. $loop(y \to x)$ in $Pr =_{def} \exists y.(LOOP(y \to x)$ in $Pr)$.

Another useful operation on processes is
Union. For processes Pr_1, Pr_2 of the same type, $Pr_1 \cup Pr_2$ inherits this alphabet and contains all strings in Pr_1 and in Pr_2.

Remark about the relevance of input bufferness. Let Pr be a process and p be its port. One can show that Pr is input buffered at p (see definition 4) iff for any port r not in Pr the process $\exists p.Pr \| buf(r \to p)$ is the same as the process obtained from Pr by renaming p by r. Therefore, in input processes a buffer is attached to every input port.

In our definition of the looping operations we explicitly rely on buffers. The input buferness is needed later only to show that the semantics based on aggregation and LOOPing coincides with the semantics based on synchronization. For example, net N_1 in Fig 1 can be described as $At_1(;a)combAt_2(a;b)$ and as $LOOP\ (a \to a')\ in\ At_1(;a)aggrAt_2(a';b)$. If an environment pp assigns to At_1 and At_2 input buffered processes, then these two terms will specify the same process in pp; otherwise these terms might specify different processes.

In the sequel under a process we have in mind an input process.

6.3 Some Laws

In order to characterize the algebras of processes we notice similarities between the logical operations conjunction, disjunction and existential quantifier on one hand and the operations synchronization, union and hiding for processes on the other hand. Let t be a first order term which uses only conjunction, disjunction and existential quantifiers. In addition to the usual logical interpretations of such terms one can consider also their process interpretations following a way similar to that we used in section 6.1 for terms in NET. For example, $(At_1(b', c) \wedge At_2(a, b))$ is interpreted in logic as the conjunction of the relations assigned by a logical environment to symbols

At_1, At_2. In the process algebra this term is interpreted as the synchronization of the processes assigned by a process environment to symbols $At_1(b', c)$, $At_2(a, b)$.

Given two terms t_1, t_2 of the same type $\{a_1 \cdots a_n\}$; say that t_1 implies t_2 in logic if the formula $\forall a_1 \cdots a_n(t_1 \rightarrow t_2)$ is first order valid formula. The following claims are valid for arbitrary not just input processes.

Claim 6.1 *(Relationship of process algebra to logic).*

If t_1 implies t_2 in logic then the process specified by $(t_1,\ pp)$ is a subset of the process specified by $(t_2,\ pp)$ in arbitrary process environment pp.

Corollary 6.2 *All basic equivalences for net constructors (see 5.3) hold in the algebra of processes, i.e., for every constructor set Σ_i considered above and terms t_1 and t_2 over Σ_i the equivalence $t_1 \equiv_{graph} t_2$ implies that for each process environment pp the interpreted terms (t_1, pp) and (t_2, pp) define the same process.*

Proof: The equivalences 1, 3, 4 from section 5.2 hold in logic and hence by claim 6.1 we obtain immediately the equivalences for combination and hiding. For other operations it may be inferred from their definition based on synchronization and hiding. □

7 Algebra of Connected relations

7.1 Preliminary Remarks

As for processes we consider below the special interpretation of Σ (the signature of net constructors) wrt relations. Σ_{rel} will designates the set of these operations on relations.

We preserve the terminology and notations used wrt nets except for combination, to which there corresponds strong conjunction ($\underline{\&}$) of relations. In addition to Σ_{rel} we consider also union (disjunction) of relations.

As an immediate consequence of the interpretation Σ_{rel}, one can use the syntax of NET for specification of relations. Let rr be a relational environment and let t be an arbitrary term in NET; then the pair $< t, rr >$ is an interpreted term whose meaning, denoted (t, rr), is a port relation which is fully determined by the environment rr and the interpretation Σ_{rel} of the net constructor symbols.

7.2 Operations on Relations

Given $x \in D^P$ and $x_1 \in D^{P_1}$, assume that $P_1 \subseteq P$ and for every port p in P_1 the equality $x_1(p) = x(p)$ holds; in this case we say that x_1 is the **projection** of x onto P_1.

First we consider the operations join and disjunction.

Join. Let R_1 be a relation of type P_1 and let R_2 be a relation of type P_2. The join of R_1 and R_2 is the relations of type $P_1 \cup P_2$ defined as follows: $x \in R_1 \& R_2$ if the projection of x on P_1 is in R_1 and the projection of x on P_2 is in R_2.

Disjunction. Let R_1 and R_2 be relations of the same type P. $R_1 \cup R_2$ is the relation of the type P which denotes the union of R_1 and R_2.

Disjunction of connected relations is a connected relation. But the result of the join of connected relations is not always a connected relation.

Now we list the operations in Σ_{rel}.

Strong Conjunction - (notation $\underline{\&}$). Let R_1 be a relation of type P_1 and R_2 be a relation of type P_2. The strong conjunction of R_1 and R_2 is the kernel of their join.

Example 7 *Consider the system of equation S_1 and the corresponding system of inequalities S_2.*

$$S_1 = \begin{cases} y = & f(x,z) \\ x = & y \end{cases} \quad S_2 = \begin{cases} y \leq & f(x,z) \\ x \leq & y \end{cases}$$

The solutions of S_1 is $R_1 = \{(x,y,z) : \ x = y = lfp\lambda x.f(x,z)\}$.

The strong conjunction of the two inequalities in S_2 is $R_2 = \{finite\ (x,y,z) : \ x \leq y \leq lfp\ \lambda x.f(x,z)\}$

Aggregation. In the case when R_1 and R_2 do not have common ports their strong conjunction is called aggregation.

It is easy to see that aggregation of connected relations coincides with their join.

Hiding. $\exists p.R$ is the relation of type $ports(R) - \{p\}$ which consists of projections of elements of R on these ports.

Again as for processes we consider two versions of looping: without and with hiding of local ports.

LOOPing of a local port y and an input port x of relation R.

$LOOP(y \to x)$ in $R =_{def} \exists x.Kern(R\underline{\&}(x \leq y))$.

$loop(y \to x)$ in $R =_{def} \exists y.LOOP(y \to x)$ in R.

7.3 Some Laws and Anomalies

As for processes we notice similarities between the logical operations conjunction, disjunction and existential quantifier on one hand and the operations strong conjunction, disjunction and hiding for relations. However, the algebra of connected relations is not rich as the algebra of processes. Some laws are valid; in particular strong conjunction is commutative and associative, hiding is commutative. But note equivalence 4 (from section 5.2); we refer to it in the sequel as $\exists$-rule:

$\exists p.\ (N_1 comb N_2) = (\exists p\ in N_1) comb N_2$, provided p is not visible port of N_2.

The rule is not valid for the set of all connected relations; in other words, for this set there holds $\exists$-anomaly. Also equivalences 5 and 6 (from section 5.2) fail. Hence, for connected relations there is no analog of corollary 6.2 we established for processes in section 6.3

8 Modularity and Robustness

8.1 Term Semantics

Sometimes (see [4, 18]) when referring to net semantics $SEM(N, env)$ what one really has in mind is term semantics $(t,\ env)$, where t belongs to some chosen set TN of descriptions of the net N. In such a case one has to make sure that for all t_i in TN the meaning of (t_i, env) is the same. Otherwise the net-semantics is not well defined.

In particular, given an interpreted net $< N, pp >$, consider the set TN of N's descriptions which perform first the combination of all atomic subnets and after that all the hidings. Due to the commutativity and associativity of process synchronization and of process hiding one can use interpreted terms $< t, pp >$ with t in TN for a well defined semantics $< N, pp >$. The same remark holds for strong conjunction and hiding wrt relations and hence for a well defined semantics of nets of relations.

Fact 8.1 *Semantics defined this way coincides with SEM_{proc} for processes and with SEM_{rel} for relations*

But what about other descriptions for (N, env). Do they provide also the same meaning as (t, pp) and (t, rr) for t in TN?

8.2 Compositional Semantics

Consider one of the sets NN_i of nets (see 5.3) equipped with its constructor set Σ_i. Below $t_1,\ t_2, \cdots$ are terms in NET which use only constructors from Σ_i; PP and RR denote some sets of input processes and relations respectively which are supposed to be closed under Σ_{proc} and Σ_{rel} respectively.

Definition 10 *The semantical model $< NN_i, PP, SEM >$ is compositional (SEM is a compositional semantics from NN_i into PP) iff for each environment (types respected!) SEM induces a Σ_i homomorphism from NN_i into PP.*

Corollary 8.2 *Every compositional model is modular.*

Claim 8.3 *1. A compositional semantics from NN_i into PP is possible (and if possible is unique) if there holds the following* **robustness condition:** *Given arbitrary terms t_1, t_2 over Σ_i the equivalence $t_i \equiv_{graph} t_2$ implies that for every environment env in PP the processes specified by $(t_1,\ env)$, $(t_2,\ env)$ are equal.*

2. Under the conditions above SEM coincides with SEM_{proc}.

3. If $< NN_i, PP, SEM_{proc} >$ is a modular model then SEM_{proc} is a compositional semantics from NN_i into PP.

Similarly for semantics from NN_i into RR.

8.3 Modular Models

Relying on corollary 8.2 and on claim 8.3 we are going to characterize some modular models $< NN_i, PP, SEM_{proc} >$ and $< NN_i, RR, SEM_{rel} >$. To this end we survey situations when the robustness condition holds.

a) Robustness holds for all models $< NN_i, PP, SEM_{proc} >$.

That is due to corollary 6.2, and it proves modularity of SEM_{proc} (see claim 2.1 from section 2.3).

For relations the situation is quite different. This can be shown directly by counterexamples, but is also evident from the $\exists$-anomaly (see section 7.3), which violates a basic equivalence for $\{comb, hide\}$. Therefore, it makes sense to look for more specific situations in which robustness and hence modularity hold. The following cases are easy and prove claim 4.4 (see section 4.3) for NN_1, NN_2 and NN_4.

b) NN_2 (No hiding). Robustness holds for arbitrary RR. That is because the only relevant equivalences are commutativity and associativity for both comb and $\underline{\&}$.

c) NN_4 (No loops). Robustness holds for all relations. We omit the details.

d) NN_1 (arbitrary nets). Robustness fails for every powerful set RR. Actually, the $\exists$-rule (see 7.3) is violated in such set.

Hence, if we want to allow both loops and hiding and at the same time to have robustness we must restrict the set NN_1. An instructive case is the set NN_3 with the constructors $\{aggr,\ loop\}$. The basic equivalences 2 and 6 (see 5.2) wrt $\{aggr,\ loop\}$ hold in general for all relations. There is still one kind of basic equivalences which should be explicitly postulated:

The looping law: For each relation R in the class RR there holds

$$loop(\vec{x}_1 \to \vec{y}_1)\ in\ (loop(\vec{x}_2 \to \vec{y}_2)\ in\ R) =$$
$$= loop(\vec{x}_2 \to \vec{y}_2)\ in\ (loop(\vec{x}_1 \to \vec{y}_1)\ in\ R) =$$
$$= loop(\vec{x}_1, \vec{x}_2 \to \vec{y}_1, \vec{y}_2)\ in\ R$$

Therefore we conclude:

Claim 8.4 *A model* $< NN_3, RR, SEM_{rel} >$ *is modular iff for all* R *in* RR *there holds the looping law.*

A powerful model of this kind is provided by the class of all functional relations. Recall [16] that a relation $R(\vec{x};\ \vec{y})$ is functional if for some continuous function f

$$R(\vec{x};\ \vec{y}) \text{ iff } \vec{x} \text{ and } \vec{y} \text{ are finite elements and } \vec{y} \leq f(\vec{x}).$$

Note that such a relation is not only input increasing but it is also decreasing wrt all local ports. (In our previous papers [16, 17] we used the terminology 'observable relations' for relations with this property). The looping law for functional relations is a consequence of the well known fact that for functions the least fixed point operators commute. Note that usually the proof of modularity for functional relations is based on the Kahn Principle for dataflow nets. Here we inferred it directly from the robustness condition. Are there other nontrivial classes RR which obey the looping equivalence and hence are modular? We know that there are such classes. According to claim 4.4.3 in section 4.3 these classes correspond exactly to powerful classes of processes which are $\leq_{rel}$-substitutive, i.e. avoid Brock-Ackerman anomaly.

9 Concluding Remarks

9.1 Comments to 8.2

It is not difficult to understand that processes and relations are not exceptions and that claim 8.3.1 (and the definitions it is based on) can be generalized to a broad class of domains. For such a domain D and for an appropriate interpretation of the signature of net constructors one can consider the robustness condition and its relationship to modularity. First, observe that under the robustness condition a net semantics is induced in a natural way. For example, in the cases of processes we would define $semrobust(N, pp)$ as the value (t, pp), where t is an arbitrary description of N over NN_i, the point being that this definition does not depend on the particular choice of the description t for N. This definition of semantics may be adapted to 'arbitrary' domain D and, what is more, one can show that the semantics will be modular. In the case of processes and relations the use of Σ_{proc} and Σ_{rel} implies also 8.3.2 and 8.3.3 i.e., the robust semantics coincides with SEM_{proc} and SEM_{rel} respectively. In the general case at this stage we do not have any a priory net semantics to compare with. But assume that we started with a modular model $< NN_i, D, SEM >$; is it the case that the signature Σ_i may be interpreted in D in such a way that robustness holds? It appears that in the general case some additional assumptions about SEM are needed. In the particular case of processes or relations these assumptions are implicit in the requirements about input buffering and input increasing.

9.2 The impact of hiding

It seems clear that nets without loops are too poor to support an interesting theory of dataflow networks. On the other hand, it makes sense to look to what extent the theory may (or should) be developed without hiding. In particular: do there exist interesting models without hiding for which the Kahn Principle and its generalization [2] hold?

It seems that in [2] Abramsky had in mind just such model. Here is a quotation from [1]: 'I didn't forget about hiding in my paper. I left it out because I didn't consider it germane for the Kahn Principle. It is no need for me to build hiding into my definition of network composition ... It is well known that this (hiding) spoils the nice properties of of composition-this is why it isn't done e.g. in CCS and CSP'.

Unfortunately, there is some slight inconsistency in [2] which can be easily repaired without affecting the results of the paper. This can be done in two ways. One of them would preserve the definition of 'process P computes function f' chosen in [2], but would require hiding internal ports of the net. The other one seems to correspond to Abramsky's idea of justifying Kahn Principle without building on hiding. It amounts to weaken the definition of 'process P computes function f'.

However, now there may be different processes which implement different relations, but compute the same functions. Therefore, unlike the case of relational substitutivity it would not make sense to distinguish between different relations to which there corresponds the same function (an idea advocated by those who insist on considering complete computations). Hence, instead of $\equiv_{rel}$-substitutivity one should consider a weaker equivalence between processes. But then anomalies would appear without hiding exactly as they appeared wrt $\equiv_{rel}$ substitutivity in the presence of hiding. As a matter of fact, the original Brock-Ackerman example illustrates this kind of anomaly without hiding.

The moral: though one can justify the Kahn Principle in models without hiding, this approach does not rescue from anomalies.

9.3 Further Research

1. We considered processes and relations over stream domains. The generalization to F-domains [17] is straightforward.

2. Technically more involved seems to be the accurate extension of the the theory to other sets and algebras of nets. But we do not see any serious difficulties on this way.

3. Deepening the knowledge about the algebras of processes and relations. We conjecture that 'logical laws' for processes (see section 6.3) may be essentially improved. On the other hand, despite the stigma of anomalies, the algebra of relations is worth to be explored carefully. Though anomalies cannot be avoided, facing them may still be possible in many situations.

4. This paper as well as our previous works [15, 16, 17] is based on a simple model of processes which does not take into account such discriminating features as branching, terminating, etc.. It seems that ignoring these features is not harmful and may be even useful as long as one can develop the theory without them. But finally we have to face the challenge of analyzing more sophisticated models which take into account, for example, complete runs [2, 3, 6, 11, 18].

Acknowledgements

We are indebted to Antoni Mazurkiewicz whose compositional approach to Petri nets provided the initial stimulus to this work. We would also like to thank Samson Abramsky and Albert R. Meyer for stimulating discussions.

References

[1] S. Abramsky. e-mail correspondence.

[2] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In M. Main, A. Melton, M. Mislove, and D. Scmidt, editors, *Mathematical Foundations of Programming Languages Semantics*, volume 442 of *Lect. Notes in Computer Science*. Springer Verlag, 1990.

[3] J. D. Brock and W. B. Ackerman. Scenarios: A model of non-determinate computation. In *Formalization of Programming Concepts*, volume 107 of *Lect. Notes in Computer Science*, pages 252–259. Springer Verlag, 1981.

[4] M. Broy. Semantics of finite and infinite networks of concurrent communicating agents. *Distributed Computing*, 2, 1987.

[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 J. L. Rosenfeld, editor, *Information Processing 74*. North Holland Publ. Co., 1974.

[8] 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.

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

[10] A. Mazurkiewicz, A. Rabinovich, and B. A. Trakhtenbrot. Connectedness and synchronization. In D. Bjorner and V. Kotov, editors, *Images of Programming (dedicated to the memory of A. Ershov).* North Holland Publ. Co., 1991.

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

[12] D. Park. The fairness problem and nondeterministic computing networks. In J. W. de Bakker and J. van Leeuwen, editors, *Proceedings, 4th Advanced Cource on Theoretical Computer Science.* Mathematisch Centrum, 1982.

[13] V. R. Pratt. On composition of processes. In *Proceedings of the Ninth Annual ACM Symposium on Principle of Programming Languages*, 1982.

[14] A. Rabinovich. in preparation.

[15] A. Rabinovich and B. A. Trakhtenbrot. Nets of processes and data flow. In *Proceedings of Rex Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lect. Notes in Computer Science.* Springer Verlag, 1988.

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

[17] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In *International Conference on Automata, Languages and Programming*, volume 443 of *Lect. Notes in Computer Science.* Springer Verlag, 1990.

[18] E. W. Stark. A simple generalization of Kahn's principle to indeterminate dataflow networks. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics for concurrency*, Workshops in Computing. Springer Verlag, 1990.