The Polytechnic of Wales
Politechnig Cymru

Computer Studies Technical Report

CS-91-9

Analysing Concurrent Systems Through Lattices

by

Alexandre V Yakovlev

Department of Computer Studies
The Polytechnic of Wales
Pontypridd
Mid Glamorgan, CF37 1DL

Tel. No. 0443 480480

Contents

Abstract

The problems of semantic relationship between *causality* and *interleaving* and analysis of *functional determinism* and *data dependency* are tackled in terms of a *lattice-theoretical approach.* We introduce a basic form of poset, *the poset of cumulative states* of concurrent system behaviour (these states can be represented as *multisets of process actions* attributed to the events thus far occurred in the system), which is called *a cumulative diagram.* It helps to characterise particular subclasses of Petri net based processes, *semi-modular* and *distributive* ones, by means of the major background theorems stating that
the cumulative diagram of a persistent (safe and persistent, marked graph) Petri net is a semi-modular (distributive) lattice with a zero element.

The study then claims the following:
(i) the *partial order semantics* is descriptively equivalent to the *interleaving semantics* only for distributive processes;
(ii) an *operationally semi-modular process* (*confluent*, in the Keller-Milner's sense), which may however be non-distributive, can have some form of *functional non-determinism* (and, hence, lose its confluence) - this results in:
(ii.1) the impossibility to derive uniquely a partial order description from a given set of *execution sequences*, and
(ii.2) the distinction of possible execution sequences outcomes because of the *merging* of differently valued token flows.
With respect to the (ii.2) issue we also claim that the *purely control flow semantics* of a semi-modular but non-distributive process is deterministic while its *data flow semantics* may be non-deterministic, and both the control and data flow semantics of a distributive process are deterministic.

The properties herein discussed, using lattice-theoretical tools, nicely fit into the previous bed of results on *stability* (by G.Winskel), *confluence* (by R. Keller and R.Milner), *{AND+OR}causality* (by J.Gunawardena), *conservatism and data-independence* (by M.Rem).

We demonstrate some interesting *practical implications* of the lattice characterisation of concurrency in *digital circuit and program design* domains.

Topics covered: *causality/partial order theory of concurrency, analysis and behaviour of nets,high-level net models, application of nets to hardware structures*

Introduction

Concurrent systems, as opposed to sequential ones, suggest a number of ways to be semantically characterised. This basically stems from the problem of finding an adequate representation of concurrency paradigm in any of the formal frameworks employed,among which the most popular are state-transition systems [1], CSP [2], trace sets [3], nets [4], algebra of interacting agents [5]. Before using any of these or some other notations one has to clearly realise which form of *concurrency semantics* is needed for him/her and which is implied by the model. For the last ten years the subject of relationship between the various concurrency semantics has acquired tremendous interest among not only theoreticians but also in the practical design community [6,7]. For example, in our everyday practice of asynchronous digital hardware design we have been witnessing how important it is to match the meaning of execution of the Petri nets used for specifying self-timed systems with that of logical circuit dynamics [8].

The approach to represent a concurrent system's *behavioral semantics* through the set of *execution sequences*, or *traces*, was obviously inspired by the relationship between state-transition systems and the languages represented by them, which appeared to be quite natural for sequential systems. Further developments of this view, with concurrency flavour, have given rise to such semantics as step sequences ("execute as possible but not necessarily with maximal concurrency") , maximally concurrent semantics [9], and partial order semantics [10].

Recently, a series of contributions to the EATCS Bulletin tackled the controversy between *causal (partial order) semantics* and *interleaving semantics* [11-13]. The main result of these discussions could be summarised as follows. Although interleaving semantics can be suitable in certain, especially simulation-oriented, applications, the causal view upon concurrency is generally more versatile, especially when one has to deal with refinable specifications [11], or when specific properties like confusion, choice-absence, strong concurrency and "causal-next" relation, inherent in causal semantics [13], should be treated. Another important practical issue in favour of partial-order semantics is that it can be helpful in avoiding state-explosion problem, by explicit separation of concurrency from *non-determinism*, thus making tractable reasoning about concurrent systems [14].

In [15] we have outlined, by means of examples observation, the idea of a somewhat alternative treatment of the above controversy, viz. using the concept of *partially ordered sets (posets)* and *lattices* on the *cumulative states* of concurrent systems. Such states represent a history of the system's behaviour through numerical records of *action occurrences* with respect to certain initial state of the system. In other words, the poset and lattice characterisation can therefore be made also for the set of traces themselves (as opposed, and in addition, to the partial orders and lattices which are usually defined on the set of processes, trace sets or other forms of computation behaviour description [2,3]): by collapsing the traces into their *equivalence classes*, the so-called *multisets* of process actions, which are just another way to represent cumulative states. The first attempt to highlight the link between dynamic behaviour of Petri nets and lattices was made in [16] by Landweber and Robertson, who used Keller's fundamental arguments about the *persistence* property in parallel computations to prove that the Parikh space (which is similar to the cumulative state set) for a *persistent* Petri net forms a lattice under the natural ordering of integer vectors.

In this paper, a more in-depth discussion of the problem is presented. First, we introduce a lattice-theoretical framework of concurrent system behaviour, which is then used for laying a distinction between *distributive* and *semi-modular* processes. For the sake of clarity, we restrict ourselves with the processes that are *free from choices* in their execution. Rather, we allow only those alternative execution sequences, in terms of interleaving semantics, which are due to concurrency and its "non-deterministic" simulation, but not pertaining to condition-branches or conflict resolution. So far, this gives us necessary separation of concerns, whereof the central is the pure concurrency concern.

One of our main results is that partial order semantics is equivalent, by its descriptive power, to interleaving semantics only for distributive processes. Paving our way to this statement, we prove important theorems stating that partial order specifications given in terms of Petri net subclasses produce the cumulative state (or, equivalently, action multiset) behaviour which forms a *semi-modular lattice* if the net is persistent, and a *distributive lattice* if the net is safe and persistent or a *marked graph net.*

Another important conclusion is that an operationally semi-modular process, which may however be non-distributive, can have some form of functional non-determinism which results in: (i) the impossibility to derive uniquely a partial order description from a given set of execution sequences, and (ii) the distinction of possible execution outcomes because of the *merging* of *differently valued* token flows.

The (ii) issue is discussed through an *extended interpretation* of a Petri net, by means of a high-level net [17], in which tokens may have certain data values, or colours, and transition firings may produce distinguished resultant markings with respect to the values of tokens in input places. Analysis of this *data-valued token interpretation* allows us to establish a relationship between the *control flow semantics*, defined as the behaviour under the assumption that all tokens are indistinguishable, and *data flow semantics*, defined under the assumption that tokens may be associated with data values. We claim that the purely control flow semantics of a semi-modular but non-distributive process is deterministic, while its data flow semantics may be non-deterministic, and that both these semantics of a distributive process are deterministic. The latter result corresponds to the concept of *data-independent* and *conservative* processes presented in [18]. Thus, a data interpreted process is data-independent iff it is distributive.

Note that the results obtained with the lattice-theoretical approach are quite coherent to the ideas in [19], where *stable event structures* and safe Petri nets are characterised within the framework of *algebraic domains*. Here, the same class of processes is called distributive, with the restriction of considering only the processes free from choices. Our analysis, however, lifts to another direction, viz. to the case of persistent (and, generally, not necessarily safe) nets, where the partial orders generated form semi-modular lattices (which are, in other words [19], a proper subclass of *consistently complete and algebraic domains* - of course, up to isomorphism), and moreover, for the case of marked graphs, even distributive lattices (corresponding to *finitary and prime algebraic domains* [19]). This suggests [20] some way for *unification* by stating that a persistent and safe net generates the same, finitary and prime algebraic, domain up to isomorphism. This latter result, although being announced here because of its obvious theoretical importance, would, of course, demand more space for discussion than the limits of the present paper may allow. It is therefore postponed to a subsequent paper.

Recently, the investigation of similar properties has been done using *causal automata* [21], which, in an explicit form of event causality, exhibit both distributive (AND-causality) and semi-modular ({AND,OR}-causality, which is identical to confluence in the sense of Milner's [22] notion) paradigms.

Our analysis, therefore, brings us to a very important point where it seems quite necessary to further examine the semantics of Petri net *non-safeness* because of its obvious *twofold nature*: on the one hand, it represents the purely event-causal (OR-causality) paradigm, or the so-called "input conflict", which focuses on the possibility of alternative, and not necessarily mutually exclusive, causes for the same event (and this has been the main glory for both [19] and [21]); on the other hand, it may represent purely flow meaning, where the multiple tokens in the net places are just the result of larger "token traffic" through the actions, without any causality complications. The importance of the above distinction becomes even higher when we need to study the semantic picture of high-level nets where even the flow semantics of multiple tokens, as we attempt to show, brings some form of input conflict manifesting itself in data-dependency.

As a point to justify our approach it is worth quoting from Winskel's introduction to [19]: "There remains the curious mismatch ... : a computation which is described by an event structure, or Petri net, gives rise to a whole domain whereas usually in denotational semantics a computation denotes a single element of a domain". It remains then to hope that this work would also contribute to solving the above mismatch.

For the reasons of *pragmatism*, which appear to be quite demanding because, as [21] states, "most authors ... have found instability to be either unnecessary or undesirable", we demonstrate some interesting practical implications of the lattice characterisation of concurrency in *circuit* and *program design domains*, where, sometimes, the question of "necessity" or "desirability" cannot simply arise because of the "inevitability" of OR-causality (and hence, instability). We put more emphasis on the hardware examples where the *"semi-modular but non-distributive" paradigm of merge* has important, and perhaps novel to the reader, consequences in functionality. Here we show how merge is implemented in a *wired-OR logic*, in a *fault-tolerant structure with module*

redundancy and in a *task scheduling system*, where it is either an initial prerequisite of structural requirements ("inevitability" case) or can speed up the performance ("both "necessity" and "desirability" case) under certain assumptions of relative delay values. The final topic of the implications section is analysis of a dichotomy, recently emerged in the area of asynchronous circuits [23], between *speed-independence* and *delay-insensitivity*, where the latter is a stronger form of insensitivity of a circuit's correct behaviour to the delays in components (the delays are both in gates and wires) than the former (the delays in wires are negligible). We prove that the cumulative diagram describing the behaviour of a *delay-insensitive circuit* is distributive. This also means that any *speed-independent circuit* whose behaviour is semi-modular and non-distributive cannot be delay-insensitive.

Background

The discussion presented in this paper is based on some subclasses of Petri nets. The reader may find some of our notions as being restrictive. Basically, we need only the formalism of ordinary Petri nets, which is defined below, but in treating non-distributive processes with valued tokens, in Section 4, we shall make a special resort to token-valued nets.

Also, for supporting analysis of our models by means of lattices, a short list of definitions and an outline of appropriate proof technique is presented in Appendix. For more serious introduction on lattices we recommend [24].

Definition 2.1 *A quadruple $PN = (S,T,F,M_o)$ is said to be a marked Petri net (shortly, a net) iff: S and T are finite non-empty sets; $S \cap T = \emptyset$; $F \subseteq (S \times T) \cup (T \times S)$; $M_o: S \to N$; where N denotes the set of non-negative integers. Every mapping $M: S \to N$ is called a marking of PN, and for every $x \in S \cup T$ we denote:*

$$In(x) = \{y: (y,x) \in F\} \text{ and } Out(x) = \{y: (x,y) \in F\}$$

The elements of S and T are called *places* and *transitions*, respectively. Graphically they are represented as *circles* and *boxes* (*bars*). F is the *flow relation* of PN. It is represented by arcs joining boxes and circles, thereby defining the *incidence function* between transitions and places. M_o is called the *initial marking* of PN. Any marking M is visualised by the appropriate number of dots, called *tokens*, inside circles, i.e. $M(s)$ tokens inside the circle representing a place s.

A Petri net PN can generate a *dynamic behaviour*. This behaviour, or execution, is characterised by the corresponding semantics of the *sequences of transition firings*. We introduce it as follows.

Definition 2.2 *A transition $t \in T$ in a net $PN = (S,T,F,M_o)$ is said to be enabled under the current marking M iff for all $s \in In(t)$ $M(s) \geq 1$.*

Definition 2.3 *A transition $t \in T$ in a net $PN = (S,T,F,M_o)$ can fire iff it is enabled. The firing of t results in the new marking M' such that for every $s \in S$:*

$$M'(s) = \begin{cases} M(s) - 1 & \text{iff } s \in In(t) - Out(t) \\ M(s) + 1 & \text{iff } s \in Out(t) - In(t) \\ M(s) & \text{otherwise} \end{cases}$$

Graphically, the firing of a transition is represented as an instantaneous action removing exactly one token from each of the input places of the transition and adding exactly one token to each of its output places.

The above two definitions are fundamental in defining the so-called *execution sequences* model, which was the most commonly used kind of Petri net semantics [25]. It was shown that such a view does not distinguish between arbitrary interleaving (when two transitions can fire sequentially but in either order) and concurrency (when two transitions are enabled under the same marking and

can fire independently). For the purposes of this study, we do not need to abandon this semantics and refer to some other, more recent ones, such as, for example, the step sequence semantics, in which subsets of transitions firing simultaneously are defined.

Definition 2.4 *Two markings M and M' of a net PN = (S,T,F,M_o) are said to be in direct sequence relation with respect to a transition $t \in T$ iff t fires under M and such firing results in M'. We denote this as $M [t > M'$. Correspondingly, M and M' are said to be in sequence relation with respect some sequence of transitions $\sigma = t_1 \ldots t_n$ where $t_1, \ldots, t_n \in T$ iff there exist markings $M_1,\ldots,M_n$ satisfying: $M[t_1>M_1 [t_2> M_2 \ldots [t_n>M_n = M'$. We denote this as $M [\sigma> M'$. We also use notation $M [> M'$ if we are not particularly interested in mentioning a firing sequence.*

If two markings *M* and *M'* are in the (direct) sequence relation, this fact also means that *M'* is (*directly*) *reachable* from *M*.

For two firing sequences σ and τ, $\tau = t_1 \ldots t_n$, we define $(\sigma \div \tau)$, called *the excess of σ over τ*, as follows: Let σ_0 be σ. Obtain σ_{i+1} by deleting the leftmost occurrence of t_i from σ_i, if t_i occurs in σ_i. If not, then $\sigma_{i+1}=\sigma_i$. Define $(\sigma \div \tau)$ to be σ_{n+1}.

The dynamics of net behaviour can be conveniently depicted by the so-called *reachability graph* in which vertices stand for the markings reachable from the initial marking M_o.

Definition 2.5 *The triple $MD = (M_o, ->, R)$ is said to be the marking diagram of a net PN $=(S,T,F,M_o)$ where M_o is the initial marking, -> is the direct sequence relation, and R is the set of markings reachable from M_o.*

The net's marking diagram is thus represented by the reachability graph. An arc of such a diagram is associated with the transition which fires under the marking standing for the beginning vertex of the arc and results in the marking associated with its ending vertex. Thus, the marking diagram allows one to build, for a given firing sequence of transitions $\sigma = t_1,\ldots,t_n$ starting under the marking M_o, the unique corresponding sequence of markings $M_o,\ldots,M_n$, and vice versa.

It is easily seen that from the marking diagram of a net one can derive all the firing sequence semantics. A number of characteristic properties of the net can also be established from it, e.g. reachability, liveness, boundedness etc. In this discussion, our main interest turns to algebraic properties of the net behaviour that have no direct link with these, operational, issues.

Let σ be a sequence of transitions of $PN = (S,T,F,M_o)$. Let $\#\sigma(t)$, $t \in T$, denote the *number of times t occurs in σ*. The following statement is true due to the firing semantics definition.

Statement 2.1 *For a $PN = (S,T,F,M_o)$ and a firing sequence σ starting under marking M, the resulting marking M' is such that for every $s \in S$:*

$$M'(s) = M(s) + \sum_{t \in In(s)} \#\sigma(t) - \sum_{t \in Out(s)} \#\sigma(t)$$

Definition 2.6 *Let $PN = (S,T,F,M_o)$ be a net. A vector **a** having dimension $n=|T|$ is said to be the transition firing vector iff for some firing sequence $\sigma = t_{k1},\ldots,t_{km}$ starting from initial marking M_o: $a_i = \#\sigma(t_i)$ where $i = 1,\ldots,n$ and $t_i \in T$.*
*a_i is said to be the i-th component of **a**.*

It is seen from this definition that each sequence of transition firings, with respect to initial marking M_o, stands for the unique value of the transition firing vector, and hence, again, with respect to M_o, we can build the set of such values which can be *mapped* onto the set of reachable markings using the equation from Statement 2.1. Such mapping implies that the all zero value, $a0 = (0,...,0)$ stands for M_o, and for some t_i and M such that $M_o[t_i > M$, we build combination $a1$ which differs from $a0$ only by the presence of one in the i-th position, thus meaning that transition t_i has fired for the first time. When moving along any firing sequence in the marking diagram from M_o it is quite straightforward to map (one-to-many) on vector values $a0, a1, \ldots$.

Consider a set A of values of the transition firing vector. On this set, we define *a partial order relation such that for* $\mathbf{a}, \mathbf{b} \in A$ $\mathbf{a} \leq \mathbf{b}$ *iff there exist two firing sequences* α *and* β *starting from the initial marking* M_0, *whose firing vectors are* $\mathbf{a}$ *and* $\mathbf{b}$, *respectively, and* α *is prefix of* β. Furthermore, the *least upper bound* of $\mathbf{a}$ and $\mathbf{b}$, if exists, is denoted $LUB(\mathbf{a}, \mathbf{b})$, and the *greatest lower bound* as $GLB(\mathbf{a}, \mathbf{b})$.

Note 2.1 Such an ordering was proved to be a partial order by Gunawardena [20], who called it *prefix order up to permutations* to reflect the characteristic property of Petri nets, sometimes called "*the principle of irrelevance of history*" or *trace permutability*.

Definition 2.7 *A pair C(PN) = (C, ≤), which is a partially ordered set (poset), is said to be the cumulative diagram of net PN =(S,T,F,M_o) iff C is the set of values of the transition firing vector constructed for all firing sequences of PN with respect to M_o.The elements of C are called cumulative states.*

The graphical representation of *C(PN)* would be a *Hasse diagram* with the *zero element* standing for the initial marking.

The presence of the zero element in *C(PN)* guarantees that for any $a,b \in C$ there exists a lower bound. However, due to potential non-determinism in the net's behaviour we cannot state that *C(PN)* is generally a lattice. The following sections restrict our consideration to such subclasses of Petri nets as marked graphs and persistent nets, for which *C(PN)* forms a lattice with respect to the above ordering relation.

Note 2.2 The reader may however try as an exercise to disprove that *C(PN)* of an arbitrary live net is a lattice (or even a lower sublattice), by finding an example of such a net for which, for some two elements a and b in C, there is no $GLB(a,b)$ in C. We call a net *live* iff each of its transitions is live. According to [25], a transition t_i is said to be *live* iff for any M, which is reachable from M_o, there exists a firing sequence such that t_i is enabled under M' and $M[\sigma > M'$.

Note 2.3 It is also possible to build the same partial order on the set of equivalence classes of the net's firing sequences, which can be regarded as multisets of transitions, or "traces" using the terminology of [3].

Semi-modular and Distributive Processes

In this section we restrict ourselves with the subclass of behaviours which have no conditional branches or conflicts between transitions. Using such a restriction, we establish a class of lattices that are generated by purely concurrent processes. In terms of marked Petri nets these processes

can be defined, in their finite representation, as marked graphs.

Definition 3.1 *A net PN = (S,T,F,M_o) is said to be a marked graph iff for each place $s_i \in S$: $|In(s_i)| \leq 1$ and $|Out(s_i)| \leq 1$.*

Since a marked graph assumes no more than one input and one output transition for each place, we do no longer need a bipartite representation of such a net. Without any special definition we shall allow a marked graph, wherever it gives notational convenience, to be a directed graph whose vertices are semantically identical to the transitions of the corresponding Petri net and the arcs stand for the places of the net. Hence, the marking function is defined on the set of arcs. Again, graphically, it is designated by placing the appropriate number of tokens onto the arcs.

Before we present a lattice-theoretical characterisation of the processes described by marked graphs we should introduce a wider subclass of Petri nets, which are also conflict-free with respect to transition firing.

Definition 3.2 *A net PN = (S,T,F,M_o) is called persistent iff for each marking M reachable from M_o any transition $t_i \in T$ enabled under M either remains enabled in the markings directly reachable from M or fires thus leading to a marking M' such that $M\,[t_i> M'$, i.e. if $M[t_i>$ and $M[t_1,...,t_k>M_1$, where $t_i \notin \{t_1,...,t_k\}$, then $M_1[t_i>$.*

In other words, in a persistent net a once enabled transition cannot be disabled by the firing of another transition. Persistent nets were studied in [16], but, by and large, with respect to the reachability and boundedness problems. There is, however, an important theorem (Theorem 3.1) in [16] stating that the Parikh space (the set of cumulative states, in our terms) built for a persistent net is a lattice under the natural ordering of vectors ($x \leq y$ iff $x_i \leq y_i$ for all i, $1 \leq i \leq n$). The proof of this theorem is based on the following important lemma.

Lemma 3.1[16] *Let α_1 and α_2 be two firing sequences starting under the initial marking M_o of a persistent net. Then there is a firing sequence β, starting at M_o, such that*

$$b = max\,(a_1,a_2),$$

where a_1,a_2 and b the firing vectors (Parikh map points in [16]) *corresponding to α_1 ,α_2 and β, respectively, and max denotes the componentwise maximum operation. Moreover, β may be constructed so that $\beta=\alpha_1 \cdot (\alpha_2 \dot{-} \alpha_1)$, where $\cdot$ stands for usual concatenation of two sequences, and $\dot{-}$ denotes the excess* (as defined in Section 2).

This lemma also helps in proving the following.

Lemma 3.2 *For a persistent net, if $a_1 <= a_2$, where a_1 and a_2 are two cumlative states in $(C, \leq)$ and <= stands for the natural ordering of vectors, <= is identical to the partial order $\leq$*

Proof

It is sufficient to show just that $a_1 <= a_2$ implies $M_1[>M_2$, where M_1 and M_2 are reached from the initial marking by the firing sequences corresponding to a_1 and a_2 in accordance with Definition 2.6 and Statement 2.1. Let α_1 and α_2 stand for such sequences, respectively. According to Lemma 3.1, we shall have, following the same notation, that $b = max\,(a_1,a_2)=a_2$. Hence, due to Statement 2.1, β leads to the same marking, M_2, as α_2 . Furthermore, since $\beta=\alpha_1 \cdot (\alpha_2 \dot{-} \alpha_1)$, which means that β passes through M_1, after its partial execution α_1, it is clear that M_2 is reachable from M_1 by the firing sequence $(\alpha_2 \dot{-} \alpha_1)$.**Q.e.d.**

This lemma justifies, in our further discussion that takes place within the class of persistent nets (and marked graphs), the usage of the natural ordering of vectors as the partial order $\leq$ relation under the conditions of Definition 2.7.

The following theorems are most crucial for the taxonomy of processes in terms of lattices. Our Theorem 3.1 strengthens the corresponding result of Theorem 3.1 in [16].

Theorem 3.1 *The cumulative diagram C(PN) of a persistent PN is semi-modular lattice with a zero element under the ordering of Definition 2.7(the natural ordering of vectors).*

For notational convenience, define a *disjunction* $\boldsymbol{a} \vee \boldsymbol{b}$, as a componentwise maximum, i.e. $(\boldsymbol{a} \vee \boldsymbol{b})_i = max\ (a_i,b_i)$,and a *conjunction* $\boldsymbol{a} \wedge \boldsymbol{b}$, as componentwise minimum, i.e. $(\boldsymbol{a} \wedge \boldsymbol{b})_i = min\ (a_i,b_i)$ for any $\boldsymbol{a}$ and $\boldsymbol{b}$ in C.

Proof of Theorem 3.1

By arguments similar to those used by Landweber and Robertson [16] and by Lemmas 3.1 and 3.2 we can state that *C(PN)* is a lattice under the natural ordering of vectors.

For semi-modularity of the lattice, we must show that if two distinct elements $\boldsymbol{a}$ and $\boldsymbol{b}$ in C both cover an element $\boldsymbol{c}$ in C (in which case $\boldsymbol{c}$ is obviously *GLB(a,b)*), then both are covered by *LUB(a,b)*. By covering, we mean that $\boldsymbol{a}$ covers $\boldsymbol{c}$ if $\boldsymbol{c} < \boldsymbol{a}$ and there is no element $\boldsymbol{d}$ such that $\boldsymbol{c}<\boldsymbol{d}<\boldsymbol{a}$. In *C(PN)*, it is clearly seen that $\boldsymbol{a}$ covers $\boldsymbol{c}$ iff there is one transition in *PN* such that $a_i = c_i + 1$ and $a_j=c_j$ for all other transitions.

Let $\boldsymbol{a} = (a_1,a_2,...,a_n)$ and $\boldsymbol{b} = (b_1,b_2,...,b_n)$ where $\boldsymbol{a} \neq \boldsymbol{b}$. Since $\boldsymbol{c}$ is covered by both $\boldsymbol{a}$ and $\boldsymbol{b}$, we have that $\boldsymbol{c}$ differs from $\boldsymbol{a}$ in just one component, say $a_i \neq c_i$; similarly, $\boldsymbol{b}$ differs from $\boldsymbol{c}$ in just one component, say $b_j \neq c_j$. Note that $i \neq j$. Thus we have that under some marking M corresponding to $\boldsymbol{c}$ two transitions are enabled, t_i and t_j. Due to persistency, there must be markings M_1, M_2 and M_3 such that $M[t_i>M_1$, $M[t_j>M_2$, $M_1[t_j>M_3$ and $M_2[t_i>M_3$ with $\boldsymbol{a},\boldsymbol{b}$ and *LUB(a,b)* corresponding to M_1, M_2 and M_3, respectively. Thus $\boldsymbol{a}$ and $\boldsymbol{b}$ differ from *LUB (a,b)* by only one component, a_j and b_i respectively, which is, in each case, greater than the corresponding component of $\boldsymbol{a}$ and $\boldsymbol{b}$. Hence *LUB(a,b)* covers both $\boldsymbol{a}$ and $\boldsymbol{b}$, and we have proved the semi-modularity of the lattice *C(PN)*. **Q.e.d.**

Statement 3.1 *A marked graph is a persistent net.*

This is true because if a transition t_i of the marked graph is enabled, each of its incoming arcs (in a monochromatic version) contain at least one token and there is no way to remove this token other than to have this transition fired.

Theorem 3.2 *The cumulative diagram C(PN) of a marked graph PN is a distributive lattice with a zero element under the ordering of Definition 2.7(natural ordering of vectors).*

Proof.

1. It is easy to prove that if *C(PN)* is closed under the disjunction and conjunction, then *C(PN)* is a distributive lattice. In fact, with such assumptions and Lemma 3.2, $LUB\ (\boldsymbol{a},\boldsymbol{b})=\boldsymbol{a} \vee \boldsymbol{b}$, $GLB(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{a} \wedge \boldsymbol{b}$. Then

$GLB(\boldsymbol{a},LUB(\boldsymbol{b},\boldsymbol{c}))_j = (\boldsymbol{a} \wedge (\boldsymbol{b} \vee \boldsymbol{c}))_j = min(a_j, max(b_j,c_j)) = max\ (min(a_j,b_j),min(a_j,c_j)) = ((\boldsymbol{a} \wedge \boldsymbol{b}) \vee (\boldsymbol{a} \wedge \boldsymbol{c}))_j = LUB(GLB(\boldsymbol{a},\boldsymbol{b}),GLB(\boldsymbol{a},\boldsymbol{c}))_j$.
Similarly, the second distributivity law can be checked.

2. We now prove that for a marked graph *PN*, *C(PN)* is closed under disjunction and conjunction. In fact, Lemma 3.1 provides us with the fact that *C(PN)* is closed under disjunction.

To prove the second issue we need the definition of the inverse of a net.

Definition 3.3 *The inverse of a net* $PN = (S,T,F,M_o)$ *is the net* $PN' = (S,T,F',M_o)$ *where* $F' = \{(s,t): (t,s) \in F\} \cup \{(t,s): (s,t) \in F\}$.

Note 3.1 For a marked graph *PN*, the inverse *PN' preserves* all the basic properties, such as strong connectivity, liveness, safeness to name but a few. The reachability of *PN'* is equivalent to the *backward* reachabilty of *PN*.

Due to the above note we have that the number of firings of some transition t_i between any two markings *M'* and *M"* belonging to some sequence of reachable markings in a marked graph *PN* is equal to that of the same transition t_i between *M"* and *M'* belonging to the sequence of markings for the inverse.

Consider two elements $\boldsymbol{a}$ and $\boldsymbol{b}$ in *C(PN)*. Let $M(\boldsymbol{a})$ denote the marking corresponding to $\boldsymbol{a}$.

Thus there can be found two non-intersecting (except for the beginning and ending points) sequences of values of the transition firing vector $GLB(\boldsymbol{a},\boldsymbol{b}),\ldots, \boldsymbol{a},\ldots, LUB(\boldsymbol{a},\boldsymbol{b})$ and $GLB(\boldsymbol{a},\boldsymbol{b}),\ldots,\boldsymbol{b},\ldots, LUB(\boldsymbol{a},\boldsymbol{b})$ which correspond to the sequences of markings $M(GLB(\boldsymbol{a},\boldsymbol{b}),\ldots,M(\boldsymbol{a}),\ldots, M(LUB(\boldsymbol{a},\boldsymbol{b}))$ and $M(GLB(\boldsymbol{a},\boldsymbol{b})),\ldots,M(\boldsymbol{b}),\ldots,M(LUB(\boldsymbol{a},\boldsymbol{b}))$. Due to closure under disjunction, we have $LUB(\boldsymbol{a},\boldsymbol{b})_j = (\boldsymbol{a} \vee \boldsymbol{b})_j = max(a_j,b_j)$ for any *j* such that $1 \leq j \leq n$.

Consider now the inverse of *PN* initialised in marking $M(LUB(\boldsymbol{a},\boldsymbol{b}))$, which we denote as *PN'*. For such *PN'* we have two corresponding execution sequences beginning in $M(LUB(\boldsymbol{a},\boldsymbol{b}))$: $M(LUB(\boldsymbol{a},\boldsymbol{b})),\ldots, M(\boldsymbol{a}),\ldots,M(GLB(\boldsymbol{a},\boldsymbol{b}))$ and $M(LUB(\boldsymbol{a},\boldsymbol{b})),\ldots,M(\boldsymbol{b}),\ldots,M(GLB(\boldsymbol{a},\boldsymbol{b}))$, which are the inverses, both in terms of series of markings and those of transitions fired, of the sequences for *PN*, since the reachabilities, forward and backward, are equivalent for a marked graph and its inverse, respectively.

For these two sequences of markings of *PN'* we can build the corresponding sequences of values of the transition firing vector with respect to the initial marking $M(LUB(\boldsymbol{a},\boldsymbol{b}))$, which stands for the zero element of *C(PN')*.

For any transition t_j in *PN'* the firing numbers between $M(LUB(\boldsymbol{a},\boldsymbol{b}))$ and $M(\boldsymbol{a})$, $M(\boldsymbol{b})$ can be respectively expressed as:

$$a'_j = max(a_j,b_j) - a_j \text{ and } b'_j = max(a_j,b_j) - b_j \qquad (*)$$

Since the set *C(PN')* is closed under disjunction we have $LUB(\boldsymbol{a}',\boldsymbol{b}')_j = max(a'_j,b'_j)$ and it follows from (*) that $max(a'_j,b'_j) = max(a_j,b_j) - min(a_j,b_j)$.

Thus,

$$LUB(\boldsymbol{a}',\boldsymbol{b}')_j = max(a_j,b_j) - min(a_j,b_j).$$

Since the number of firings of t_j between $M(GLB(\boldsymbol{a},\boldsymbol{b}))$ and $M(LUB(\boldsymbol{a},\boldsymbol{b}))$ in *PN* is equal to that between $M(LUB(\boldsymbol{a},\boldsymbol{b}))$ and $M(GLB(\boldsymbol{a},\boldsymbol{b}))$ in *PN'*, we have

$$LUB(\boldsymbol{a},\boldsymbol{b})_j - GLB(\boldsymbol{a},\boldsymbol{b})_j = LUB(\boldsymbol{a}',\boldsymbol{b}')_j - GLB(\boldsymbol{a}',\boldsymbol{b}')_j = [max(a_j,b_j) - min(a_j,b_j)] - 0 = max(a_j,b_j) - min(a_j,b_j).$$

Also, $LUB(\boldsymbol{a},\boldsymbol{b})_j = max\ (a_j,b_j)$, and hence $GLB(\boldsymbol{a},\boldsymbol{b})_j = min\ (a_j,b_j)$. Thus *C(PN)* is closed under conjunction. Bearing in mind part 1 of the proof, *C(PN)* is a distributive lattice with zero $(0,0,\ldots,0)$. **Q.e.d.**

Without a full proof, which is rather lengthy, we claim the following.

Theorem 3.3 *The cumulative diagram C(PN) of a persistent and safe net PN is a distributive*

lattice with a zero element under the ordering of Definition 2.7(natural ordering of vectors).

Note 3.2 A net $PN = (S,T,F,M_O)$ is called *safe* iff for any marking M reachable from M_o $M(s) \leq 1$ for each $s \in S$.

Sketch of proof of Theorem 3.3.

The proof can be done using the fact that a safe and persistent net can be transformed into a marked graph by *duplicating* some transitions and places [26]. This duplication, however, preserves the strict ordering between firings of the marked graph transitions corresponding to the same transitions in the original safe and persistent Petri net.

The necessary behavioural equivalence of the cumulative diagrams, of the original net and the marked graph, is guaranteed by this transformation because potential multiple instances of the same transition, in the marked graph, can not be enabled concurrently. The complete proof, which is postponed to a subsequent paper, would of course require the following technique:
(1) defining a net with a multiset of transitions and(or) a labelled net, by introducing a labelling (functional) mapping of the set of transitions onto the set of transition names. (The transformation gives the names of the original net transitions as the labels for the marked graph net transitions);
(2) defining the set of labelled firing sequences, and the corresponding set of cumulative states of a labelled net (the so-called labelled cumulative states) with respect to the occurrences of the transition labels;
(3) defining the mapping of a set of original (unlabelled) cumulative states onto that of labelled ones, and then proving that this mapping preserves the partial order (implying homomorphism of the cumulative diagrams);
(4) proving that if the labelled *PN*, in any reachable marking, does not allow, in its behaviour, more than one transition with the same label to be enabled, then the unlabelled cumulative diagram is isomorphic to the labelled one.

It is worth noting that the effect of labelling, which is further exemplified in Section 4, discussing the merge paradigm, has also been examined in the context of deterministic causal automata [21].

Note 3.3 It is obviously not true that a persistent non-safe net always generates semi-modular non-distributive cumulative diagram. It is sufficient to take a non-safe marked graph. This means that the non-distributivity paradigm has a nature different from unsafeness or unboundedness. It stems from the purely causal reasons and the so-called "input conflict" (or "OR-causality" [21]), when an action may be initiated by alternative, but not mutually exclusive, causes. This kind of conflict would manifest its non-deterministic flavour in the backward reachability (for example, when it is impossible, for a marking with two tokens in the same place, to determine which one of the two concurrently fired transitions has first added a token into their shared output place).

Now we are ready to tackle our main objective on the relationship between partial order semantics and interleaving semantics. Although we should, first, consider an example.

Example 3.1 Consider two nets, *PN1* and *PN2*, as shown in Fig.1. They are both persistent. Furthermore, *PN1* is a marked graph. The corresponding marking diagrams, *MD1* and *MD2*, for these nets are shown in Fig.2. The execution sequences, in terms of the interleaving semantics, are given by *ES1 = {abc, bac}* and *ES2 = {abc,bac,acb,bca}*, for *PN1* and *PN2* respectively. The cumulative diagrams of transition firing number vectors, *C(PN1)* and *C(PN2)*, are shown in Fig.3. From these two diagrams one can deduce that for *PN1* a poset of cumulative states is *PO1 = ({000,100,010,110,111},≤)* and it forms a distributive lattice with zero *000*, and for *PN2* the poset is *PO2 = ({000,100,010,101,110,011,111}, ≤)*, which is semi-modular lattice with zero *000*. Since for $\boldsymbol{a}=101$ and $\boldsymbol{b}=011$, $\boldsymbol{c} = \boldsymbol{a} \wedge \boldsymbol{b}$ is determined as $c_i = min\ (a_i,b_i)$, $i = 1,2,3$, and hence, $\boldsymbol{c} = 001$, whereas the $GLB(\boldsymbol{a},\boldsymbol{b}) = 000$, we deduce that *PO2* is not distributive.

From this example we can see that there must be some relationship between the lattice-theoretical class of a process described by the net (here we had a distributive process depicted by the marked graph and semi-modular process associated with the persistent non-safe net) and the possibility to construct a unique partial order on the process events. To characterise such a relationship we first

need to define a *process* as an acyclic net (often called an *occurrence net* [19]) in which every transition fires at most once thus forming a unique process event. The appropriate definition can be found in [27].

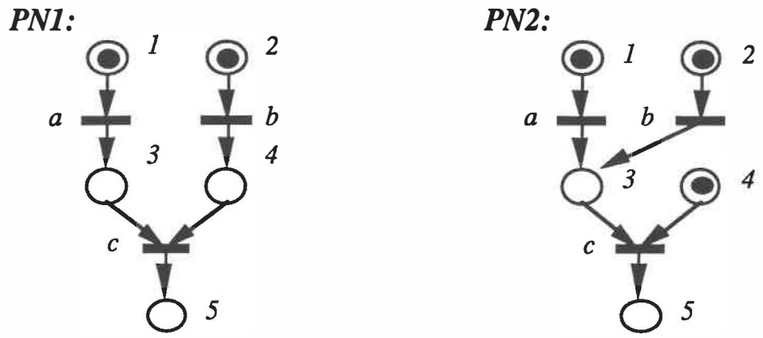

Figure 1. Petri nets for Example 3.1

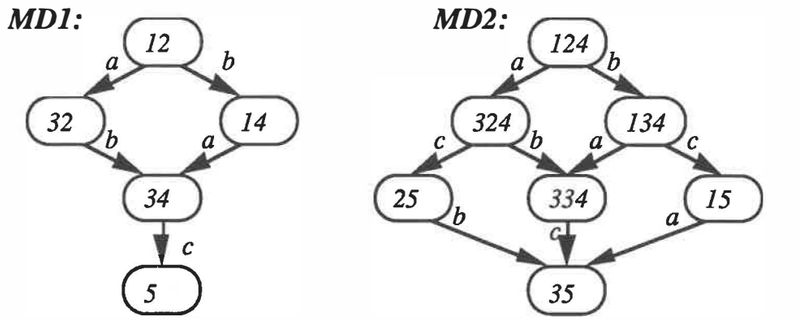

Figure 2. Marking diagrams for Example 3.1

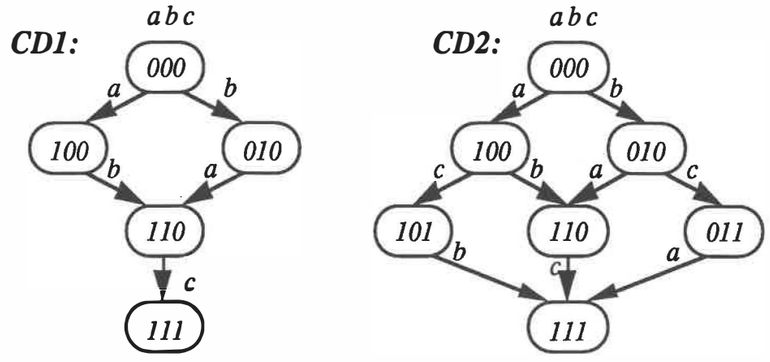

Figure 3. Cumulative diagrams for Example 3.1

For a process, the poset of transition firing vector values would of course be defined on binary n-tuples where n is the number of transitions. Each component a_i of such cumulative state $\boldsymbol{a}$ would either be zero (transition t_i has not yet fired) or one (transition t_i has already fired). If we restrict ourselves to considering only finite length (n is finite) processes, which is justifiable here because the termination/liveness paradigm has no effect upon the lattice-theoretical characterisation of concurrency semantics of processes, we can also introduce, like in Example 3.1, the notion of a set of execution sequences as another process definition.

Let ES be a set of process execution sequences each of which is defined on a finite alphabet T of actions (called transitions), i.e. $ES \subset T^*$ where T^* is the set of all finite-length sequences of elements in T. Let also $PREF(ES)$ be the set of all prefixes of sequences in ES. For each element u in $PREF(ES)$ we associate a binary n-tuple $\boldsymbol{a}$ such that $n=|T|$ and for each $i=1..n$, $a_i=1$ if t_i occurs in u, and $a_i=0$ if it does not. After applying the equivalence relation to the elements of $PREF(ES)$, by which the sequences corresponding to the same n-tuples collapse into one element (*equivalence class*) [3], we obtain a new form of a set of traces, denoted as ES°, where each element is in one-to-one correspondence with a transition firing vector value. If we construct a partial order relation, $\leq$, over this set, which is based on the prefix-order relation between sequences of $PREF(ES)$ up to permutations, we shall come to the conclusion that ES° is also a poset. Furthermore, if ES is the original set of execution sequences for a process net PN, then it is clear that due to the above theorems ES° is a semi-modular lattice, with respect to $\leq$ being the natural ordering of vectors, if PN is a persistent net, and it is a distributive lattice if PN is a marked graph.

Note 3.4 The definition of a process, where each transition may fire only once (so-called *single-run execution*), makes it unnecessary to consider persistent and safe nets as a special case because their single run-execution would be represented as a marked graph.

Now consider the problem of how to construct a partial order semantics on the set of process actions from the interleaving semantics of the set of execution sequences. We claim the following.

Theorem 3.4 *Let us be given the interleaving semantics, ES, of some process which produces some poset of cumulative states, denoted as ES°. This poset is built over the set of sequences in PREF(ES) prefix-ordered up to permutations. A partial order whose sequentialisations generate the same ES°, and therefore satisfying ES, can be built on the set of process actions iff ES° is a distributive lattice with respect to the partial order.*

Proof

(i) Assume a process satisfying ES° can be defined by a partial order on the set of its actions. Then it is possible to construct the corresponding marked graph net PN by associating a transition with each action and a place with each ordering pair. Furthermore, we should add the input place for each minimum action in the partial order. These additional arcs are then marked with tokens. Due to the construction of sets ES° and $C(PN)$ and Theorem 3.2, the set $C(PN)$ is a distributive lattice with zero, and so is ES°.

(ii) Assume ES° is a distributive lattice with zero. To prove the possibility of building a partial order from ES we need to define the following two relations on the set of process actions.

Two actions a and b are said to be in the $<$ relation, called "*a precedes b*" and denoted as $a < b$ iff a precedes b for each sequence of ES which they occur in. Two actions are said to be in the $\|$ relation, called "*a is independent of b*" and denoted $a \| b$, iff $\wedge(a < b)$ & $\wedge(b < a)$.

Thus using the $<$ and $\|$ relations we can build a partial order on the set of actions. This order is obviously unique for the given ES because of the way the $<$ and $\|$ relations are built. The only thing that remains to be treated is that whether this partial order will generate exactly the same ES° set. Let it generate some $ES^{\circ\prime}$. It is clear, by the proof of (i) that $ES^{\circ\prime}$ is also a distributive lattice.

Without loss of generality, assume that $ES^{\circ\prime} \neq ES^\circ$ and let $ES^{\circ\prime} \supset ES^\circ$. The latter means that the cumulative diagram for $ES^{\circ\prime}$ must contain some additional cumulative states. If these states are intermediate, i.e. covered each by at least one of the states from ES°, then it contradicts to the distributivity of ES°. If this is a set of additional states forming their own independent sequences in $ES^{\circ\prime}$, then it is easily seen that the partial order which has been built from ES° will be different from the order which has generated $ES^{\circ\prime}$ (it will contain more $\|$ pairs, and less $<$ ones), which is,

again, a contradiction. **Q.e.d.**

Note 3.5 We should bear in mind that in constructing a partial order for some *ES* whose *ES*° is not distributive, it is possible to obtain the order which, being converted into a marked graph *PN* by the technique described in (i), will generate another interleaving semantics *ES'*. Of course, *ES'*° is a distributive lattice. It may differ from *ES*° by some additional prefixes which have not been presented in *ES*°. These new prefixes will correspond to some additional cumulative states in *C(PN)* which are covered by the states of the original lattice. Here we emphasize the fact that the distributive lattice is the greatest possible set that "contains" prefixes of the execution sequences which comply with the given partial order on the process actions (Cf., the synchronisation [3] and weaving, or shuffle, [18] operators on the set of traces would be examples of alternative manifestations of distributivity.)

Example 3.1 (continued) Let us build a partial order consistent with *ES2* using the < and || relations introduced above. We obtain the following: $a \parallel b$, $b \parallel c$, $a \parallel c$. For such a partial order we can construct a marked graph net shown in Fig.4. This net generates interleaving semantics *ES2'* = *ES2* ∪ {*cab*, *cba*}. *ES2'*° is a distributive lattice.

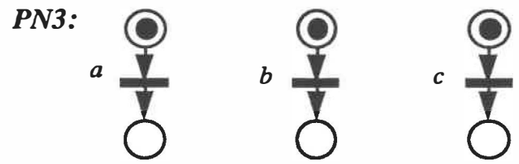

Figure 4. Marked graph for a distributive process

We conclude this section with a reference to the work of D.E. Muller (it is well presented in [28]), who made very similar observations on the taxonomy of lattice characterisations for speed-independent asynchronous circuits. Muller had reportedly shown that for specifying distributive circuits (those that generate distributive cumulative diagrams of logical element switchings), one can use the so-called change charts, which appear to be posets labelled with switching actions. We can thus use marked graphs for most succinct specifications of concurrent logical circuits if we need to obtain distributive processes in these circuits [29].

Semi-modularity and the Merge Paradigm

In this section we outline a closer (though in a less formal style) look at the processes which have been characterised in the above lattice framework as semi-modular. The class of semi-modular processes contains that of distributive ones. So we are especially interested here in semi-modular and non-distributive processes. Such processes can be specified by persistent, but non-safe, nets. An example of such process was presented in Fig.1 as *PN2*. Looking at the behaviour of this net more carefully we notice that transition *c* can fire as a result either of transition *b* or transition *a*, and not necessarily as a result of the both of them. Furthermore, if we do not distinguish between the tokens "coming from" *a* and *b* to place *3*, we have a rather deterministic picture where *c* "doesn't care" whose token it is enabled by and it fires rather "indifferently" bringing again such an undistinguishable token to its output place *5*. This operational determinism results in that the cumulative diagram for such a net (Fig,3, *C(PN2)* is semi-modular and has the greatest element *111*. The corresponding marking diagram *MD2* shown in Fig.2 represents a confluent [1] transition system.

Now look at another way of interpreting the process *PN2*, which was suggested to the author by W.Reisig [30]. We redraw *PN2* in such a way as to make the tokens in places *1* and *2* have two different values (colours) as shown in Fig. 5. We denote it as *PN2'*. Assume this net has the semantics which is represented by the marking diagram *MD2'* shown in Fig.6. It has two final states, which the process may enter in either way. This depends on which of the values, **A** or **B** token, first passes through *c* from place *3* to place *5*. To characterise this value-dependent non-determinism of *PN2'* in terms of lattices of cumulative states we can extend our definition of the "non-coloured" transition firing vector to the case of the coloured firing vector. Thus for *PN2'* we have a cumulative diagram *C(PN2')* as shown in Fig.7. One can clearly see that for 0,**A** and **B**

ordered as 0 < **A** and 0 < **B**, this is not a lattice because **ABA** and **ABB** have no *LUB* in it.

PN2':

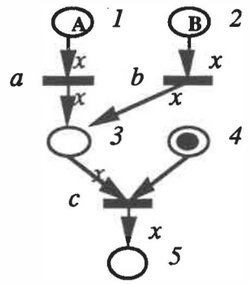

Figure 5. High-level Petri net with non-deterministic behaviour

MD2':

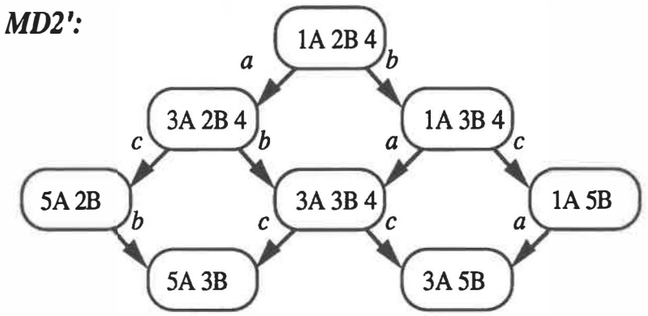

Figure 6. Marking diagram for Petri net in Fig.5

CD2':

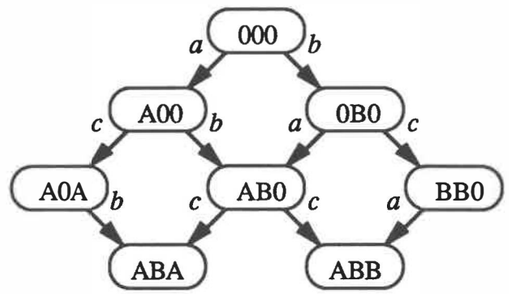

Figure 7. Cumulative diagram for Petri net in Fig.5

From these considerations, which may look just a special case, we should however claim the following general statements establishing the relationship between control flow semantics, defined as the behaviour under the assumption that all tokens are indistinguishable, and data flow semantics, defined under the assumption that some tokens may be associated with data values (colours).

Statement 4.1 *Purely control flow semantics of a semi-modular non-distributive process is deterministic, whereas its data-flow semantics may be non-deterministic.*

Statement 4.2 *Both control and data flow semantics of a distributive process are deterministic.*

Note 4.1 Under the definition of data-independence introduced by M.. Rem [18], we can also claim as a corollary:

Corollary 4.1 *Distributive processes are data-independent.*

The data-flow non-determinism of non-distributive processes has a very interesting paradigm which is often regarded as a merge operator. The functional semantics of *merge* has been defined for example in [2] where the *merge* function has two input data sequences *Left1* and *Left2* and one output sequence *Right* which merges *Left1* and *Left2* in either way.

The functional specification of *merge* can be done in a number of ways. For example, in a Lisp-like notation:

```
merge (Left1, Left2)
    if Left1 = NIL then Left2
             else
    if Left2 = NIL then Left1
             else
    cons(car(Left1),merge(cdr(Left1),Left2))
              or
    cons(car(Left2),merge(Left1,cdr(Left2))
    fi
    fi
```

The "*or*" in the above specification is the source of non-determinism of the resultant *Right* sequence, which can be an arbitrary interleaving of the *Left1* and *Left2* sequences providing only that the projection of *Right* on the alphabet of *Left1* channel is equal to *Left1*, and the projection of *Right* on the *Left2* channel alphabet is equal to *Left2*.

Our example of the *PN2'* net (Fig.5) is a net specification of the merge operator for the case of one element sequences.

Another interesting paradigm of generating a semi-modular but non-distributive lattice behaviour is attributable to the introduction of *labels* on the process transitions. In other words, if we consider the definition of a *labelled process net* as a tuple (S,T,F,M_o,W,L) where S,T,F,M_o have the same meaning as for a non-labelled net, and W is a finite alphabet of *labelling symbols*, labels, and $L:T \rightarrow W$ is a *labelling function*, which assigns certain labels to all the transitions of the net. Of course, it is possible that this function may assign the same label to more than one transition.

Now if we introduce the same partial order framework on the behaviour of such a labelled net as we did for nets in terms of cumulative diagrams on the set of transition firing vectors, but here the vector components would stand for the labels rather than for transitions, we may obtain a different lattice-oriented semantics of the given net. (Cf., Hint for proof of Theorem 3.3.) For example, having been given a labelled marked graph we may have a distributive lattice of the transition firing vector values but the partial order built on the firing vector values for labelled actions may be a non-distributive lattice, if, for some reachable markings, there is at least one pair of concurrently enabled transitions that have same labels. The following example illustrates this paradigm.

Example 4.1 Let a labelled marked graph be given in Fig.8. This graph generates a deterministic behaviour whose marking diagram converges to the marking corresponding to a token in place 5. However, we can distinguish our interpretation of the process observing separately two different cumulative diagrams, one for the transition firing vector, as shown in Fig.9, and the other, for the labelled action firing vector, presented in Fig.10. The first forms a distributive lattice because the given net is a marked graph, while the second, which is homomorphic to the first (with respect to the mapping defined in Hint for Theorem 3.3), is a semi-modular, non-distributive lattice because action *c* can be enabled for the first time either as a result of *a* or *b*. This example shows that it is possible to change the lattice observational view on the process not only by means of (i) token distinction (data dependency), but also by (ii) introducing a labelling function, which may assign the same label to some mutually concurrent transitions, thus resulting in non-distributive semantics

(action labelling dependency).

Note 4.2 It is interesting to point out again that interleaving semantics, represented in this example by the cumulative diagram in Fig.10, does not give full information about the causal structure of events because it may correspond to some other nets, for example, to the non-safe one with a place which is shared as an output place by transitions *a* and *b* and which is the input place for transition *c*.

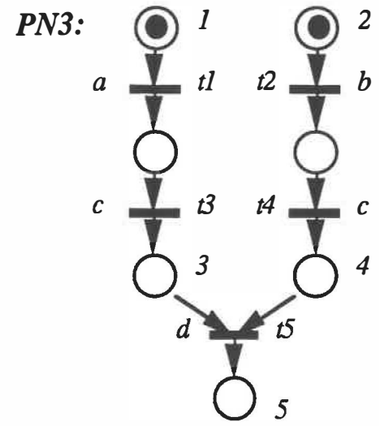

Figure 8. Marked graph with labelled transitions

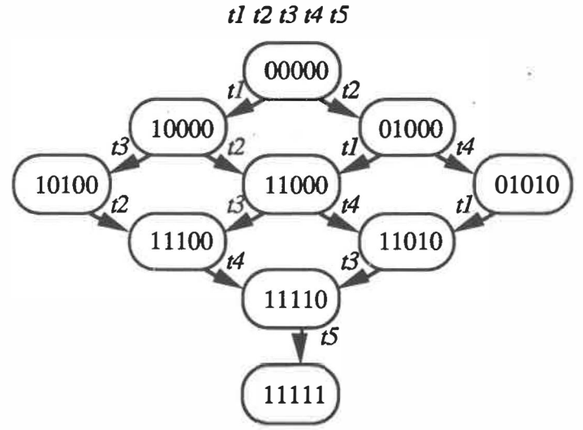

Figure 9. Cumulative diagram for the transition firing vectors

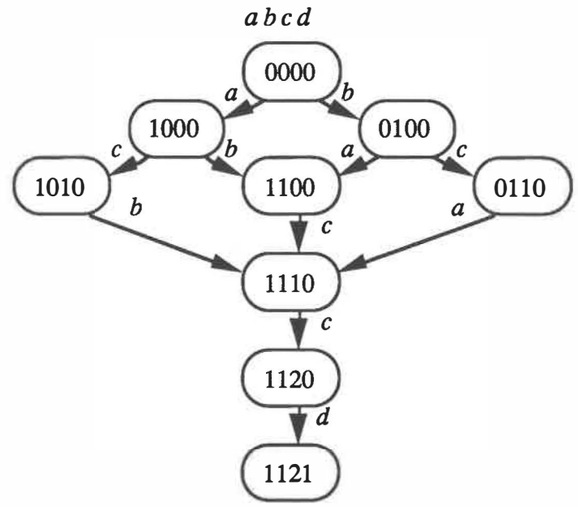

Figure 10. Cumulative diagram for labelled action firing vectors

Some Practical Implications of Non-distributivity

The practice of parallel programming, which is applied to both software and hardware design domains, shows that the constructs used for building concurrent programs have either deterministic process semantics, which can always be described by partial orders (distributive, or purely parallel processes), or non-deterministic process semantics related with such constructs like mutual exclusion or general merge operators (non-semi-modular, or non-confluent processes). There is however an interesting class of computational behaviour, which corresponds to semi-modular but non-distributive process semantics and can be associated with a special construct called a *control flow merge*. The latter provides not only some specific functionality, which is often overlooked or disregarded as "unnecessary or undesirable" (Cf., discussion in Introduction), but, in many cases, helps to achieve better performance of the specified concurrent system.

Wired-OR Logic Synchronisation

The *self-timed design principle* used in constructing concurrent hardware [29,31] demands that each signal in such a system must be "*acknowledged*" by other signals through causal relationship between them. Thus, the system operates safely, without *hazards*, and it is impossible that an activated signal (for some 0-1 or 1-0 transition) is disabled without reaching its goal state. Usually, to achieve such a feature, the *synchronisation* between signals is organised by means of the so-called Muller *C-elements* [28], which function as some event-oriented AND elements. The two

input C-element,which is sometimes called a synchronisation flip-flop (it has an internal feedback interconnection), is defined by the following logical function:

$$y = x_1 x_2 \vee (x_1 \vee x_2) y$$

where x_1 and x_2 are the input signals and y is the output (and feedback) signal. In both transition phases, the value of y is changed last with respect to the changes of x_1 and x_2. This ordering, corresponding to the required discipline of using a C-element, is depicted by the marked graph net in Fig. 11.

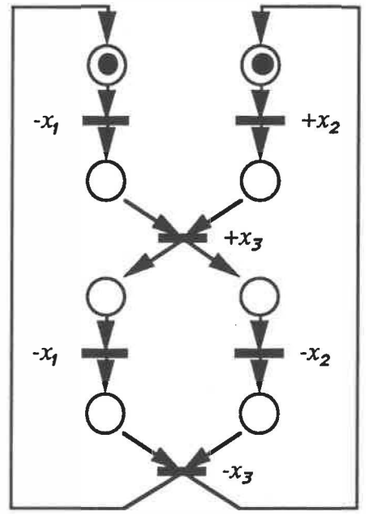

Figure 11. The marked graph specification of C-element behavior

From this specification we can see that the process associated with this element is distributive. Let us now assume that we need to provide synchronisation of a group of modules whose number is large enough, and it is impossible to interconnect them to a multi-input C-element.

In order to synchronise modules in a *distributed* (but non-distributive !) way, we can use a *wired-OR logic*, connecting modules to a fixed number of wires (independently of the number of modules synchronised and performing the synchronisation in a safe, deterministic way). It has been proved elsewhere [32] that the *least possible number* of wires needed for such synchronisation is three and the module signal transitions on these wires are *cyclically shifted* through these three wires, say, x,y and z, executing at each cycle both the transition-AND and transition-OR synchronisation actions. The corresponding behaviour, on a pair of wires, say, x and y, can be specified in the way shown in Fig. 12 (for the case of two modules).

This figure presents a one-third part of the whole specification, and it is easily seen that this process is non-distributive: the transition labelled with $+x$ may be enabled either after the $+x_1$ or $+x_2$ labelled transition. Thus, here, the use of the non-distributivity paradigm helps to have an elegant distributed synchronisation instead of having a bunch of individual wires coming to a common C-element. The part of the dynamic behaviour of a wired-OR "logic element" related to the above transitions on wire x plays the same role as a one-element merge operator with the purely control flow semantics - it merges the result of actions $+x_1$, $+x_2$ etc on the same wire x, which exhibits the $+x$ transition as an acknowledgement of the first of the OR-causal actions.

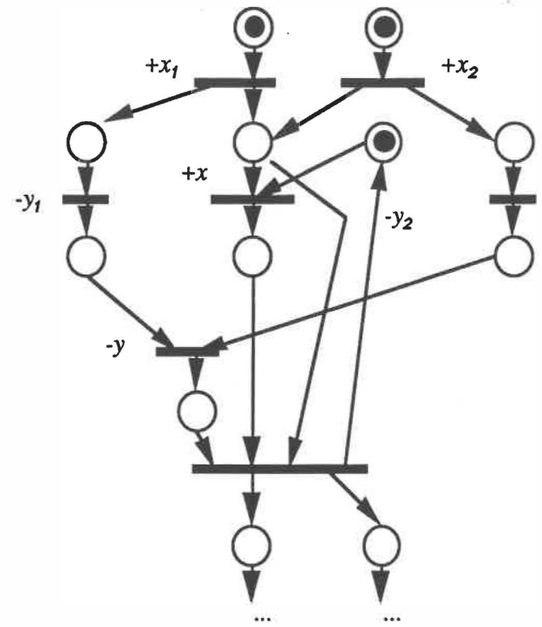

Figure 12. The Petri net specification of wired-OR logic operation

Hardware Structures with Redundancy

Another example of organising non-distributive computations can be found in hardware structures with *redundancy*. Let us have a group (again, for simplicity, we take two) of *functionally equivalent*, but possibly working with different speeds, *modules* which produce the results of the same tasks onto the *common data bus*, as shown in Fig.13.

They do it in the manner where the computation result is taken from the first-to-complete module and the subsequent completion of the task in the other module is needed only to make the comparison of the result of each module with the result established on the merging bus, with the possibility to make a backward recovery action in the case of a mismatch. If there is no mismatch, the normal computation flow can be already quite ahead of the current point of the result matchings, thus making the whole fault-tolerance mechanism operate, generally, *faster* than in conventional schemes, in which a voting element waits for the completion of the tasks in all the redundancy modules before producing the final result to the next computation stage.

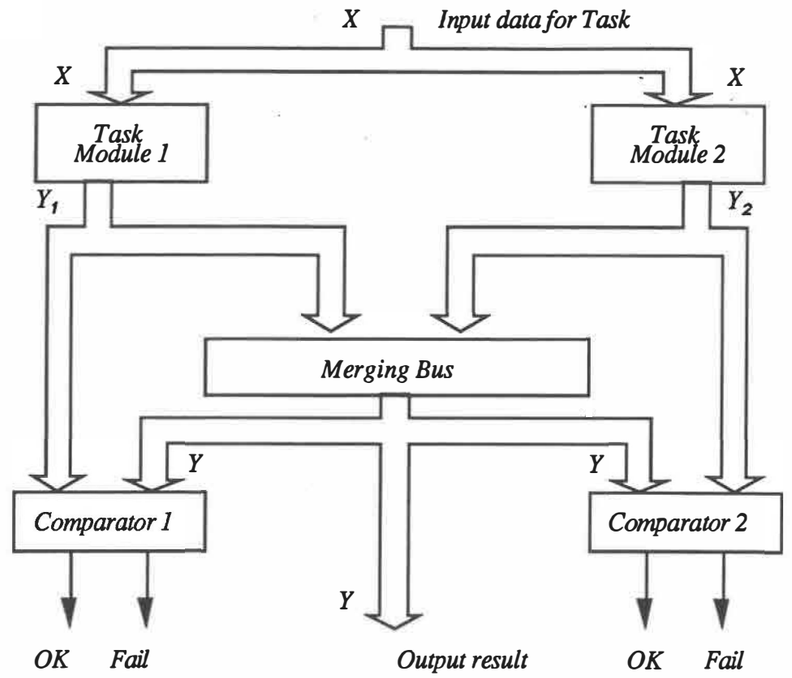

Figure 13. A hardware structure with redundancy

Scheduling Tasks on Limited Resources

The final example of a non-distributive but deterministic computation is taken from the domain of analysis of dynamic software system behaviour.

Let us have a group of n independent tasks, say, in an operating system kernel, which *may* be executed concurrently because they have no dependency on data. Unfortunately, the number of available resources for their concurrent execution, denoted by m, is such that $m < n$. again, for simplicity, take $n = 3$ and $m = 2$.

Since there is no way to run all the three tasks concurrently, as for example was possible in Fig.4 for actions a,b and c standing for the tasks, whose duration is denoted by T_a, T_b and T_c respectively, the natural way for achieving maximum performance would be, first, to run two of them and then, upon the completion of the fastest of them, to allocate the released resource to the third task. This organisational discipline can be depicted by the semi-modular but non-distributive process shown in Fig.1, *PN2*. This scheduling strategy provides some intermediate situation between two adjacent distributive computation structures, a fully concurrent computation of Fig. 4 and a too-restrictive computation of Fig.1, *PN1*.

To show the performance gain, which is intuitively quite obvious, we compare the time values for these three computation structures. If we denote the computation times for the scheduling schemes of Fig.1, *PN1*, Fig.1, *PN2* and Fig.4 as *T1*, *T2* and *T3*, respectively, we can easily derive the expressions:

$$T1 = max\ (T_a,T_b) + T_c,$$
$$T2 = max\ ((min(T_a,T_b) + T_c),T_a,T_b),$$
$$T3 = max\ (T_a,T_b,T_c)$$

Obviously, $T1 \geq T2 \geq T3$.

Thus if the number of resources is limited, one may resort to a schedule, whose behaviour is non-distributive, in order to eliminate the performance loss pertaining to using a partially ordered schedule, which would fit the original partial order, determined by data dependencies etc, into the limited resource framework.

Those above and some other examples demonstrate that the semi-modularity paradigm is by no means a theoretical exercise, superfluous in real life. Rather, it can support certain modes of concurrent computations (distributed synchronisations, structural redundancy, task scheduling etc.) in a much more efficient way than can be achieved using just partial order (AND-causality) structures.

Speed-Independent versus Delay-Insensitive Circuits

The dichotomy between semi-modularity and distributivity (or, in the sense of [20], between {AND,OR}- causality and {AND}-causality) brings also some analytical power into the area of dichotomy between speed-independence and delay-insensitivity, which makes a special interest for the asynchronous design community [23].

It is now commonly accepted that an asynchronous hardware circuit is called *speed-independent* if its behaviour is *correct* and *insensitive* to the *delays* of logical elements (elementary gates) that constitute the circuit, and the delays of interconnecting wires are negligible. A circuit is called *delay-insensitive* if its behaviour is correct and insensitive both to the *gate and wire delays*. Hence, the class of delay-insensitive circuits is narrower than that of speed-independent ones.

The word "correct" in the above definitions has the following meaning: the circuit is free from logical hazards and every signal transition, often called a signal change, produced at the output of a component (a logical gate, for speed independence, and a gate or a wire, for delay-insensitivity) is "acknowledged" by some other component before the component is again committed to change its output signal.

It was shown by D.E.Muller (see [28]) that circuits whose behaviour is described by a semi-modular cumulative diagram on the set of cumulative states, which are the values of signal transition vector, has correct a behaviour that is independent of delays of the components whose output signals are the components of the cumulative states. According to this property, a circuit with a semi-modular behaviour with respect to its logical gates is speed-independent.

Assume, again, that the circuit is described by a semi-modular diagram with respect to its gates. Can this circuit be delay-insensitive? The answer is not clear until we check if the behaviour of the circuit, which is now defined on an extended set of signals because we also include the wires as separate components, will remain semi-modular. Staying on general terms of analysis, we can easily prove the following important property.

Statement 5.1 *If a circuit which is speed-independent has a semi-modular but non-distributive behaviour with respect to its gates, this circuit is not delay insensitive.*

Proof

Assume, to the contrary, that the circuit, whose behaviour is semi-modular but non-distributive, is delay-insensitive. This means that the circuit is insensitive to the delays both in wires and gates and every transition on the output of each component has a chance to be acknowledged by some other component in the circuit. The latter fact concerns also each wire in the circuit (of course, we should only consider the wires that serve as interconnections between the gates, rather than those interconnections which are internal to the gates). Now, because of non-distributivity, we can find a signal whose transition is caused by the OR-causality of a pair (without loss of generality, we can assume two signals as OR-causes) of signals, i.e. this transition can occur as a result of either one or the other transition. It is clear that, in order to implement this causality physically, the circuit must contain wires interconnecting the outputs of gates whose signal transitions are the OR-causes of the given transition to the input of the gate whose signal transition is the given, "non-distributive", transition. Now if we take these two wires as separate components whose output signal transitions must always be acknowledged, we come to the situation where because the gate whose output exhibits the given transition acknowledges, under OR causality, only one of the two cause-transitions produced by the above wires, we fail to acknowledge the other transition

and, hence, the corresponding wire's delay may affect the circuit operation. If the circuit is cyclic in operation (so called live) then it is possible that the next signal transition will be initiated on that wire before the previous transition has completed. So we come to contradiction. The circuit is not delay-insensitive. **Q.e.d.**

From this statement and the fact that a wire (we do not allow any wired-OR interconnections outside the components) cannot be the element whose signal transition is the effect of OR-causality - the wire has only one input signal, we can deduce another important claim.

Statement 5.2 *If a circuit is delay-insensitive, its behaviour is distributive with respect to all components, both gates and wires.*

Concluding Remarks

We have presented a lattice-theoretical characterisation of some important semantic notions in analysis of concurrent systems. Our main effort has been applied to the subclasses of Petri nets, persistent, safe and persistent and marked graph nets, which produce a behaviour definable in terms of posets of cumulative states of transition firing numbers. The lattice properties of such behaviour has led us to the important subclasses of computational behaviours, or processes, semi-modular and distributive ones, so we have been able to treat the problem of relationship between the descriptive powers of interleaving semantics and partial order semantics of processes (single-run executions).

The characterisation presented fits quite well within the domain analysis of event structures of Winskel and the boolean-algebraic analysis of causal automata of Gunawardena. Furthermore, because lattices on cumulative states of the net behaviour can generally provide more descriptive power than Winskel's domains on event structure configurations our approach sheds more light on the subclass of persistent Petri nets, and particularly marked graphs, for which Winskel's safeness requirement can be lifted.

This approach has also produced some fruitful results in analysing deterministic nature of data-valued semantics of processes, which looks quite promising for further investigation of semantics of high-level nets.

The distinction of a subclass of non-distributive processes from the class of semi-modular (confluent in Milner's sense and {AND,OR}-causal in Gunawardena's sense) ones has yielded interesting pragmatic implications in the areas of circuit and program design. For example, the analysis of speed-independence/delay-insensitivity dichotomy can further help a formulation of synthesis restrictions such as: it is impossible to construct a delay-insensitive circuit for a specification of a semi-modular and non-distributive behaviour, or there does not exist any delay-insensitive implementation for speed-independent circuit whose behaviour is non-distributive.

The paper thus points to several topics for further research, particularly claiming attention to:

(1) a more rigorous unification of the results on lattice characterisation of persistent nets with Winskel's domain analysis of safe nets,
(2) a deeper investigation of determinism and confluence in data-flow semantics of high-level nets,
(3) a formulation of a stricter link between implementability conditions for delay-insensitive circuits and the classes of their behavioral specification.

Acknowledgement

I would like to thank my friend and colleague, Leonid Rosenblum, for the endless hours of joint work and his brilliant advice, as well as Jeremy Gunawardena, Anthony Mazurkiewicz and Wolfgang Reisig for interesting discussions on the subject . I am also grateful to the Department of Computer Studies at the Polytechnic of Wales for their facilities and support during my stay there as a lecturer.

Appendix

Although the main part of the paper, e.g. Section 3, tacitly assumes that the reader is familiar with the basic notions of partial order and lattice theory, this brief supplement may appear useful, at least to provide a background for mutual understanding between the reader and the author on the usage of the following definitions.

Let S be a non-empty set and $\leq$ a partial order relation. Then the pair $(S,\leq)$ denotes the partially ordered set (poset).

Definition A.1 An *upper bound* of a pair of elements (a,b), $a,b \in S$, denoted as $UB(a,b)$, is an element $c \in S$ such that $a \leq c$ and $b \leq c$. A *lower bound* of a pair of elements (a,b), $a,b \in S$, denoted as $LB(a,b)$, is an element $c \in S$ such that $c \leq a$ and $c \leq b$.

Definition A.2 A *least upper bound* of a pair of elements (a,b), $a,b \in S$, denoted as $LUB(a,b)$, is an element $c \in S$ such that $c=UB(a,b)$ and for all other d, $d=UB(a,b)$, $c \leq d$. A *greatest lower bound* of a pair of elements (a,b), $a,b \in S$, denoted as $GLB(a,b)$, is an element $c \in S$ such that $c=LB(a,b)$ and for all other d, $d=LB(a,b)$, $d \leq c$.

Definition A.3 A poset $(S,\leq)$ is said to be a *lattice* iff every pair of elements (a,b), $a,b \in S$ has both $LUB(a,b)$ and $GLB(a,b)$ within S.

Definition A.4 For two elements a and b of a poset $(S,\leq)$, $a,b \in S$, such that $a<b$, we say that b *covers* a iff there is no element $c \in S$ such that $a<c<b$.

A usual graphical representation of a poset is a Hasse diagram, in which vertices correspond to the elements of the poset and arcs stand for the covers relation.

Definition A.5 A lattice $(S,\leq)$ is said to be *semi-modular* iff for every pair of elements (a,b), $a,b \in S$, such that both a and b cover $GLB(a,b)$, $LUB(a,b)$ covers both a and b.

Definition A.6 A lattice $(S,\leq)$ is said to be *distributive* iff for any elements $a,b,c \in S$ the following distributive law is satisfied:

$$GLB(a,LUB(b,c)) = LUB(GLB(a,b),GLB(a,c)).$$

It is known that an alternative definition based the other distributive law (dual to the above with respect to GLB and LUB) is also true, which is a characteristic property of distributive lattices.

The relationship between the above two classes of lattices is known to be such that distributive lattices are a subclass of semi-modular ones.It may also be of some interest to note that another class of lattices, modular ones, which is known in lattice theory as a "half-way" between the above two, has no adequate paradigm in the semantics of concurrent processes, or better say in the "physics of computations".

An almost standard technique of analysis of a mathematical model as a lattice prescribes that the model is first presented as a poset of elements, typically having some structure, with the $\leq$ relation defined in terms of more primitive relations on the components of the structure of the poset's elements. The structure of elements is also used as a basis for defining and proving the intuitive meaning of the LUB (or, sometimes called, "join") and GLB ("meet) elements in terms of some appropriate algebraic binary operators, such as disjunction, max etc and conjunction, min etc, respectively. This should finally facilitate giving a positive or negative answer to the question: whether the poset is a lattice or not, by checking if the poset is closed under these operations. It is also possible, using the values of primitive components, to define the cover relation between elements in an adequate and straightforward way, so as to help querying about the lattice's semi-modularity.

References

1. Keller, R.M., A fundamental theorem of asynchronous parallel computation, LNCS, Springer-Verlag, Berlin, No.24, (1975).
2. Hoare, C.A.R., Communicating Sequential Processes, Prentice-Hall International, Englewood Cliffs, NJ, 1985
3. Mazurkiewicz, A., Trace theory, LNCS, Springer-Verlag, Berlin, No.255, (1986).
4. Reisig, W., Petri Nets - An Introduction, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, (1985).
5. Milner, R., A Calculus for Communicating Systems, LNCS, Springer-Verlag, Berlin, No.92, (1980).
6. Proceedings of the Workshop on Semantics for Concurrency, Leicester University, U.K., July 1990, Spinger-Verlag, Berlin, (1990).
7. Seminar on Concurrency, Carnegie-Mellon University, Pittsburgh, PA, July 1984, LNCS, Springer-Verlag, Berlin, No.197, (1985).
8. Tiusanen, M., Some unresolved problems in modelling self-timed circuits using Petri nets, Bulletin of EATCS, October 1988, No. 36
9. Janicki, R., and M. Koutny, On equivalent execution semantics of concurrent systems, LNCS, Springer-Verlag, Berlin, No. 266, (1987).
10. Pratt, V.R., Modelling concurrency with partial orders, Int. Journal of Parallel Programming, Vol.15, No.1, (1986).
11. Castellano,L., DeMichelis, G., and L. Pomello. Concurrency versus interleaving: an instructive example, Bulletin of EATCS, February 1987, No.31.
12. Benson, D.B., Concurrency and interleaving are equally fundamental, Bulletin of EATCS, October 1987, No.33.
13. Reisig, W., Concurrency is more fundamental than interleaving, Bulletin of EATCS, June 1988, No.35.
14. Probst, D.K., and H.F.Li, Modelling reactive hardware processes using partial orders, Workshop on Semantics of Concurrency, Leicester University, July 1990, Springer-Verlag, 1990.
15. Rosenblum, L., Yakovlev,A., and V.Yakovlev, A look at concurrency semantics through "lattice glasses", Bulletin of EATCS, February 1989, No. 37.
16. Landweber, L.H., and E.L. Robertson, Properties of conflict free and persistent Petri nets, Jornal of ACM, Vol.25, No.3, July 1978.
17. Genrich, H., and K.Lautenbach, System modelling with high-level Petri nets, Theoretical Computer Science, Vol. 13, 1981.
18. Rem, M., Trace theory and systolic computations, LNCS, Springer-Verlag, Berlin, No.258, (1987).
19. Winskel, G., Event structures, LNCS, Springer-Verlag, Berlin, No.255,(1986).
20. Gunawardena, J., private communication, 1990.
21. Gunawardena, J., Causal automata I: confluence ≡ {AND,OR} causality, Proc. Workshop on Semantics of Concurrency, Leicester University, July 1990, Springer-Verlag, 1990.
22. Milner, R., Communication and Concurrency, Prentice-Hall International, London, 1989.
23. Martin, A.J., The limitations to delay insensitivity in asynchronous circuits, Proc. MIT Conference on Advanced Research in VLSI, MIT Press, 1990
24. Birkhoff, G., Lattice Theory, Providence, RI, 1967.
25. Peterson, J., Petri Net Theory and the Modelling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
26. Ramamoorthy, C.V., and G.S. Ho, Performance evaluation of asynchronous concurrent systems using Petri nets, IEEE Transactions on Software Engineering, Vol. SE-6, No.5, September 1980.
27. Best, E., Fernandez, C., and H.Plunnecke, Concurrent systems and processes, manuscript, 1985.
28. Miller, R.E., Switching Theory, Vol.2, Wiley and Sons, NY, 1965.
29. Yakovlev, A., and A. Petrov, Petri nets and parallel bus controller design, Proc. 11th Int. Conference on Applications and Theory of Petri Nets, Paris, France, June 1990.
30. Reisig, W., private communication, 1989.
31. Yakovlev, A., Designing self-timed systems, VLSI Systems Design, September 1985.
32. Varshavsky, V.I., et al., Implementation and analysis of the TRIMOSBUS self-clocking interface, Automatic Control and Computer Science (USA), Translated from Russian, Allerton Press, Vol. 19, No.4,(1985).