Connectedness and Synchronization

A. Mazurkiewicz*
A. Rabinovich†
B. A. Trakhtenbrot‡

1 Introduction

Given a description language for concurrent systems, the specification of its semantics amounts usually to assigning to each description an object from a hopefully well understood domain D whose elements are called semantical *processes.*
Here is a general accepted 'taxonomy' of event oriented processes:

1. *Linear process:* a set of runs, where a run is a sequence of events (actions) [Ho].

2. *Pomset process:* a set of node-labelled partial orders, henceforth refers to as pomsets [Pet], [Pr1], [Maz84].

3. *Automaton process* a rooted transition diagram whose edges are labelled by actions [Mil].

4. *Event structure* [NPW]; also *behavior structure* [RT0].

This list reflects the alternatives:

'Linear' time (1,2) vs branching time (3,4)
Interleaving (1,3) vs causality ('true concurrency') (2,4)

The four domains support different levels of abstraction in semantical specifications: linear processes are the coarsest, whereas event structures (behavior structures), which integrate branching and causality are the most discriminating.

On each of these levels *synchronization* is a basic operation of composing event oriented

*Institute of Computer Science, Polish Academy of Sciences PL-009-01 Warsaw PKiN
†Tel-Aviv University Tel-Aviv, and IBM Research Center, Yorktown Heights, NY 10598
‡Tel-Aviv University Tel-Aviv 69978 and Imperial College London

concurrent systems. In general, by synchronization we mean a composition of discrete concurrent systems which identifies some of their actions. Let us first recall some circumstances which are common to all these models.

1. *Action alphabet.* Each process Pr is equipped with a set $alph(Pr)$ - the alphabet of the actions which are available in this process. Note that the runs (or labels) of Pr use actions only from $alph(Pr)$, but not necessarily all of them.

2. *Synchronization.* This is a binary operation on processes; the alphabet of the synchronization is union of the alphabets of the operands. The operation is commutative and associative.

In this paper we supplement the taxonomy list above with another kind of semantical domains, namely we consider *connected relations.* In the most general setting connected relations are parametrized by a given domain D which obeys some specific finiteness conditions; we say that D is an F-domain [RT3]. In the particular case when D is the domain of natural numbers, connected relations appeared first in [Maz1] under the name of *multitrees*. For connected relations one can define a natural version of synchronization *(strong conjunction).* Multitrees may be associated with linear processes in such a way that strong conjunction of multitrees corresponds to synchronization of linear processes. When parametrized wrt an F-domain, connected relations are associated with objects called in [RT4] 'processes over F-domain'; these processes present a far reaching generalization of 'linear processes'. We came to the study of multitrees in [Maz2] and of connected relations over arbitrary F-domains in [RT3] through the investigation of different problems. Common for them was only the aspiration to achieve modular description and analysis of systems based on the net concept, namely - of Petri nets [Maz2] and of dataflow networks [RT4]. We strongly believe that the use of the compositional approach to these systems, and clarification of the fundamental limits of this approach, offer a deep insight into a variety of concurrent phenomena in discrete systems.

It is beyond the scope of this paper to deal with the original problems which suggested the conceptual background and techniques concerning connected relations and generalized processes. But let us explain briefly how multitree semantics arose in [Maz2] and how independently, in a completely different context connected relation semantics appeared in [RT4]. This coincidence confirmed our opinion that the notions under consideration are quite natural and of general interest.

1. The compositional approach to causal semantics of condition event Petri nets was first

formulated and investigated in [Maz1]. Later it was continued in [Maz2 RT0, W] for the more sophisticated Petri nets called P/T nets. Using pomsets seemed to be the proper way to follow, because pomsets exhibit explicitly the ordering intended to describe the causal dependency of transition occurrences. However, further investigation in [Maz2] convinced the author that sometimes the causal structure of P/T Petri nets is too complicated to be expressible by an ordering of transition occurrences. That is the reason why in [Maz2] it is argued that one should resign from representing processes in P/T nets by pomsets. As an alternative approach, multitrees and their synchronization were defined and advocated there.

2. Research on dataflow networks centers around fundamental problems raised in the pioneering works of Kahn [Ka] and Brock-Ackerman [BA]. The intention is to clarify the possibility of applying familiar functional and modular reasoning to input-output behavior of dataflow networks. Unlike for Petri nets it seems that causal aspects are not relevant for these problems, and that one can manage with interleaving (even with linear) processes [RT1]. Another difference from Petri nets is that in addition to synchronization one has also to analyze carefully the operations of *hiding* ($\exists$) and *union* ($\cup$) (choice). In [RT2, RT3] *nets of relations* were introduced as an appropriate tool to investigate the input-output behavior of dataflow networks over nondeterminate modules. That is how connected relations came to light. Under the influence of [Mi] the authors argued there that strong conjunction ($\underline{\&}$) is the appropriate notion of synchronization to be used for connected relations over stream domains. The next step was to realize that stream domains are not relevant and may be replaced by arbitrary F-domains.

In the sequel the material is organized as follows. We consider first the algebra of multitrees following closely the exposition of the subject in [Maz2] in what concerns the fundamental operations synchronization and strong conjunction. However, according to [RT4] we consider also (in addition to synchronization) hiding and union: this is not necessary for the semantical consideration of P/T - nets in [Maz89] but is relevant for semantics of dataflow networks. Note that the joint consideration of multitree synchronization with hiding and union, brings to light certain *anomalies* in the algebra of multitree synchronization, when compared with the ordinary algebra of process synchronization.

We continue then with the general theory for arbitrary F-domains Actually, what it remains to do is to apply more or less routine domain-theory to the ideas illustrated wrt multitrees. But note the following two circumstances:
a) About the definition of processes over arbitrary F-domains: This is a very general notion

of processes which goes beyond the ordinary event-oriented concepts. In [RT4] it is shown that processes over appropriate F-domains cover different formalisms for distributed systems. Hence, this notion provides a certain step toward the answer on the somehow vague question 'what is a model of concurrent computation?' [Ab]. Actually, processes over F-domains come very close to the general notion of 'model for asynchronous networks' as elaborated in [Ab].

b) The generalization to F-domains preserves the algebra of the operations $\underline{\&}$, $\exists$, $\cup$ as established for multitrees. But it implies even more: for every non trivial F-domain there hold also the anomalies we mentioned above for multitrees. Hence 'anomalies' are not the price one has to pay just for using multitrees.

2 String Languages and Multiset Languages

Let T^* denotes the set of all strings over alphabet T. As usual the concatenation of strings $s_1,\ s_2 \in T^*$ will be denoted by $s_1 s_2$. Any subset of T^* is called a language over alphabet T. For a language L its alphabet is denoted by $alph(L)$. Let $s_1,\ s_2$ be strings; we say that s_1 is a prefix of s_2 if there is a string s_3 such that $s_1 s_3 = s_2$. We consider the following operations on languages:

Synchronization ($\|$).

$$\begin{gathered} alph(L_1 \| L_2) = alph(L_1) \cup alph(L_2) \\ s \in (L_1 \| L_2) \text{ iff for } i = 1, 2 \\ s | alph(L_i) \in L_i \end{gathered}$$

where $s|A$ is the notation for the string one gets from s by deleting all event which are not in A.

Hiding. $\exists\ p.\ L$ results in the language over $alph(L) - p$; its strings are obtained from the strings of L through the deletion of all occurrences of p.

Union. For languages L_1, L_2 of the same alphabet, $L_1 \cup L_2$ inherits this alphabet and contains all strings in L_1 and in L_2.

We observe now that synchronization, union and hiding obey the same laws as conjunction, disjunction and existential quantifier in logic.

Claim 1.

1. $\|$ is commutative and associative.

2. Distributivity of $\|$ over $\cup$: $L_1 \| (L_2 \cup L_3) = (L_1 \| L_2) \cup (L_1 \| L_3)$,

3. Moving the quantifier: $\exists p.\ (L_1 \| L_2) = L_1 \| \exists p.\ L_2$, provided p is not in $alph(L_1)$.

Proofs (1) is well known, (2) trivially follows from the definition of synchronization for languages. Below we give a proof of (3).
a) Checking that $\exists p.\ (L_1 \| L_2) \subseteq L_1 \| \exists p.\ L_2$. If $s \in \exists p.\ L_1 \| L_2$ then by inserting in s some occurrences of p one obtains a string $s_1 \in L_1 \| L_2$. Let $s_2 \stackrel{def}{=} s_1 | alph(L_1)$ and $s_3 \stackrel{def}{=} s_1 | alph(L_2)$. Then $s_2 \in L_1$ and $s_3 \in L_2$. Since $p \notin alph(L_1)$ we have that $s_1 | alph(L_1) = s | alph(L_1)$; therefore $s | alph(L_1) \in L_1$. Also, since s_1 was obtained from s by inserting some occurrences of p, we have that $s | \{alph(L_2) - p\} = s_1 | \{alph(L_2) - p\}$. But $s_1 | \{alph(L_2) - p\} = s_3 | \{alph(L_2) - p\} \in \exists p.\ L_2$; Therefore $s | \{alph(L_2) - p\} \in \exists p.\ L_2$. To sum up we have $s | alph(L_1) \in L_1$ and $s | \{alph(L_2) - p\} \in \exists p.\ L_2$, therefore $s \in L_1 \parallel \exists p.\ L_2$.
b) Checking that $L_1 \parallel \exists p. L_2 \subseteq \exists p. L_1 \parallel L_2$. Take any $s \in L_1 \parallel \exists p. L_2$. Then $s | alph(L_1) \in L_1$ and $s_1 \stackrel{def}{=} s | \{alph(L_2) - p\} \in \exists p.\ L_2$. Therefore there is $s_2 \in L_2$ with $s_2 | \{alph(L_2) - p\} = s_1$. Let s_3 be obtained from s by inserting some occurrences of p in the same order as they appear in s_2, i.e. $s_3 | alph(L_2) = s_2 | alph(L_2)$. Then $s_3 | alph(L_2) \in L_2$ and $s_3 | alph(L_1) = s | alph(L_1) \in L_1$. Therefore $s_3 \in L_1 \| L_2$. But $s = s_3 | \{alph(L_1) \cup alph(L_2) - p\}$. Hence $s \in \exists p. L_1 \| L_2$.
□

Let T be an alphabet. By a multiset over T we shall mean any mapping $\tau : T \to \omega$. If τ is a multiset over $T = \{p_1,\ p_2,\ \ldots\ p_k\}$ then $\tau(p_i)$ is called the multiplicity of p_i in τ. Often we use notation $n_1 p_1 + n_2 p_2 + \ldots + n_k p_k$ for a multiset τ in which p_i has multiplicity n_i; we also drop members with multiplicity 0 from this expression. A multiset over T which assigns 0 to every $p \in T$ will be denoted by 0_T, or just by 0 when T is clear from the context. For any two multisets $\tau,\ \tau'$ we say that τ' includes τ and write $\tau \leq \tau'$ if $\tau(p) \leq \tau'(p)$ for each $p \in T$. Sets of multisets over T will be called multiset languages over T.

Example Let $\{a,\ b,\ c\}$ be an alphabet; then $a + 2b$ is a multiset with multiplicity of $a = 1$, of $b = 2$, and of $c = 0$. The multiset $a + 2b + c$ includes the multiset $a + 2b$.

The following operations are defined on multiset languages:
Conjunction. $L_1 \& L_2$

$$alph(L_1 \& L_2) = alph(L_1) \cup alph(L_2)$$
$$\tau \in (L_1 \& L_2) \text{ iff for } i = 1, 2$$
$$\tau | alph(L_i) \in L_i$$

where $\tau | A$ is the the multiset over A defined as $(\tau | A)(p) \stackrel{def}{=} \tau(p)$ for $p \in A$.
Hiding. $\exists\ p.\ L$ results in the language over $alph(L) - p$; $\tau \in \exists\ p.\ L$ if there is τ' in L such

that $\tau(q) = \tau'(q)$ for $q \in alph(L) - p$.

Union. For languages L_1, L_2 over the same alphabet, their union $L_1 \cup L_2$ inherits this alphabet and contains all multisets in L_1 and in L_2.

For multiset languages claim 1 holds with $\|$ replaced by &.

Claim $1'$.

1. & is commutative and associative.
2. Distributivity of & over $\cup$: $L_1 \& (L_2 \cup L_3) = (L_1 \& L_2) \cup (L_1 \& L_3)$,
3. Moving the quantifier: $\exists p.\ (L_1 \& L_2) = L_1 \& \exists p.\ L_2$, provided p is not in $alph(L_1)$.

Comment. Again, these are the same laws as for conjunction, disjunction and existential quantifier in logic.

3 Processes and Multitrees

A linear *process* is a prefix closed string language. Synchronization, hiding and union on processes are defined in the same way as for the corresponding languages. It is easy to see that the set of linear processes is closed under these operations.

Let beh be a mapping which to each string s over T assigns a multiset $beh(s)$ over T; namely for each $p \in T$ $beh(s)(p)$ is the number of occurrences of p in s. Extend beh to languages: $beh(L) \stackrel{def}{=} \{beh(s) \ : \ s \in L\}$, for any language L. We say that a multiset language L' is the behavior of a process L, or L implements L', if L' is the image of L under beh. It is easy to see that not each multiset language is implemented by a process. A necessary and sufficient condition for implementability is *connectedness.* Let L be a multiset language over T. We say that L is connected, if for each $\tau \in L$, which is different from 0, there are $p \in T$ and $\tau_1 \in L$ such that, $\tau(p) = \tau_1(p) + 1$ and $\tau(q) = \tau_1(q)$ for all $q \neq p$. In this case we say that τ_1 immediately precedes τ (notation $\tau_1 \ll \tau$). The definition implies that $\{0\}$ over any alphabet is connected and for every τ in a connected language L there is a finite sequence $0 = \tau_1 \ll \tau_2 \ll ... \ll \tau_n = \tau$ with each τ_i in L.

Connected multiset languages are called multitrees and will serve us as another model of semantical domains. It is easy to see that multitrees are closed under union and hiding but are not closed under conjunction.

Example. The following sets are multitrees:

$$\{0,\ a,\ b,\ a+b,\ a+c\} \text{ over } \{a,\ b,\ c\}$$
$$\{0,\ a,\ b,\ a+b,\ a+c,\ b+c,\ a+b+c\} \text{ over } \{a,\ b,\ c\}$$
$$\{\ \tau\ :\ \tau(b) \leq \tau(a)+1\} \text{ over } \{a,\ b\}$$

The set $\{\ \tau\ :\ \tau(b) = \tau(a)\}$ over $\{a,\ b\}$ is not a multitree (not connected).

For any multiset language L the maximal connected subset of L will be called the kernel of L and denoted by $Kern(L)$.
By *strong conjunction* of two multitrees $L_1,\ L_2$ we shall mean the multitree $L_1\underline{\&}L_2 \stackrel{def}{=} Kern(L_1\&L_2)$.

Claim 2.

1. $\underline{\&}$ is commutative and associative.
2. $L_1\underline{\&}L_2...\underline{\&}L_k = Kern(L_1\&L_2...\&L_k)$
3. Distribution anomaly: $(L_1\underline{\&}L_3) \cup (L_2\underline{\&}L_3) \subseteq (L_1 \cup L_2)\underline{\&}L_3$, but inclusion cannot be replaced by equality.
4. Hiding anomaly: $\exists p.\ (L_1\underline{\&}L_2) \subseteq L_1\underline{\&}\exists p.\ L_2$, provided that p is not in the alphabet of L_1, but inclusion cannot be replaced by equality.

Proof. (1) Commutativity of $\underline{\&}$ follows from commutativity of $\&$. Associativity of $\underline{\&}$ follows from associativity of $\&$ and from the following:
Fact. For any multiset languages $L_1,\ L_2$ $Kern(L_1\&L_2) = Kern(L_1\&Kern(L_2))$ and
(2) also follows from the associativity of $\&$ and Fact above.
The inclusion in (3) follows from the monotonicity of $\underline{\&}$ wrt inclusion of multiset languages.
The inclusion in (4) follows from claim $1'(3)$ and from the fact that $\exists p.\ Kern(L) \subseteq Kern(\exists p.\ L)$.
Below we show by appropriate examples that 'anomalies' hold, that unlike the laws for logical conjunction inclusion cannot be replaced by equality in (3) and (4).

Example of distribution anomaly. Consider the following languages $L_1,\ L_2,\ L_3$ over $\{a,\ b\}$. $L_1 = \{0,\ a,\ 2a,\ 2a+b\}$ $L_2 = \{0,\ b,\ a+b\}$ $L_3 = \{0,\ b,\ b+a,\ 2a+b\}$.

$L_1\&L_3 = \{0,\ 2a+b\}$, therefore $L_1\underline{\&}L_3 = \{0\}$. $L_2\&L_3 = \{0,\ b,\ a+b\}$, therefore $L_2\underline{\&}L_3 = \{0,\ b, a+b\}$. $L_1 \cup L_2 = \{0,\ a,\ b,\ a+b,\ 2a,\ 2a+b\}$. Hence, $(L_1 \cup L_2)\&L_3 = \{0,\ b,\ a+b,\ 2a+b\}$ and this is connected multiset language. Therefore $(L_1 \cup L_2)\underline{\&}L_3 = \{0,\ b,\ a+b,\ 2a+b\}$. From this we see that $(L_1 \cup L_2)\underline{\&}L_3 \neq (L_1\underline{\&}L_3) \cup (L_2\underline{\&}L_3)$.

Example of hiding anomaly. Consider languages L_1 and L_2 over alphabets $\{a,\ b\}$ and $\{a,\ b,\ c\}$ respectively. $L_1 = \{0,\ b,\ b{+}a,\ b{+}2a\}$, $L_2 = \{0,\ b,\ b{+}a,\ c,\ c{+}a,\ c{+}2a,\ c{+}2a{+}b\}$. Let $L_3 \stackrel{def}{=} \exists c.\ L_2$. Hence, $L_3 = \{0,\ b,\ b+a,\ a,\ 2a,\ 2a+b\}$. The multiset language $L_1 \& L_3$ is equal to $\{0,\ b,\ b+a,\ b+2a\}$ and is easily seen to be connected. Therefore, $L_1 \& L_3 = L_1 \underline{\&} L_3 = \{0,\ b,\ b+a,\ b+2a\}$
On the other hand $L_1 \& L_2 = \{0,\ b,\ b+a,\ c,\ c+2a+b\}$ is not connected and $L_1 \underline{\&} L_2 = \{0,\ b,\ b+a,\ c\}$; Therefore $\exists c.\ L_1 \underline{\&} L_2 = \{0,\ b,\ b+a\}$. Hence,

$$\exists c.\ L_1 \underline{\&} L_2 = \{0,\ b,\ b+a\} \neq \{0,\ b,\ b+a,\ b+2a\} = L_1 \underline{\&} \exists c.\ L_2$$

□

Let *fat* be the mapping which assigns to each multitree L over T the process $fat(L)$ over T defined as follows: $fat(L) = \{\ s\ :\ beh(t)$ is in L for each prefix t of $s\}$. Clearly $fat(L)$ is the maximal process which implements L. We say that a process L' is fat if $L' = fat(L)$ for some multitree L (it is clear that in this case L should be the behavior of L').

Claim 3. (Strong Conjunction vs Synchronization). For fat processes $L_1,\ L_2$ the process $L_1 \| L_2$ is fat and implements the relation $beh(L_1) \underline{\&} beh(L_2)$.

Proof. First we show that $beh(L_1 \| L_2) \subseteq\ beh(L_1) \underline{\&} beh(L_2)$. Let $s = a_1 a_2 ... a_n \in L_1 \| L_2$. (a) all prefixes of s are in $L_1 \| L_2$; (b) $beh(t) \in beh(L_1) \& beh(L_2)$ for every $t \in L_1 \| L_2$; (c) $0 \ll beh(a_1) \ll beh(a_1 a_2) \ll ... \ll beh(a_1 a_2 ... a_i) \ll ... \ll beh(a_1 a_2 ... a_n)$. By (a) and (b) for every i the multiset $beh(a_1 a_2 ... a_i)$ is in $beh(L_1) \& beh(L_2)$; therefore by (c) $beh(s) \in Kern(beh(L_1) \& beh(L_2)) = beh(L_1) \underline{\&} beh(L_2)$. We have shown that $L_1 \| L_2 \subseteq fat(beh(L_1) \underline{\&} beh(L_2))$. Now we shall show that $fat(beh(L_1) \underline{\&} beh(L_2)) \subseteq L_1 \| L_2$. Let $s = a_1 a_2 ... a_n \ \in fat(beh(L_1) \underline{\&} beh(L_2))$. Then for every i $beh(a_1 a_2 ... a_i)|alph(L_1) \in beh(L_1)$ and $beh(a_1 a_2 ... a_i)|alph(L_2) \in beh(L_2)$. And since $L_1,\ L_2$ are fat we have that $s|alph(L_1) \in L_1$ and $s|alph(L_2) \in L_2$ Therefore $s \in L_1 \| L_2$. □
As a straightforward consequence we get that $beh(L_1 \| L_2) = beh(L_1) \underline{\&} beh(L_2)$ for fat processes $L_1,\ L_2$. Note also the following facts:
(1) $beh(L_1 \cup L_2) = beh(L_1) \cup beh(L_2)$ and $beh(\exists p.\ L) = \exists p.\ beh(L)$ for arbitrary processes $L_1,\ L_2,\ L$ and $p \in alph(L)$.
(2) fat processes are not closed under union and hiding, but are closed under synchronization.

4 Port Relations

Let D be a set of data 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 $Ports(R)$.

Clearly a multiset language over T is a port relation of type T over the set ω of natural numbers. The operations on multiset languages generalize straightforwardly for arbitrary port relations:

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

Conjunction. (This is the 'join' operation from data base theory). Let R_1 be a relation of type P_1 and R_2 be a relation of type P_2. The **conjunction** of R_1 and R_2 is the relation of type $P_1 \cup P_2$ defined as follows:
$x \in R_1 \& R_2$ iff 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.

Clearly, & and $\cup$ are commutative and associative.

Hiding. $\exists p.\ R$ is a relation of type $Ports(R) - p$ which is the projection of R onto these ports.

It is easily seen that claim $1'$ holds for arbitrary port relations.

Claim $1''$.

1. & is commutative and associative. .
2. Distributivity of & over $\cup$: $R_1 \&(R_2 \cup R_3) = (R_1 \& R_2) \cup (R_1 \& R_3)$,
3. Moving the quantifier: $\exists p.\ (R_1 \& R_2) = R_1 \& \exists p.\ R_2$, provided p is not in $Ports(R_1)$.

Note that up to now no structure on D was assumed. To deal properly with connectedness we are going to consider partial ordered data domains.

5 F-domains

Let D be a set partially ordered by a relation $\leq$. A subset $A \subset D$ is *directed* if every finite subset of A has an upper bound in A. D is a (cpo) *completely partial ordered set* if every directed subset A has a least upper bound in D; in particular there is a least element in D denoted by $\perp$.

Examples.

1. $\overline{\omega}$. Note that ω with arithmetical order $\leq$ is not a cpo because there is no upper bound for the directed set of all numbers. By extending ω to $\overline{\omega} \stackrel{def}{=} \omega \cup \{\infty\}$ we get a cpo in which the least element is 0, and ∞ is the maximal element. But not every cpo has a maximal element.

2. $STREAM(\Delta)$. Let Δ be an arbitrary set. The stream domain $D = STREAM(\Delta)$ over a set 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'.

3. Flat domains. For any set Δ there is the flat cpo $\Delta_\perp$ which is the set $\Delta \cup \{\perp\}$ ordered by $x \leq y$ if $x = \perp$ or $x = y$.

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.

In $\overline{\omega}$ the natural numbers are just the finite elements; in $STREAM(\Delta)$ finite strings are finite elements; in flat domains all elements are finite.

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 $i < n$; we say that this strict chain leads to x_n; by definition $value(s)$ is x_n. We say that x is *finitely reachable* if there is a strict chain leading to x.

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

1. For each element z in D the set of finitely reachable elements bounded by z is directed and has z as its least upper bound.

2. If $c,\ a$ are finitely reachable and $c < a$ then there is a chain $s = \{x_i\ :\ i = 1...n\}$ such that $c = x_1,\ a = x_n$ and $x_i \ll x_{i+1}$ for all $i < n$.

Examples of F-domains.

1. $\overline{\omega}$.

2. Flat domains; for example BOOL=$\{\perp,\ true,\ false\}$

3. The domains $STREAM(\Delta)$ and

4. (Construction of F-domains). If D and E are F-domains so are $D \times E$, and D^{PORT} for any set of ports.

Note that in examples 1-3 the set of finite elements and the set of finitely reachable elements coincide. From now on when referring to domains we have in mind F-domains.

6 Connected Relations over F-domains

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

In particular, a relation R is called **connected** if $R = Kern(R)$. Obviously, $Kern(R)$ is the maximal connected subset of R.

Examples of relations. Let D be a F-domain and let $p,\ q$ be ports.

1. $p \leq q$ is the subset of finitely reachable elements of $D^{\{p,\ q\}}$ defined as

$$\eta \in p \leq q \text{ iff } \eta(p) \leq \eta(q)$$

   Clearly $p \leq q$ is a connected relation.

2. $p = q \quad =_{def} \{\eta\ :\ \eta(p) = \eta(q)$ is a finitely reachable element of $D\}$. This relation is not connected.

Operations on connected port relations. Operations on connected port relations straightforwardly extend the operations on multitrees. The results of hiding and union are also connected, but the conjunction of connected relations is not necessarily a connected relation (in the examples above, $p = q$ is the conjunction of $p \leq q$ and $q \leq p$ and it is not connected); therefore instead of conjunction we consider **strong conjunction** of connected relations (notation $\underline{\&}$)

$$R_1 \underline{\&} R_2 =_{def} Kern(R_1 \& R_2).$$

Comments. Clearly, $Kern(Kern(R)) = Kern(R)$. Hence $Kern$ is a projection of the CPO of all relations into the CPO of connected relations (both CPO are ordered by inclusion between relations). Recall that $R_1 \underline{\&} R_2 = Kern(R_1 \& R_2) = sup\{R \ : R$ is a connected subset of $R_1 \& R_2\}$. Therefore $\underline{\&}$ is the projection of $\&$. Moreover $\underline{\&}$ is the 'best' approximation to $\&$ in the domain of connected relations.

Claim $2'$.

1. $\underline{\&}$ is commutative and associative.
2. $R_1 \underline{\&} R_2 ... \underline{\&} R_k = Kern(R_1 \& R_2 ... \& R_k)$
3. Distribution anomaly: $(R_1 \underline{\&} R_2) \cup (R_1 \underline{\&} R_3) \subseteq R_1 \underline{\&} (R_2 \cup R_3)$, but inclusion cannot be replaced by equality.
4. Hiding anomaly: $\exists p.\ (R_1 \underline{\&} R_2) \subseteq R_1 \underline{\&} \exists p.\ R_2$, provided that p is not a port of R_1, but inclusion cannot be replaced by equality.

Comment. The proof of claim $2'$ is an adaptation of the proof of claim 2; in (3) and (4) inclusion cannot be replaced by equality in *every domain* which contains a chain of 3 *elements.*

7 Processes over F-domains

A **process** Pr of type P over D is a prefix closed set of strict chains in D^P; the type of Pr is denoted by $Ports(Pr)$.

For a strict chain s in D^P and for $A \subseteq P$ the *projection* $s|A$ is the strict chain obtained as follows: take the chain composed from projections of the elements of s and delete repetitions.

As usual, for processes one considers the fundamental operations:

Synchronization ($\|$).

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

Hiding $\exists\ p.\ Pr$ results in the process with ports $Ports(Pr) - p$; its strict chains are the projections of the strict chains in Pr.

Union. For processes Pr_1, Pr_2 of the same type, $Pr_1 \cup Pr_2$ inherits this type and contains all chains in Pr_1 and in Pr_2.

Claim $1'''$.

1. $\|$ is commutative and associative.
2. Distributivity of $\|$ over $\cup$: $Pr_1 \| (Pr_2 \cup Pr_3) \ = \ (Pr_1 \ \| \ Pr_2) \cup (Pr_1 \ \| \ Pr_3)$,
3. Moving the quantifier: $\exists p.\ (Pr_1 \| Pr_2) = Pr_1 \| \exists p.\ Pr_2$ provided p is not in $Ports(Pr_1)$.

The *value* of a process Pr is the port relation $value(Pr) = \{value(s)\ :\ s \in Pr\}$. If R is the value of Pr we say that Pr *implements* R.

Fact. A port relation is the value of a process iff it is a connected relation in the corresponding F-domain,

Definition. A process Pr is *fat* if it consists of all strict chains s which have the property: for every prefix s_1 of s it is the case that $value(s_1) \in value(Pr)$.
A process is *meager* if it is not fat. The function which maps a connected relation to its corresponding fat process will be denoted by *fat.*

Claim $3'$. (Strong Conjunction vs Synchronization). For fat processes Pr_1, Pr_2 the process $Pr_1 \| Pr_2$ is fat and implements the connected relation $value(Pr_1) \underline{\&} value(Pr_2)$.

Proof. Similary to the proof of calim 3.

8 Processes with Message Passing

We describe two different extensions of the notion 'linear process' which are used in models of dataflow [WA, PA]. We show that (up to isomorphism) both are instances of the general notion 'process over F-domain'. It follows in particular, that claims $1'''$, $3'$ hold in both cases.

8.1 Example 1. Stream Processing

Often one has to be more specific about the action alphabet of the linear processes. Here are the relevant stipulations:

Communications. The action alphabet is structured as a Cartesian product $Ports \times \Delta$. An action $< p,\ d >$ is said to be a communication through port p which passes the message (the datum value) d. In particular it may happen that Δ is unary (consists of one element); usually in this case the data value is omitted, and communications are identified with the ports. Actually, we get a linear process in the simplest original sense of section 3.

We describe message passing processes parametrized by a set P of ports and a set Δ of data.

(1) *(Message passing) run:* a string of communications.

(2) A *(message passing) process* Pr of the type P is a prefix closed set of runs.

As usual, one considers the fundamental operations. Their definitions up to minor details concerning the types of the processes remain the same as for linear processes.

Synchronization ($\|$).

$$\begin{gathered} Ports(Pr_1\|Pr_2) = Ports(Pr_1) \cup Ports(Pr_2) \\ \text{run } s \in (Pr_1\|Pr_2) \text{ iff for } i = 1,2 \\ s|Ports(Pr_i) \in Pr_i \end{gathered}$$

where $s|A$ is the notation for the string one gets from s by deleting all communications which are not on ports in A

Hiding. $\exists\ p.\ Pr$ results in the process with ports $Ports(Pr) - p$; its runs are obtained from the runs of Pr through the deletion of all communications on the port p.

Union. For processes Pr_1, Pr_2 of the same type $Pr_1 \cup Pr_2$ inherits this type and contains all runs in Pr_1 and in Pr_2.

To every run $s \in Pr$ there corresponds a function $beh(s)$ from the set P of ports defined as: $beh(s)(p) =$ the sequence (stream) of data communicated through p in s.

The relational behavior of a process Pr is the port relation $beh(Pr) = \{beh(s) \ : \ s \in Pr\}$.

Message passing processes as processes over F-domains of streams. Consider an arbitrary message passing run $s = c_1 c_2 ...$ of type P. Associate with s the sequence $\perp, \ beh(c_1), beh(c_1 c_2)....$ Clearly, this is a strict chain in the F-domain $STREAM(\Delta)^P$, and the value of this chain coincides with $beh(s)$. Moreover, in this way we get a 1-1 mapping ϕ of runs of type P onto strict chains in $STREAM(\Delta)^P$. It is easily seen that the algebra of synchronization, hiding and union wrt message passing processes is isomorphic to that of the abstract processes over the F-domains $STREAM(\Delta)$. Namely, isomorphism is provided just by the mapping ϕ which transforms a run s of the process into the corresponding strict chain.

8.2 Example 2. Processing Streams with 'Holes'

Sometimes in dataflow theory [PA, WA] are used also processes related to the domain of 'streams with holes' over a set Δ. This domain consists of all infinite sequences over $\Delta \cup \perp$ and is partially ordered by pointwise ordering. It is isomorphic to the domain of functions from natural numbers to $\Delta \cup \perp$ and also to the set of subsets $< n_i, d_i >$ of $N \times \Delta$, which do not contain pairs with the same first component. We use for this domain and any of its isomorphic representations the notation $HSTREAM(\Delta)$ (mnemonic for 'streams with holes').

Here are the definitions parametrized by a set P of ports and a set Δ of data.

(1) **Communication:** a pair $<$ *port*, *marked datum* $>$. A marked datum is itself a pair $<$ *natural number*, *datum* $>$

(2) **Run:** a consistent string of communications, such that for every p and n there is at most one occurrence of $< p, \ < n, \ d > >$ in the run.

(3) **Behavior.** To every run $s \in Pr$ there corresponds a function $beh(s)$ from the set P of ports defined as: $beh(s)(p) =$ the *set of marked data* $< n, \ d >$ *which are communicated through* p *in* s.

Up to these differences further definitions are the same as in example 1. It can be shown like in example 1 that such processes are isomorphic to abstract processes over $HSTREAM$ domain.

Despite the first impression that domains $STREAM$ and $HSTREAM$ are looking very similarly, they have essentially different structure. For example for any finite $a < b$ in $STREAM$ there is a unique chain $\{d_i \ : i = 1...n\}$ with $d_1 = a$, $d_n = b$ and $d_i \ll d_{i+1}$, $i = 1, ..., n-1$. But in $HSTREAM$ for $a < b$ there are many such chains.

References

[Ab] S. Abramsky, A generalized Kahn principle for abstract asynchronous networks, in Mathematical Foundations of Program Semantics, Springer Verlag, 1990 (to appear).

[BA] Brock, J.D and Ackerman, W.B, Scenarios: A Model of Non-Determinate Computation, In Formalization of Programming Concepts, LNCS 107, 1981, 252-259.

[Ho] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall International 1985.

[Ka] Kahn, G., The Semantics of a Simple Language for Parallel Programming, IFIP 74.

[Maz1] A. Mazurkiewicz, Semantics of Concurrent Systems: A modular fixed point trace approach, in Advanced in Petri nets 1984, LNCS 188.

[Maz2] A. Mazurkiewicz, Concurrency, Modularity and Synchronization, MFCS 1989, LNCS 379.

[Mil] R. Milner, A calculi for Synchrony and Asynchrony, TCS 25, 1986.

[Mi] Misra J,. Equational Reasoning About Nondeterministic Processes, in Proceedings of PODC 1989

[NPW] M. Nielsen, G. Plotkin, G. Winskel, Petri Nets, Event Structures, and Domains, Part 1, TCS 13, 1981.

[PA] Pingali, Arvind, Efficient Demand Driving Evaluation, ACM, TOPLAS vol 7(2), 1986

[Pet] C. A. Petri, Communication with Automata, TR-65-377, Applied Data Research, Princeton, 1966.

[Pr1] V. Pratt, Modelling Concurrency with Partial Orders, International Journal of Parallel Programming, Vol 15, 1986.

[RT0] Rabinovich, A., Trakhtenbrot, B. A., Behavior Structure and Nets, in Fundamenta Informatica 11(4), 1988.

[RT1] Rabinovich, A., Trakhtenbrot, B. A., 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 1988, LNCS vol. 354

[RT2] Rabinovich, A., Trakhtenbrot, B. A., Nets and Data Flow Interpreters, in the Proceedings of the Fourth Symposium on Logic in Computer Science 1989.

[RT3] Rabinovich, A., Trakhtenbrot, B. A., Communication among Relations (Preliminary Draft), Technical Report 151/89 Tel Aviv Un.

[RT4] Rabinovich, A., Trakhtenbrot, B. A., Communication among Relations, to appear in ICALP 90, LNCS vol 443.

[Tr] B. A. Trakhtenbrot, Understanding Nets, in MFCS 1989, LNCS 379.

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

[W] Winkowski, J., Event Structure Representation of the Behavior of Place/Transition Systems, Fundamenta Informatica 11(4), 1988.