Fundamenta Informaticae 28 (1996) 183–195
IOS Press

On the Power of Compositional Proofs for Nets: Relationships Between Completeness and Modularity

B.A. Trakhtenbrot
School of Mathematical Sciences
Sackler Faculty of Exact Sciences
Tel Aviv University
Ramat Aviv 69978, Israel
e-mail: trakhte@math.tau.ac.il

This paper is dedicated to the memory of Professor Helena Rasiowa, a prominent scientist and a committed friend through changing times.

Introduction

Recall first that the concepts mentioned in the title can be explained informally as follows:

1. Modularity: if two systems have the same meaning then one is replaceable by the other in any context and the meaning of the overall system is unchanged.
2. The principle of compositional verification asserts [6]:
   The specification of a program should be verified on the basis of specifications of its constituent components without knowledge of the internal construction of those components.
3. Finally, completeness is to be understood as completeness of proof systems:

Nets of communicating processes are a popular model of concurrency and much of the research dedicated to them covers two subjects: modularity and completeness of compositional verification

(i) Lack of modularity is an unfortunate circumstance that one tries to avoid in different ways, for example via appropriate restrictions on the nets and/or processes under consideration. From this perspective, we carried out in [RT_1, RT_2, RT_3] an extensive study of the foundations of modular semantics for dataflow and general nets.

(ii) Compositional verification in general (and in particular in the case of nets) amounts often to the formulation and use of appropriate proof systems. Hence the crucial question: are these proof systems complete? Over the last years numerous papers were devoted to compositional verification of concurrent systems, and in particular to verification of networks of communicating processes (see [6] and [5] for a more detailed bibliography).

Notice that there are different notions of completeness. What is considered in [5] (and in most of the sources quoted there) is called in [6] *compositional completeness* and sometimes (also in [5]) – *backward soundness*. In this paper, we use "*completeness*" in a sense which comes very close to what is called in [6] *modular completeness*.

At the beginning the two subjects evolved independently of each other. Neelam Soundaararajan was the first to point on possible links between them. Moreover, in [4] he suggested that the conceptual framework and the techniques from the "modularity chapter" might be beneficial to improve and even to correct some existing completeness results.

The paper [5] was my first reaction to [4] and follows Soundaararajan's suggestions. [5] was conceived as a critical comparison of the conceptual framework developed in [RT1, RT2, RT3] with that used in the verification community. This comparison made explicit three anomalies which may limitate the power of the proof systems under consideration. It suggested how to cure some vulnerable points in the existing literature through careful treatment of these anomalies.

The intention of the present paper is to elucidate at a more abstract (and hopefully more persuasive) level the relationship between modularity of net semantics and completeness of compositional proofs for nets. This is to be achieved by a separation of concerns which is not reflected in common terminology. The following two preliminary observations may be suggestive.

Observation 1. Modularity and compositionality seem to be close notions; often these terms are even used synonymously. Therefore, it is only natural to expect close relationship between modular semantics and compositional proofs. However, in our analysis "modularity" will relate directly with "completeness" and only indirectly with "compositionality".

Observation 2. Nets are widely associated with concurrency because they help to visualize the communication structure of systems. However, in the sequel nets play the role of a non-standard syntax, whose semantics is not committed apriori to any idea of concurrency. As a matter of fact, the relationship between modularity and completeness wrt verification of nets will be analyzed prior to any semantics of concurrency.

Since there is no a-priori connection of completeness and modularity to concurrency, we aim at a survey of the area that makes evident the role of each of the three factors and the way they interact. A part of what is needed for this task is already implicit in the papers [RT1, RT2, RT3,T], which contain in addition much material less relevant for our current subject. As it is often the case with "semantical" papers, they are not free from massive and painful terminology and notation. Unfortunately this seems to be almost inevitable in a full report for two reasons. First, we have to cover a great diversity of concepts from semantics, proof theory, concurrency, and net-theory. Second – in many cases, there is still a lack of stable and generally accepted notation.

In the present sketchy survey we try to avoid cumbersome formalization, and to use often a very informal style, giving up mathematical rigor in favor of (hopefully) persuasive hints. Also, definitions of some important notions (mostly concerning syntax and semantics of nets) are omitted; references are given to existing sources.

The main statements are summarized in Theorems. The auxiliary "Facts" are mostly adapted folklore (Facts 0,1,2,3,4) or borrowed from our earlier papers (Facts 6,7,8). Proofs are completely omitted.

We hope that despite these shortcomings, our presentation provides guidance in the area and contributes to a clearer understanding of the relationship between modularity, completeness and concurrency.

The subsequent exposition is organized in three stages which allow to gradually accumulate concepts and notation and the relevant facts about them.

Stage 1. Context Free Syntax

Recall that compositional verification of a program presumes its suitable decomposition into components. Such decomposition into components is quite obvious in the case of the context free syntactic structures that we consider in this stage. Moreover, we have no commitments

to specific semantic constructs, in particular to concurrency constructs. Nevertheless, already at this abstract level it is possible to define central notions like: compositional proof, completeness, strong consequence. This material is partially folklore.

We also formulate necessary and/or sufficient conditions for completeness which are summarized in a preliminary criterion of completeness (Theorem 1.1). It is worth mentioning that despite its generality and abstractness this criterion helps to identify some inaccuracies in the existing literature (more details in [5]). Finally a sufficient condition for completeness is formulated in Theorem 1.2. It is based on "denesting" techniques and anticipates the analysis in the next stage.

Stage 2. Nets

Decomposition into components and compositional proofs make sense for more complex structures as well; in particular – as mentioned in [6] – for flowcharts with named "black boxes". We take even a more liberal view and broadly use nets as syntax. Here the main reference is [1] where a great diversity of nets is considered; among them – the version chosen for this paper, namely, directed nets with hidden internal ports. It turns out that most of the notions and constructs from stage 1 can be adapted and reformulated in a natural way, (though not always as simple as one might expect) using this kind of nonstandard syntax. In addition one has to take into account the notion of modularity for net-semantics and its impact on denesting techniques. This allows to reveal in Theorem 2.1 the crucial relationship between completeness of Proof System and modularity for nets of specifications. No idea of concurrency appears here and, hence no connection is assumed between nets and concurrency.

Stage 3. Verifying Nets of Processes

Here the concern is about port processes and the specification of their input-output behavior via port relations. Also the concurrent interpretation of the net constructs enter the play.

We start with the more general situation when arbitrary port processes are allowed. Then we pass to the more restricted class of buffered processes and observable relations which are adequate to handle asynchronous message passing.

It turns out that the links between modularity and completeness analyzed earlier from a very general and abstract viewpoint can be applied to this particular interpretation. This paves the way to the use of concepts and techniques from [RT1, RT2, RT3] in the investigation of completeness issues. Actually that was the main goal of [5].

At the previous stages the rules of a proof system and the constructions are based on, are presented in a very abstract, axiomatic way. Now at a more concrete level one has to do with specific algebras of agents and algebras of specifications; it is only natural to require that the proof rules and constructions be formulated by means of these algebras. The specific expressibility issues which arise in this connection were analyzed in [1]. In particular modularity for nets of specifications can be characterized exactly through suitable equations in these algebras.

Therefore, ultimately three phenomena converge: modularity, completeness (these two identified already earlier) and "benignness" of specification algebra.

1. Context Free Syntax

1.1. Notation and Terminology

They are best explained wrt a model

$$Mod = \langle AG,\ SPEC,\ sat,\ SIGN \rangle$$

with

1. AG - the set of agents
2. $SPEC$ - the set of specifications
3. $sat \subset AG \times SPEC$ – the satisfaction relation between agents and specifications
4. $SIGN \triangleq \{N_1, N_2, \ldots\}$ – the signature of basic operators on agents.

We refer to the algebra $\langle AG, SIGN \rangle$ of agents in Mod and define in a standard way the meaning of terms over $SIGN$ in agent environment.

In the sequel we refer to the following classification of terms:

(i) Basic terms; $N_i(var_1, var_2, \ldots var_k)$ with $N_i \in Sign$ and all object variables different.
(ii) Flat terms: may be basic but many occurrences of the same variable are allowed as well.
(iii) Nested terms $=_{\text{def}}$ non-flat terms.

Usually one considers *Interpreted Assertions* (briefly – *Satisfactions*) of the following kinds:

1. Simple Satisfactions

$$term\ (ag_1, \ldots, ag_k)\ sat\ spec \qquad (simp)$$

Conditional Satisfactions

$$(ag_1\ sat\ spec_1, \ldots, ag_k\ sat\ spec_k) \rightarrow (simp)$$

For simplicity we ignored above typing restrictions. Actually the algebra $\langle AG, SIGN \rangle$ might be many sorted. Then specifications are also assumed to be sorted. Moreover, *sat* may hold only for an agent and a specification which have the same sort (type). This precisely will be the case at Stage 3 where port automata play the role of agents, and relations – those of specifications; accordingly – the port types are there quite relevant.

There is yet another notational complication. Namely, instead of referring to arguments in terms and in satisfactions by their positions, we would better to refer to them by their names. In other words, instead of sequences of agents (specifications) actually one should use respective environments. For example,

$$\pi\ sat\ \rho \longrightarrow [[t]]\pi\ sat\ spec,$$

is the alternative notation for the conditional satisfaction:
Here:

(i) π is an agent environment
(ii) ρ a specification environment
(iii) $\pi\ sat\ \rho$ is shorthand for: For all $x\big(\pi(x)\ sat\ \rho(x)\big)$.

As usual, $[[t]]\pi$ is the meaning of t under environment π.

When there is no danger of confusion, we use the sequence-format.

Definition 1.1. (Induced partial preorders and equivalences on agents and on specifications).

1. $ag_1 \preceq ag_2 \triangleq \forall spec.\ (ag_1\ sat\ spec \Rightarrow ag_2\ sat\ spec)$
2. $spec_1 \preceq spec_2 \triangleq \forall ag.\ (ag\ sat\ spec_1 \Rightarrow ag\ sat\ spec_2)$
3. $\approx_{spec}$ an $\approx_{ag}$ are the equivalences induced by the preorders above.

It is clear that the following inference rule is sound.

The Strongest Adaptation Rule:

$$\frac{ag\ sat\ spec,\ spec \preceq spec'}{ag\ sat\ spec'}$$

1.2. Semantical Consequence

Definition 1.2. (Connectors and constructors) A specification *spec* is a *semantical consequence* of $spec_1 \dots spec_n$ wrt an operation *op* on agents iff the satisfactions ag_1 *sat* $spec_1 \dots ag_n$ $op(ag_1, \dots ag_n)$ *sat spec* .

If *spec* is a consequence of $spec_1 \dots spec_n$ wrt *op* we write $Cons(op; spec_1 \dots spec_n;\ spec)$.

A connector $Con(op; ...)$ associated with *op* is a relation over specifications which should be sound i.e.

$$Con(op;\ spec_1, \dots, spec_k;\ spec) \quad \text{implies} \quad Cons(op;\ spec_1, \dots, spec_k;\ spec)$$

(notice: the inverse implications is not assumed)

We say that *spec* is a *strong semantical consequence* of $spec_1 \dots spec_n$ wrt operation *op* iff for any spec':

$$Cons(op; spec_1 \dots spec_n;\ spec') \text{ holds if } spec \preceq spec' \text{ holds}$$

An operator $\widetilde{op}$ on specifications is said to be a *specification constructor* (a strong specification constructor) conjugated with the operator *op* on agents iff

(i) for all $spec_1, \dots spec_n$ it returns a consequence (a strong consequence) of $spec_1, \dots spec_n$ wrt *op*.

(ii) the constructor $\widetilde{op}$ is monotonic:

$$\text{spec}_i \preceq spec_i' \quad \text{implies} \quad \widetilde{op}(spec_1, \dots, spec_k) \preceq \widetilde{op}(spec_1', \dots spec_k')$$

(for strong constructors monotonicity follows from (i).

From the definitions and the adaptation rule it follows:

Fact 0. $$Cons(op; spec_1, \dots, spec_n; spec) \Longleftrightarrow \widetilde{op}(\text{spec}_1, \dots, spec_n) \preceq spec$$

for strong constructor $\widetilde{op}$: only implication from right to left is guaranteed in general for a constructor.

Example 1.1. (Connector for Hoare Logic). By abuse of notation we do not distinguish below programs (syntax) from state transformers (semantics)... Also consider for Hoare-assertion $\{p\}Prog\{q\}$ the satisfaction format of *Prog sat* $\langle p, q\rangle$.

Here is a well known connector for the operator *seq* on programs:

$$Con(seq; \langle p_1, q_1\rangle, \langle p_2, q_2\rangle; \langle p, q\rangle) =_{\text{def}}$$
$$(p \to p_1)\&(q_1 \to p_2)\&(q_2 \to q)$$

Example 1.2. (Specification-constructor for processes.) Consider the operator $\|$ (synchronization) on input-output processes and specification of processes via input-output relations (see section 3 later). The conjunction of relations is a (possible) conjugate operator for $\|$. However, notice that it is not a *strong* constructor. The genuine conjugate strong constructor is what we called in [3] strong conjunction (notation $\underline{\&}$).

Example 1.3. (Strong conjugates for the identity operator.) Assume that $Id \in SIGN$ is interpreted as the identity operator on agents. Clearly, the identity operator on specifications is a conjugate constructor; but in general it is not a *strong* conjugate. The genuine strong constructor is what we called in [3] the Kernel operator. (In [3] Kernel was considered only for the particular case when the role of specifications is played by relations.)

The concepts and notations above are easily adapted to arbitrary terms over $SIGN$. For example

Definition 1.3. Assume that t is a term and ρ is a specification environment.

1. We say that *spec* is a semantical t-consequence of ρ (notation $Cons(t; \rho;\ spec)$) if for any agent environment π there holds

$$\pi \ sat \ \rho \quad \text{implies} \quad [\![t]\!]\pi \ sat \ spec.$$

1.3. Compositional Proof Systems (CPS)

A Proof System (PS) replaces semantical consequence by appropriate provable consequence (derivation). The (meta)notation

$$\pi \ sat \ \rho \vdash [[t]]\pi \ sat \ spec \tag{1}$$

means: the succeedent is provable from the antecedent.

We write $Prov(t; \rho;\ spec)$ if the statement (1) holds for all π.

Definition 1.4. (Compositionality, soundness, completeness)

(i) A Proof System is considered to be compositional iff it obeys the following criterion: whenever (1) holds for some π then $Prov(t; \rho; spec)$ holds as well.

(ii) Soundness of the CPS means that

$$Prov(t;\ \rho;\ spec) \quad \text{implies} \quad Cons(t;\ \rho;\ spec)$$

(iii) Completeness amounts to the inverse implication.

Usually one has in mind two closely related formats of CPS: the connector based format and the constructor based format. A short presentation follows.

Connector Based Proof Systems

Such a system is defined through a set of connectors, each one associated with an operation from the signature $SIGN$.

Each connector induces the corresponding derivation rule:

$$\frac{A_1 \ sat \ spec_1, \ldots A_k \ sat \ spec_k;\ Con(N;\ spec_1, \ldots, spec_k;\ spec)}{N(A_1, \ldots, A_k) \ sat \ spec}$$

Constructor Based Proof Systems

To each $N \in Sign$ there corresponds a conjugate specification constructor N' and the induced rule:

$$\frac{A_1 \ sat \ spec_1, \ldots, A_k \ sat \ spec_k}{N(A_1, \ldots, A_k) \ sat \ N'(spec_1, \ldots, spec_k)}$$

In addition, the system includes an Adaptation Rule which may happen to be weaker than the strongest one. Its format is

$$\frac{A \ sat \ spec,\ spec \leq \text{spec}'}{A \ sat \ spec'}$$

where $\leq$ is a partial order on specifications which refines the preorder $\preceq$ from Definition 1.1.

Obviously, to the Examples 1.1-1.2 there correspond the following rules:

1) Connector Format

$$\frac{\{p_1\}Prog_1\{q_1\}, \{p_2\}Prog_2\{q_2\}, (p \to p_1)\&(q_1 \to p_2)\&(q_2 \to q)}{\{p\}Prog_1 \ seq \ Prog_2\{q\}}$$

2) Constructor Format

$$\frac{P_1 \ sat \ R_1,\ P_2 \ sat \ R_2}{P_1 \| P_2 \ sat \ R_1 \& R_2}$$

with Adaptation Rule:

$$\frac{P \ sat \ R, \ R \subseteq R'}{P \ sat \ R'}$$

Notice that here $\subseteq$ refines the preorder $\preceq$.

There is a consensus about how one has to use the rules (in either of the formats) in order to prove satisfactions and/or to generate provable consequences. We omit the details but mention the following

Fact 1. Both kinds of PS obey the compositionality criterion.

1.4. Complete CPS

For expository convenience, we assume below three restrictions which still leave open the main problems we have to face but allow us to ignore more routine details.

Restriction 1. Although the connector format is slightly more general, we focus only on the constructor format.

Restriction 2. The equivalence between specifications which is induced by the preorder in Definition 1.1 is assumed to be the identity relation.

Restriction 3. The adaptation rule is used in its strongest form.

Under these restrictions a constructor based PS is uniquely characterized by the choice of the conjugate constructors, i.e. by the choice of the conjugate interpretation for the signature *Sign* (via operators on specifications). One can define in a standard way semantics for terms over *Sign* in this conjugate interpretation. From now on, use notation $[[t]]$ for agent semantics of t and notation $[[t]]'$ for the conjugate specification semantics of t. Using these notations one can characterize provability as follows:

Fact 2. $Prov(t; \rho; spec) \Longleftrightarrow [[t]]'\rho \preceq spec.$

Remark 1.1. The characterization of provability provided by this lemma is actually all we need to know in the sequel about the inference mechanism incarnated in the constructor based PS. Hence the reader who is not aware about the consensus referred to at the end of Subsection 1.3 (or does not share it) can adopt this characterization as a definition.

According to Definition 1.4, Fact 2 immediately implies

Fact 3. A constructor based PS is complete iff for all terms t there holds

$$Cons(t; \rho; spec) \Longleftrightarrow [[t]]'\rho \preceq spec.$$

Further elaboration on completeness criteria is facilitated by the following observation about strong constructors.

Fact 4. Let op be an agent operator; then the strong conjugate $\widetilde{op}$ is unique up to equivalence on spec's (hence – according to Restriction 2 – it is unique).

Say that a constructor based PS is strong iff the conjugates for all basic agent operators in *Sign* are strong.

Fact 5. If PS is strong then for all flat terms t there holds:

$[[t]]'$ is a (actually – the!) strong conjugate constructor for $[[t]]$

Warning. That is not necessarily the case for nested terms t.

Finally, we have the following

Theorem 1.1. (Completeness Criterion). *A constructor based PS is complete iff for each term t there holds:*

$$[[t]]' \quad \textit{is a strong specification constructor for} \quad [[t]]$$

Corollary 1.1. (From Theorem 1.1 and the preceding Warning)
In order for a constructor based PS to be complete, it is necessary but not sufficient that it be strong.

Comments. This corollary allows to identify the "vulnerable" points cured in [5] as follows:

(i) The *PS* criticized there were not always strong.
(ii) Even in cases when strength was guaranteed (or assumed) it was not evident that the completeness criterion persists for nested terms (systems).

We end this subsection with a sufficient condition for completeness which relies on denesting. Despite the fact that it is not a necessary condition, it is useful in "realistic" situations and allows a clear paraphrase for net-syntax.

Let *denest* be a (syntactical) operation over terms such that $denest(t)$ is a flat term which is *similar* to t in the following sense:

(i) both have the same set of free (object) variables.
(ii) each of these variables has the same type in both terms.
(iii) the terms themselves have the same type.

Say that *denest* is robust if it preserves semantics, i.e. iff t and $denest(t)$ have the same meaning in each environment.

Returning to constructor based *PS* it will often be the case that the operation *denest* is known to be robust for agent semantics, whereas its robustness for the conjugate specification semantics has still to be investigated.

Theorem 1.2. *Assume that a constructor based PS is strong and that denest is robust for agent semantics. Then PS is complete iff denest is robust also wrt the conjugate specification semantics.*

2. Nets

2.1. Syntax and Semantics

Following the terminology of Petri nets a net is a labeled bipartite directed graph. There are nodes of two kinds, pictured as circles and boxes are called respectively places and ports. It is assumed that given a port p and a place pl there may be no more than one edge e which connects them.

Respectively, p is an input (output) port for pl if e exits from p (enters p).

The ports of a net N are partitioned into

(i) Input (output) ports of N: ports with no entering (exiting) edges.
(ii) Internal ports: all the other ports. A port is external for N if it is either an input or output port of N.

Places serve as (typed!) argument holders, the type of a place pl is the pair $\langle P, Q\rangle$ where $P(Q)$ is the set of labels assigned to the input (output) ports of pl. On the other hand, the type of a net N is the pair $\langle I, O\rangle$ where $I(O)$ is the set of labels assigned to its input (out- put) ports. A net N with k places can be interpreted as a typed net operator $[[N]]$ with k arguments. Hence, one can consider signatures $SIGN$ consisting of nets (syntax) and also interpretations of such signatures (semantics). Roughly speaking a net whose places are labeled by variables plays the same role that a flat term plays in common syntax. Hence it

is quite evident how to define semantics for "flat" (and in particular for "basic" nets). The role of common nested terms is played by net expressions. The formalization of their syntax and semantics seems to be conceptually straightforward but its accurate realization is more troublesome than one might expect. (As a matter for fact, it was never published). In [1] the notation

$$N[N_1/pl]$$

is used for the syntactical operation: "substitute the net N_1 for the place pl of the net N", which is legitimate if N_1 and pl have the same type. Simultaneous substitutions are handled similarly.

Net expressions are defined inductively starting with flat nets. Further, one increases nesting through the construct

$$let\ (H_1 = \ell_1 \quad and \cdots H_k = \ell_k) \quad in \quad H$$

where H_i, H are net expressions, ℓ_i-labels of places. The similarity to the substitution notation is apparent but not completely trivial (because one has to substitute for labels instead for places). It suggests a well defined syntactical operation *denest* which, transforms every net expression into a flat net expression.

In order to define semantics for net expressions one has first to agree on the choice and on the interpretation of the signature $SIGN$. So we assume that:

(i) $SIGN$ is a substitutional class of nets, i.e. a class closed under the *syntactical* operation of substitution.
(ii) appropriately typed domains AG and $SPEC$ are chosen. Namely, each element in the domain has as a type a pair:
⟨ set of labels for input ports, set of labels for output ports.⟩
(iii) for each net N in $SIGN$ there is a fixed interpreting net operator on agents.

Then semantics can be extended in a natural way to arbitrary net expressions.

2.2. Completeness Issues

Now, the machinery from Stage 1 can be adapted through the use of net expressions instead of conventional terms. In particular to each net expression there corresponds (via the conjugate semantics) a conjugate operator on specifications.

So far the analogy to Stage 1 works well, except that it ignores the crucial issue of modularity for nets.

Although nets are not standard syntax there is a natural notion of subnet which brings to the notion of contexts in hierarchical net structures.

Definition 2.1. A net semantics is said to be modular iff two nets with the same meaning are replaceable by each other in every context of a net expression.

Modularity is a nice property and usually it is supposed that net semantics for agents is indeed modular. It turns out that this does not guarantee that the conjugated net semantics for specifications is also modular. This discrepancy is manifested in the following fact:

Theorem 2.1. *A CPS is complete iff it is strong and iff the conjugate semantics on specifications is modular.*

This claim is a direct consequence of Theorem 1.2 (appropriately adapted to net semantics) and the following

Fact 6. Net semantics is modular iff *denest* operation on is robust

3. Verifying Nets of Processes

3.1. Port Processes and Port Relations

We are going to apply the concepts and facts of the previous section wrt models with a particular semantics which interprets:

(i) Agents as port processes
(ii) Specifications as port relations
(iii) Net of agents as the process provided by the net of communicating component-processes.

First we briefly recall some basic notation and terminology and explain what it means for a process to satisfy a relation.

Let P be a set of ports which is partitioned into two sets I (input ports) and O (output ports) and let $DATA$ be a fixed data set.

1. A communication event over P is a pair $\langle port,\ d\rangle$ with $port \in P$ and $d \in DATA$.
2. A linear run over P is a finite string of communications over P.
3. A linear process of type $\langle I; O\rangle$ is a prefix closed set of runs over P.
   Note, that processes of different types might consists of the same set of strings; such processes are different.
4. A stream (of data) is a string (maybe empty) over $DATA$.
5. A snapshot of type P is a function which assigns to each port in P a finite or empty stream.
6. A port relation of type P is a set of snapshots of type P.
7. 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 *snapshot*, and to the process Pr of type P there corresponds a port relation of type P, which we denote by $rel(Pr)$.
8. A process Pr **satisfies** a relation R iff $rel(Pr) \subseteq R$.
9. $Pr_1 \| Pr_2$ is the notation for the composition of processes Pr_1 and Pr_2 i.e. their synchronization wrt common ports. $\exists a Pr$ is the notation for "hide port a in Pr".
10. The result provided by a net of processes may be computed as the composition of these processes with subsequent hiding of the internal ports.

In order to guarantee Restriction 2 from Subsection 1.4 we also adopt the following restriction on relations:
A port relation R is a legitimate specification iff there exists a process Pr such that $rel(Pr) = r$. (In [3] such relations are said to be *connected*.

Omitting other details we mention only the following two facts which are relevant for our subject:

Fact 7. Net semantics for processes is modular.

Fact 8. For each net of processes among the conjugate nets of relations there is a (unique!) strong conjugate net of relations.

Comment. Uniqueness is guaranteed by Fact 4 and the restriction above on relations.

Hence it is natural to consider strong PS (they exist – according to Fact 8) and to ask whether they are complete:

Theorem 3.1. *For the models under consideration a strong PS is complete iff the conjugate net semantics for relations is modular.*

The following comments are in order.

a) The restriction on relations guarantees that Theorem 3.1 holds even in the following improved form: instead of using the strongest adaptation rule (see Stage 1) one can use the adaptation rule:

$$\frac{Pr \text{ sat } R,\ R \subseteq R'}{Pr \text{ sat } R}$$

in which $\preceq$ is replaced by inclusion of relations instead of $\preceq$.

b) It turns out that the conjugate net semantics for port relations is equivalent to the original definition of the concept "net of relations" in [RT2, RT3]. In these papers "nets of relations" appear in connection with the Brock-Ackerman anomaly for dataflow nets. We investigated there the question of how semantics for nets of relations should be defined and showed that there is a unique reasonable semantics. No explicit relationship was assumed there to compositional proofs and to completeness. The convergence of the two approaches gives evidence for the adequacy of the original definition; for other arguments to this effect see [RT2, RT3].

c) In general, as observed in [2] nets of relations lack modularity. But if one considers only nets over appropriate subclasses of relations, it may happen that the semantics is modular. This fact stimulated efforts to discover and to describe modular classes of relations i.e. such restricted classes for which the net operation is modular. Now, in view of the links to compositional proofs the accumulated knowledge about modular classes can be used to identify situations in which completeness holds.

3.2. Nets of Specifications vs Algebra of Specifications

There is an evident lack of symmetry in our generic model *Mod*. Whereas agents are elements of the algebra $\langle AG, SIGN\rangle$ no a priori algebraic structure is mentioned in *Mod* wrt *SPEC*. And this is so despite the fact that usually specifications are drawn from a source for which some well defined operations usually make sense. For example at Stage 3, the use of logical operations "and", "or",etc... (or some of their transparent adaptations) upon relations makes sense. Briefly – the specifications might be taken from an existing algebra $\langle SPEC, SIGN^s\rangle$ which is important for its own sake, prior to the intention of using its elements as specifications against agents from *AG*. Hence the following question arises:

Expressiveness. Is it possible to express in the original algebra $\langle SPEC, SIGN^s\rangle$ the strong constructors $[[N]]'$ on specifications?

Assume that the answer is negative. How should one enrich the original algebra $\langle SPEC, SIGN^s\rangle$ in order to achieve expressiveness? Obviously, one could add the $[[N]]'$ themselves or their disguised versions as new operations, but that is just what we would like to avoid. The intention is to discover (to invent?!) "benign" operations and a "'benign" algebra of specifications which:

(i) make sense for their own right without a priori commitments to specific verification issues,
(ii) allow to express the strong constructors,
(iii) allow to express the partial order between specifications to be used in the adaptation rule (actually this was already achieved at stage 3 with "$\subseteq$" replacing "$\preceq$",
(iv) allow to formulate a modularity criterion for nets of specifications via inspiring equations in the algebra of specifications.

Since (i) is informal and (iii) is already guaranteed – we have to focus on (ii) and (iv). In [3] we defined the kernel operation on relations (see Example 3) and argued that its inclusion into the algebra of specifications provides a satisfactory solution to the "benignness" task.

Even a a more persuasive solution is possible when additional restrictions are imposed on the processes and relations allowed by the model. In particular this is the case of reasonable assumptions about the semantics of input-output as suggested by the mechanism of

asynchronous message passing. In ([RT1, RT2], [5]). the respective formalization is given in terms of buffered processes and observable (monotonic) relations. The looping operation on observable relations ([2]) is also defined there. It turns out that looping is a natural generalization of the least fixed point operator and that together with usual logic, it provides a "benign" algebra. But unlike fixpoints for looping the following "anomaly" may happen:

Simultaneous looping may produce a result different from that produced by nested looping.

Theorem 3.2. (Expressiveness). *Nets of observable relations are expressible in the algebra OB of observable relations over logic and looping.*

Theorem 3.3. (Modularity criterion). *Net semantics is modular wrt relations from a subalgebra OB' of the algebra OB iff in OB' there holds the law (identity):*

Simultaneous looping is equivalent to nested looping.

Concluding Remarks

1. Levels of abstraction The underlying issues are better tackled on two levels. The first (Sections 1-2) does not essentially depend on specific models (nor on concurrency). The second one (Section 3) is committed in a crucial way to nets of communicating processes and requires a deeper penetration into their semantics.

2. The power of Compositional Proof Systems (CPS) This can be characterized already at the first level of abstractions via *strength* and *completeness*. A strong CPS guarantees provability for flat systems; this does not extend in general to nested systems. Hence the idea of *robust denesting* as a condition sufficient for completeness.

3. Nets For this non-standard syntax denesting is a natural and well defined syntactical operation. Moreover, robustness of denesting is equivalent to modularity. In this way connection is established between modularity and completeness.

4. Automata vs. processes At the second level, nets of automata may look more attractive than nets of processes. However, as long as we are interested in the linear behaviour of automata, processes already contain all the information relevant for the input-output specification of automata. (Indeed, in [5] we started with automata and passed from there to processes. Unfortunately, in [5] there are some annoying inaccuracies in the presentation of automata. They may be easily detected and corrected. In any case, they do not affect the further treatment of the subject.)

5. Nets of relations The most subtle mathematics in dealing with our topics occurs around modularity for nets of relations ([RT1, RT2, RT3]). At level 1, this modularity leads to compositionality. At level 2 – to nontrivial algebras of specifications.

Acknowledgement

I wish to thank Arnon Avron, Shai Geva, Yoram Hirshfeld and Alex Rabinovich for useful discussions and Mrs. Gila Markowitz for skillful and patient collaboration in editing the text.

References

[1] Rabinovich A., Trakhtenbrot B.A., On nets, algebras and modularity. In *International Conference on Theoretical Aspects of Computer Software*, Volume 526 of *Lecture Notes in Computer Science*, Springer-Verlag, 1991.

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

[3] Rabinovich A., Trakhtenbrot B.A., Communication among relations. In *ICALP90*, Volume 443 of *Lecture Notes in Computer Science*, Springer-Verlag, 1990.
[4] Soundararajan N., Completeness issues in trace based systems, Technical Report, Department of Informatics, University of Oslo, 1991.
[5] Trakhtenbrot B.A., Compositional proofs for networks of processes, Fundamenta Informaticae, vol. 20, pp. 231-275, 1994.
[6] Zwiers J., Compositionality, concurrency and partial correctness. *Lecture Notes in Computer Science* 321, 1987.