3. Ginzburg A. — *Algebraic Theory of Automata*. Academic Press, New York, 1968.
4. Hopcroft J. E., Ullman J. D. — *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reeding Mass. 1979.
5. Livovschi L., Popovici C., Georgescu H., Tăndăreanu N. — *Bazele Informaticii*, Ed. Did. şi Ped., 1981.
6. Marcus S. — *Gramatici şi automate finite*. Ed. Academiei, Bucureşti, 1964.
7. Mateescu A. — *O generalizare a teoremei Myhill-Nerode*, Lucrările Sesiunii Ştiinţifice a CCUB, Bucureşti, 20–21 Februarie 1987.
8. Păun G. — *Probleme actuale în teoria limbajelor formale*. Ed. St. şi Enc., Bucureşti, 1984.
9. Rudeanu S. — *Curs de bazele informaticii*, Fac. de Matematică, Univ. din Buc., 1979–1980 (nepublicat).

Received 15.XI.1987

University of Bucharest
Faculty of Mathematics
Str. Academiei 14
70109 Bucureşti
România

ANALELE ŞTIINŢIFICE ALE UNIVERSITĂŢII „AL. I. CUZA" DIN IAŞI
Tomul XXXIII, s. I a. Matematică, 1987, f.asc. 4

AN ALGEBRAIC APPROACH TO CONCURRENCY BY MUTUAL EXCLUSION

BY

GABRIEL CIOBANU

The paper represents a first attempt to introduce an algebraic formalism for expressing concurrent computation by means of the mechanism of mutual exclusion. It proposes the structure of lattice-ordered semigroups as a domain of mathematical (denotational) semantics of programming languages with concurrent computation facilities.

The basic notions are introduced; then the usefulness of algebraic lattices in denotational semantics of programming languages for the representation of nondeterminism is proved. A binary concurrent operation is then defined on an algebraic lattice and we assume that the structure of a lattice-ordered semigroup is obtained. The introduction of such an operation is justified by the established properties and rules. The paper is concerned with these properties; it emphasizes the expressive power of lattice-ordered semigroups in the approach to concurrent properties. The results prove that the proposed structure can be useful, offering a model for actual practice and is leading to a good understanding of the complex instances that are harder to master by people directly involved in experimental work. The results prove also that the monitors constitute the most elaborate real mechanism of mutual exclusion.

1. Introduction. Denotational semantics of programming languages has been given strong theoretical foundations by the work of D. Scott. The basic idea was to represent the data types as partially ordered sets called domains, and computations on such data as continuous functions on domains. Initially, Scott considered the continuous lattices as domains, but afterwards it was proved that the algebraic lattices could be considered as well [11]. The domains must be closed under sum $+$, product $\times$, the function space construct $\rightarrow$ etc. to be able to describe the usual programming constructs and it also has to allow for solutions of domain equations $D \simeq E(D)$, where E is an expression built up from $+$, $\times$, $\rightarrow$ etc.

Let $(D, \leqslant)$ be a partially ordered set (poset). $D \neq \emptyset$, $a \in D$ is a lower (upper) bound of $X \subseteq D$ and write $a \leqslant X (X \leqslant a)$ if $a \leqslant x (x \leqslant a)$ for every $x \in X$. Let $\underline{X}$ be the set of the lower bounds of X and X the set of the upper bounds of X. For any $X \subseteq D$, the least upper bound of X, when it exists, is denoted by $\bigvee X$ or sup X, and the greatest lower bound of X is denoted by $\bigwedge X$ or inf X(if exists). Particularly, $0 = \sup \emptyset$ is the least element of D and $I = \inf \emptyset$ is the greatest element of D, if they exist. If we have a partially ordered set $(D, \leqslant)$, it is called a semilattice if $\sup\{a, b\}$ always exists, for every $a, b \in D$. In a semilattice, zero is an element 0 satisfying $0 \vee a = a$ for every a. An ideal of a semilattice D is a nonvoid subset I of D such that, for all $a, b \in D$, $a \vee b \in I$ if and only if $a, b \in I$. Let H be any nonvoid subset of D and let $(H]$ denote the smallest ideal containing H. If $H = \{a\}$ we shall write $(a]$ and $(a]$ is called the principal ideal generated by a. It is clear that $(a] = \{x / x \leqslant a\}$. This implies that if $(a] = (b]$ then $a = b$. A poset $(D, \leqslant)$ is a lattice if $\sup\{a, b\}$ and $\inf\{a, b\}$ exist for all $a, b \in D$. A lattice $(D, \leqslant)$ is called complete if $\sup H$ and $\inf H$ exist for any subset $H \subseteq D$. The element $a \in D$ is called compact if $a \leqslant \sup X$ for each $X \subseteq D$, then there exists $x \in X$ such that $a \leqslant x$ — i.e. if a is contained in an infinite join, it is already contained in a finite join. The set of all compact elements of D is denoted by $K(D)$, and the set of all ideals in D is denoted by $I(D)$. A domain is an algebraic lattice $(D, \leqslant)$ i.e. the complete lattice $(D, \leqslant)$ for which every element a can be written as $a = \sup\{x \in K(D) / x \leqslant a\}$.

Proposition 1. *Let $(D, \leqslant)$ be a semilattice with zero. Then $(I(D), \subseteq)$ is a domain and the mapping $d \to (d]$ preserves order and finite join. Moreover, the principal ideals are precisely the compact elements of $(I(D), \subseteq)$.*

A proof can be found in [6].

Proposition 2. *Let $(D, \leqslant)$ be a domain. Then there exists a semilattice $(D_0, \leqslant)$ with zero such that D is isomorphic to $I(D_0)$.*

We indicate only the steps of the proof:

i) Let $D_0 = K(D)$. $(K(D), \leqslant)$ is a semilattice with zero.

ii) Let $I_a = \{x \in K(D) / x \leqslant a\}$. $I_a \in I(K(D))$.

iii) If $a \neq b$ then $I_a \neq I_b$; if $a < b$ then $I_a \subset I_b$.

iv) Let $I \in I(K(D))$; then there exists an $a \in D$ such that $I = I_a$ — in fact, $a = \vee\{y / y \in I\}$.

v) The correspondence $i : a \to I_a$ sets up an isomorphism between D and $I(K(D))$. A detailed proof can be found in [6].

Thanks to these results, every domain can be represented as a domain of ideals on some semilattice $(D, \leqslant)$ with zero, ordered by inclusion. The elements of D can be identified with the principal ideals on D. Intuitively, elements of $I(D)$ represent evolutions of some (perhaps infinite) computations, while elements of D are the partial results, obtainable in finite time, of such computations.

2. **Nondeterminism.** Scott's theory is quite satisfactory for description of sequential, deterministic computations, but is inadequate to express nondeterminism, parallelism and concurrency. The progress made in this direction is connected with the notion of powerdomain introduced by P l o t k i n [10] and S m y t h [12]. With the help of this notion can be described some nondeterministic features of programming languages. The present aim is to show that a power domain can be defined as an algebraic lattice and that this approach provides nice mathematical properties [8].

If a nondeterministic computation is considered now, what is obtainable in finite time is a finite subset of elements of D. Thus let $S(D)$ be the set of all nonempty finite subsets of D. The natural ordering on $S(D)$ is the E g l i-M i l n e r preordering $X \prec_{EM} Y = (\forall x \in X\, \exists y \in Y : x \leqslant y)$ & $(\forall y \in Y\, \exists x \in X : x \leqslant y)$. This preordering induces an ordering $\leqslant_{EM}$ on the set $P(D)$ of all equivalence classes defined by the relation $\equiv_{EM}$ on $S(D)$.

$X \equiv_{EM} Y \Leftrightarrow X \prec_{EM} Y$ & $Y \prec_{EM} X$. In order to define a powerdomain of an algebraic lattice in such a way that it is again an algebraic lattice, it is used the notion of ideal due to Frink [8]. Let $(D, \leqslant)$ be any poset, $D \neq \varnothing$; a set $I \subseteq D$ is an Frink ideal if for every finite $F \subseteq I$ and every $x \in D$, $x \leqslant F$ implies $x \in I$. The set $F(D)$ of all Frink ideals of D, ordered by inclusion, is an algebraic lattice.

Now let $(D, \leqslant)$ be an algebraic lattice; we consider its powerdomain $\mathcal{P}(D)$ the set of all Frink ideals of the poset $(P(K(D)), \leqslant_{EM})$, i.e. $\mathcal{P}(D) = F(P(K(D)))$. Obviously, $(\mathcal{P}(D), \subseteq)$ is again an algebraic lattice. Furthermore, $\mathcal{P}$ is a functor in the category of algebraic lattices and preserves surjectivity of morphisms and projective limits. As a result, the solutions to domain equations involving $\mathcal{P}$ and $+$, $\times$, $\to$ etc. exist [5, 8].

3. **Concurrency.** At present not enough is known about concurrent systems and processes, especially from a practical standpoint. This can be the main reason that there isn't as yet a firm formal approach of nonsequential processes. Therefore at this stage, one can merely speculate about the properties that a „meaningful" concurrent computation should have.

A concurrent program consists of a collection of processes and shared objects. Each process is defined by a sequantial program; the shared objects allow these programs to cooperate in accomplishing some tasks. The processes can be implemented by multiprogramming, by multiprocessing, or by some combination of these approaches. The shared objects can be implemented in shared memory or can be a computer — communications network. The data (variables) are accessible to the processes according to the rules of disjointness and mutual exclusion. The rule of disjointness is due to H o a r e [7] and states that each component process makes its contribution to the common task, without interfering with other component processes. However, the interesting problems require some form of interaction between the component processes. One of the reasons why parallel processes have to interact with each other is because they need to share some limited resource. We might view the action of each parallel process thus: for a part of the time it operates freely in parallel with all the other processes, but from time to time it enters a so-called critical region and has exclusive use of some resource. The effect of a critical region is to restrict interleavings of processes, thus preventing undesirable interleavings. This is the mutual exclusion mechanism. The critical region can be considered the unit of action and it is defined by the programmer. In order to be possible to make meaningful statements about the effect of concurrent computation, the mutual exclusion of operations on shared data (variables) must hold.

Because the denotational approach of program semantics is implementation-independent, and one of the objectives for a theory of parallel programming is conceptual simplicity, the details of scheduling resources among competing processes, the forms of synchronization and other implementation-related considerations are ignored. Thus the concurrency can be specified by $p_1 \| p_2 \| \ldots \| p_n$, where $p_1, \ldots, p_n$ are sequential processes. The meaning of such a specification is the restricted (by the mutual exclusion) interleaving of the atomic actions of the processes. Now, we can associate data of concurrent processes with operations (actions) defined on them. Therefore a process evolution can be overtaken by the adequate data values sequences.

If $(D, \leq)$ is considered the set of values vectors of processes data, where the enumeration within the vector may possibly be established by the memory location of the values representation, a process evolution can be found to be a principal ideal of the domain $(D, \leq)$. Thus, because we associate the data with the actions defined on them, a process evolution up to the moment when a (partial) result $a \in D$ appears, is the principal ideal generated by a. In this way, all the legal modalities of obtaining the (partial) result a are considered. The order relation on D is induced by the corresponding order relations on every involved data domain to be more or less defined. $(D, \leq)$ is an algebraic lattice; for every $a, b \in D$, $a \wedge b$ represents the common data values of the vectors a and b, and $a \vee b$ is the vectors of all data values which appear in a and b.

Let be $(D, \leq)$ and let us try to define a concurrent operator over it. The modularity of denotational semantics method requires that for every construction operator which forms a new element $a \| b$ from two smaller elements a, b there exists a semantic operator S which relates the meanings of a and b to that of $a \| b$. Denoting by $m(a)$ the meaning of a we need a commutation rule $m(a \| b) = S(m(a), m(b))$. In order to appoint a meaningful semantic operator, we must consider the processes with results a and b separately, since their actions are executed consecutively, while for $a \| b$ their actions are interleaved thereby creating new effects. A solution is to consider $m(d)$ the principal ideal $(d]$ which reflects the process evolutions, the execution sequences with the result d. Then $m(a \| b) = M(m(a), m(b))$, where M is a restricted merging operator. This means that the result $a \| b \in D$ is generated by the interleavings of the actions reflected by $(a]$ and $(b]$, interleavings restricted by mutual exclusion of actions on common data. In order to assure a fair merge between processes, the operator M introduced above could be considered discontinuous. We note that, in accordance with proposition 1, the semantic range is also an algebraic lattice.

Therefore, let be $\| : D \times D \to D$ and we postulate that $(D, \|)$ is a semigroup with the unit 0. Following the definition, $a \| b = b \| a$. Moreover, for every $a, b, c \in D$ we assume that $a \| (b \vee c) = (a \| b) \vee (a \| c)$ and $a \| (b \wedge c) = (a \| b) \wedge (a \| c)$. These assumptions could be inferred, especially for finite processes, but we prefer to postulate them. The adoption of these assumptions allows a considerable algebraic power which could not be obtained otherwise. Briefly, $(D, \leq, \|)$ is a lattice-ordered semigroup [1]. Using this structure, significant properties of the concurrent phenomenon could be pointed out.

Remark 1. The distributivity of $\|$ can be extended for finite sets. Thus, for every finite index set I

$$a \| (\bigvee_{i \in I} b_i) = \bigvee_{i \in I} (a \| b_i) \text{ and } a \| (\bigwedge_{i \in I} b_i) = \bigwedge_{i \in I} (a \| b_i).$$

A lattice-ordered semigroup $(D, \leq, \|)$ is complete if $(D, \leq)$ is a complete lattice and $a \| (\bigvee_{i \in I} b_i) = \bigvee_{i \in I} (a \| b_i)$ for every set I. From now on D will mean a commutative complete lattice-ordered semigroup.

Proposition 3. *If $a, b, c \in D$ and $a \leq b$, then $a \| c \leq b \| c$.*

The proof is obvious.

Remark 2. The strict relation $a < b$ implies only $a \| c \leq b \| c$.

Proposition 4. *Let be $a, b \in D$. Then $a \| b = (a \vee b) \| (a \wedge b)$.*

Proof. On the one hand, in accordance with proposition 3, $a \| b \leq a \| (a \vee b)$ and $a \| b \leq (a \vee b) \| b$, following $a \| b \leq a \| (a \vee b) \wedge b \| (a \vee b) = (a \wedge b) \| (a \vee b)$. On the other hand, $(a \wedge b) \| (a \vee b) = (a \wedge b) \| a \vee (a \wedge b) \| b$. Applying proposition 3 again, $(a \wedge b) \| a \leq b \| a = a \| b$ and $(a \wedge b) \| b \leq a \| b$, from which it results $(a \wedge b) \| (a \vee b) \leq a \| b$. This means that the concurrent operation between a and b is the same as the concurrent operation between $a \vee b$ and $a \wedge b$. But this emphasizes the common part of the data a and b, and implicitly the operation associated with them. This isn't anything but the basic Brinch Hansen idea to implement multiprogramming by monitors [1]. Brinch Hansen presents these essential properties needed to implement „structured multiprogramming":

— an association of shared data with operations defined on them;

— mutual exclusion of these operations in time;

— synchronizing primitives which permit the control of process scheduling.

Our approach meets these requirements and therefore the results show that the concurrent operation by mutual exclusion is equivalent to a concurrent operation using monitors.

Proposition 5. *Let be $a, b \in D$.*

i) *If $a \wedge b = 0$, then $a \| b = a \vee b$.* ii) $a \| b \geq a \vee b$.

The proof is obvious. This result shows that parallelism can be considered as a specific case of concurrency.

Corollary 1. *If $a \| b = a$, then $b \leq a$.*

Proposition 6. *Let be $a, b, c \in D$.*

i) *If $a, b \leq c$ and $a \wedge b = 0$, then $a \| b \leq c$.*

ii) *If $a \| b = a \vee b$ and $c \leq b$, then $a \| b = b \vee (a \| c)$.*

Proof. Obvious.

Proposition 7. *Let be $a \in D$. The set $s(a) = \{b \in D \mid a \| b = a\}$ is a lattice ideal and a subsemigroup of D; if $a \| a = a$ then $s(a) = (a]$.*

Proof. A lattice ideal can be characterized by: if $b, c \in s(a)$, then $b \vee c \in s(a)$ and if $b \in s(a)$ and $c \leq b$ then $c \in s(a)$. It is easy to show that $s(a)$ is a semilattice. Now let be $b \in s(a)$, $c \in D$ and $c \leq b$. By corollary 1, $b \leq a$ and thus $a = a \vee b = a \| b$. By proposition 6 ii), $a = a \| b = b \vee (a \| c)$ which shows that $a \| c \leq a$. But $a \| c \geq a$, which forces $a \| c = a$, hence $c \in s(a)$. Straightforward calculations show that $s(a)$ is a subsemigroup of D. Clearly, $s(a) \subseteq (a]$. Let be $x \in (a]$; then $a \| x = (a \vee x) \| (a \wedge x) = a \| (a \wedge x) = (a \| a) \wedge (a \| x) = a \wedge (a \| x) = a$ and thus $x \in s(a)$.

In what follows, the notation $a_1 \ldots \bar{a}_i \ldots a_n$ will mean $a_1 \| a_2 \| \ldots \| a_{i-1} \| a_{i+1} \| \ldots \| a_n$.

Proposition 8. *Let be $a_i \in D$, $1 \leq i \leq n$. Then*

i) $\|_{i=1}^{n} a_i = (\bigvee_{i=1}^{n} a_1 \ldots \bar{a}_i \ldots a_n) \| (\bigwedge_{i=1}^{n} a_i) = (\bigwedge_{i=1}^{n} a_1 \ldots \bar{a}_i \ldots a_n) \| (\bigvee_{i=1}^{n} a_i)$.

ii) *If $a_i \| a_j = a_i \vee a_j$ for every $1 \leq i, j \leq n$, then* $\|_{i=1}^{n} a_i = \bigvee_{i=1}^{n} a_i$.

Proof. i) We will prove the first equality by induction. For $n=2$ the result holds by proposition 4. Suppose it true for an arbitrary fixed $n\in N$. We have

$$\bigvee_{i=1}^{n+1} a_1\ldots\bar{a}_i\ldots a_{n+1} = a_{n+1}\parallel\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\vee\mathop{\Big\|}_{i=1}^{n} a_i =$$

$$= a_{n+1}\parallel\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\vee\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\parallel\Big(\bigwedge_{i=1}^{n} a_i\Big)=$$

$$=\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\parallel\Big(a_{n+1}\vee\bigwedge_{i=1}^{n} a_i\Big) \text{ and then } \Big(\bigvee_{i=1}^{n+1} a_1\ldots\bar{a}_i\ldots a_{n+1}\Big)\vee\Big(\bigwedge_{i=1}^{n+1} a_i\Big)=$$

$$=\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\parallel\Big(a_{n+1}\vee\bigwedge_{i=1}^{n} a_i\Big)\parallel\Big(a_{n+1}\wedge\bigwedge_{i=1}^{n} a_i\Big)=$$

$$=\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\parallel\Big(a_{n+1}\parallel\bigwedge_{i=1}^{n} a_i\Big)=$$

$$=\Big(\bigvee_{i=1}^{n} a_1\ldots\bar{a}_i\ldots a_n\Big)\parallel\Big(\bigwedge_{i=1}^{n} a_i\Big)\parallel a_{n+1}=\Big(\mathop{\Big\|}_{i=1}^{n} a_i\Big)\parallel a_{n+1}=\mathop{\Big\|}_{i=1}^{n+1} a_i.$$

The second equality can also be proved by induction.

ii) By induction.

Proposition 9. *For every $a, b_i\in D$, $1\leqslant i\leqslant n$, we have* $a\wedge\mathop{\Big\|}_{i=1}^{n} b_i = a\wedge\mathop{\Big\|}_{i=1}^{n}(a\wedge b_i)$.

Proof. $a\wedge\mathop{\Big\|}_{i=1}^{n}(a\wedge b_i)=a\wedge[a\parallel\ldots\parallel a\wedge a\parallel(\bigwedge_{i=1}^{n} b_i)\wedge\ldots\wedge a\parallel(\bigwedge_{i=1}^{n} b_1\ldots b_i\ldots b_n)\wedge\mathop{\Big\|}_{i=1}^{n} b_i]=a\wedge\mathop{\Big\|}_{i=1}^{n} b_i.$

Corollary 2. *If $a, b_i\in D$ and if $a\wedge b_i=0$ for every i, $1\leqslant i\leqslant n$, then* $a\wedge\mathop{\Big\|}_{i=1}^{n} b_i=0$.

Proposition 10. *Let be $a\in D$. The set $p(a)=\{b\in D\,|\,a\wedge b=0\}$ is a lattice ideal and a subsemigroup of D.*

Proof. By corollary 2, $p(a)$ is a subsemigroup of D. Let be $b, c\in p(a)$; then $a\wedge(b\vee c)\leqslant a\wedge(b\parallel c)=0$ by corollary 2 again, showing that $p(a)$ is a semilattice. On the other hand, if $b\in p(a)$, $c\in D$ and $c\leqslant b$ then $0\leqslant a\wedge c\leqslant a\wedge b=0$, hence $c\in p(a)$.

In order to investigate the evolution of concurrent computation, residuals of D could be defined. Let be $a, b\in D$; it is called residual of a by b and is written $a:b$ the greatest element x (if exists) which holds $b\parallel x\leqslant a$. A lattice-ordered semigroup is called residuated if there exist residuals for all pairs of its elements. Our lattice-ordered semigroup is not residuated, because if $b>a$, the residual $a:b$ doesn't exist. Indeed, if there is $c\geqslant 0$ so that $b\parallel c\leqslant a$, then $b\leqslant a$; thus, if $b>a$ then $a:b$ does not exist. However, the residual $a:b$ exists for all $a, b\in D$ with $b\leqslant a$, because $b\parallel 0\leqslant a$ and the lattice-ordered semigroup is complete. Moreover, $a:0=a$ for all $a\in D$.

Proposition 11. i) *If $a, b, c\in D$ and $a\leqslant b$, then $a:c\leqslant b:c$.*

ii) *If $a, b, c\in D$ and $b\leqslant c$, then $a:c\leqslant a:b$.*

iii) *If $a, b\in D$ and $a\parallel c=b\parallel c$ implies $a=b$ for every $c\in D$, then $a=(a\parallel b):b$.*

Proof. i) The proof follows directly from the definition.

ii) Because $c\parallel(a:c)\leqslant a$ and $b\leqslant c$, then $b\parallel(a:c)\leqslant c\parallel(a:c)\leqslant a$, therefore $a:c\leqslant a:b$.

iii) We have $a\leqslant(a\parallel b):b$ for every $a, b\in D$. By hypothesis and proposition 3, $x\parallel b\leqslant y\parallel b$ implies $x\leqslant y$. By definition, $[(a\parallel b):b]\parallel b\leqslant a\parallel b$. Then $(a\parallel b):b\leqslant a$ and thus $a=(a\parallel b):b$.

If $a, b\in D$, then $b\parallel(a:b)\leqslant a$. However, if there exists $c\in D$ so that $b\parallel c=a$, then $b\parallel(a:b)=a$.

Proposition 12. *Let be $a, b, c\in D$, $a\geqslant c$. Then $(a:c):b$ exists if and only if $a:(c\parallel b)$ exists; if they exist, then $(a:c):b=a:(c\parallel b)=a:(b\parallel c)=(a:b):c$.*

Proof. Let be $X=\{x\in D/b\parallel x\leqslant a:c\}$ and $Y=\{y\in D/(c\parallel b)\parallel y\leqslant a\}$. $b\parallel x\leqslant a:c$ implies $c\parallel(b\parallel x)=(c\parallel b)\parallel x\leqslant a$, hence $X\subseteq Y$. But $(c\parallel b)\parallel y\leqslant a$ implies $b\parallel y\leqslant a:c$. Thus $X=Y$.

Proposition 13. i) *Let be $a, b, c\in D$, $a, b\geqslant c$. Then $(a\wedge b):c=(a:c)\wedge(b:c)$.*

ii) *Let be $a, b, c\in D$, $a\geqslant b, c$. Then $a:(b\vee c)=(a:b)\wedge(a:c)$.*

A more general result can be found in [4].

Other results can also be obtained, which are more or less significant for the concurrent computation [3, 4]. But the fact that this formalism allows for expressing easily the properties of concurrency is important. All these results increase the interest in the demonstration of the lattice-ordered semigroup as a domain of the semantics of programming languages with concurrent features. In order to support this alternative, it remains to prove that the proposed structure allows for solutions of domain equations involving $\parallel$ and $\mathcal{P}$ and $+$, $\times$, $\to$ etc.

REFERENCES

1. Brinch Hansen P. — *Structured multiprogramming*. Comm. ACM 15 (1972), 574—578.
2. Brinch Hansen P. — *The programming language Concurrent Pascal*. IEEE Trans. Software Engrg. 1 (1975), 199—207.
3. Ciobanu G., Deaconescu M. — *On certain lattice-ordered semigroups*. Semigroup Forum 31 (1985), 367—371.
4. Dubreil-Jacotin M. L., Lesieur L., Croisot R. — *Leçons sur la théorie des treillis, des structures algébriques ordonnées et des treillis géométriques*, Gauthier-Villars, Paris, 1953.
5. Gierz G., Hofmann K. H., Kiemel K., Lawson J. D., Mislove M., Scott D. S. — *A compendium of continuous lattices*, Springer-Verlag, 1980.
6. Grätzer G. — *Universal algebra*, 2[nd] Ed., Springer-Verlag, 1979.
7. Hoare C.A.R. — *Towards a theory of parallel programming*, in „Operating systems techniques", C.A.R. Hoare, R.N. Perrott (eds.), Academic Press, New York, 1972, 61—71.
8. Hrbacek K. — *Powerdomains as algebraic lattices*, in „ICALP 85", W. Brauer (ed.) Lect. Notes in Comp. Sci. 194, Springer-Verlag, 1985, 281—289.

9. Milne G., Milner R. — *Concurrent processes and their syntax*, J.ACM 26 (1979), 302–321.
10. Plotkin G. — *A powomain construction*, SIAM J. Comp 5 (1976), 452–486.
11. Scott D. S. — *Data types as lattices*, SIAM J. Comp 5(1976), 522–587.
12. Smyth M. B. — *Powerdomains*, J. Comp. System Sci. 16(1978), 23–36.
13. Stoy J. E. — *Denotational semantics: the Scott-Strachey approach to programming language theory*, MIT Press, 1977.

Received 15.XI.1987

University „Al. I. Cuza"
Computer Center
6600 Iași—Romania

ANALELE ȘTIINȚIFICE ALE UNIVERSITĂȚII „AL. I. CUZA" DIN IAȘI
Tomul XXXIII, s. I a. Matematică, 1987, f. 4

CONTEXTUAL SYSTEMS AND CONTEXTUAL TRANSDUCERS

BY

DOREL LUCANU and DAN CRISTEA

1. Introduction. The ideas presented in this paper evolved from practical problems related to natural language comprehention in well bounded universes of discourse.

Rather frequently, in natural language sentences appear groups of words whose meanings are different than those done by the sums of their components. This is the case with idiomatic expressions or technical jargon. Such expressions must be considered without trying to breake their meanings into elementary building bricks. Let us refer to these linguistical phenomena as *sintagmae*. In fact what is to be done, before any other process of interpretation, is to translate them directly to symbols that, conventially or not, incorporate their whole meanings. Moreover there is often the case that a sintagma has more than a single meaning, even in a perfectly bounded universe of discourse. Nevertheless a human listener can easily decide between a sense or another. He uses the context to desambiguate among more interpretations. What precisely is meant by „context" is certainly hard to establish. At a first glance it is something like a neighbourhood around the ambiguous sintagma. How large this neighbourhood must be considered is again an open question and we prefer to pay no atention to this question here. A more attentive insight reveals that not everything in this neighbourhood is important in establishing the correct meaning and that usually there is there included a string of words that only matters. This remark modifies a little our definition for sintagma. When we insist on meaning rather then sintax, a sintagma could be seen as a pair of strings of words which, appearing in text at a certain distance one of the other, precisely determine a sense to one of them. Every one of the two strings of words will be denoted by *context*.

The paper proposes two mathematical tools for sintagmae transformation. The first of them, called *n-contextual systems*, is a generalization of rewriting systems on strings [2]. The second, called *n-contextual transducers*, is a combination between the first and the generalized sequential machine [3, 4]. The formal definitions of the two mechanisms and some properties of them are presented.

2. Preliminaries. In the rest of the paper V will denote a numerable set of symbols. The free monoid generated by V will be denoted with V^* and the empty string of V^* with e. A *sintagma* is a tuple

$$<c_1, \ldots, c_n>,$$