Provided for non-commercial research and educational use.
Not for reproduction, distribution or commercial use.

Serdica
Bulgariacae mathematicae publicationes

Сердика
Българско математическо списание

The attached copy is furnished for non-commercial research and education use only.
Authors are permitted to post this version of the article to their personal websites or institutional repositories and to share with other researchers in the form of electronic reprints.
Other uses, including reproduction and distribution, or selling or licensing copies, or posting to third party websites are prohibited.

For further information on
Serdica Bulgaricae Mathematicae Publicationes
and its new series Serdica Mathematical Journal
visit the website of the journal http://www.math.bas.bg/~serdica
or contact: Editorial Office
Serdica Mathematical Journal
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Telephone: (+359-2)9792818, FAX:(+359-2)971-36-49
e-mail: serdica@math.bas.bg

DISTRIBUTED SYSTEMS AS CONTINUOUS FUNCTIONS

ILIA DIMITROV

An approach to mathematically characterizing distributed systems (DS) is developed in this paper. It allows to treat a DS as a continuous function from input state histories to system state histories, depending on explicitly provided synchronization.

The underlying model of DS includes a set of operators, asynchronously and concurrently acting on shared memory objects. The important to the operation of the DS synchronization, casuality precedence, etc. relations are abstracted by two functions, called the "read" and "write" synchronizers. They are a natural device to handle the behavioural non-determinism at the level of synchronization.

It is shown that the approach includes as a special case analogous results about message-passing (communicating) systems.

Introduction. Recently, much attention has been given to mathematically characterizing distributed systems (DS) as continuous functions in the style of Kahn [5]. In this paper his approach is generalized for a much larger class of DS. Kahn's results concern message-passing systems with determinate operators, acting as continuous functions from and to mutually independent sequence domains (histories of communication lines). Keller [6] tried to introduce in this scheme non-determinism but without great success ([1], also the discussion on [6] in [2]). Brock and Ackerman develop Keller's idea that non-determinism of the "merge" type ([6] and the discussion on it in [2]) has something to do with synchronization. They introduce in [1] scenarios to represent casuality constraints and overcome "the shortcoming of history relations" used by Keller. But for some reason they contradict themselves, stating that "due to the time-independent nature of data flow computation, there are no casuality relations between items of different input ports, between items of different output ports, and from output items to input items".

The last part is not necessarily true and they show it themselves in their own example that two networks with the same history relation are not substitutable as components of a larger network. The essence of the example is exactly that synchronization problems appear when individual items of the input to a possibly compound operator depend circularly on individual items of its output. And, once allowing dependence between items of output and items of input, all kinds of casuality relations between input items and output items can appear.

Such systems in which there are dependences between individul items of different histories are not Kahn's systems, i. e. his fixed-point theory is not applicable to them. To generalize his continuity results, synchronization functions are introduced in Section 1 of this paper, reflecting the dependences between individual items of input and output.

The most significant distinction between the model of DS, used in this paper, and Kahn's model is that only communications via chanels (lines) are allowed in the latter model, while in the former one shared memory is used for interoperator communications. Certainly, there is a boom in studying message-passing systems (communicating systems [4], data flow systems, application systems [7], actor systems [3], etc.), but nevertheless it does not seem reasonable to squeeze any system into such a model, all the more that the opposite move is easier (i. e. to model channels or lines by shared memory with suitable synchronization). Therefore, message passing systems can be considered as a special case of DS with distributed memory.

1. A model of DS. Let's consider as an example of DS a man-computer space rocket control system R, consisting of:

— an astronaut A;

— a computerized onboard control system C;

— a data-gathering system G which keeps record of the rocket's condition and its position in space with the help of sensors and which feeds this information I to a data base B;

— a TV-screen V on which the astronaut A can observe the environment.

The astronaut has access to the data base through a terminal. The conrol acts of A and C are sent to the engines for execution and are registered in B (acting as a shared memory). The way R operates fairly obvious.

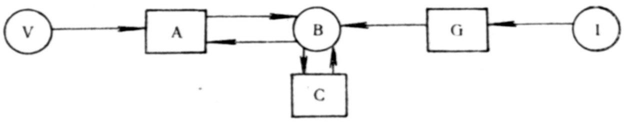

Fig. 1. The control system R

More generally, a distributed system (DS) consists of a finite set $O=\{o_1, o_2, \ldots, o_n\}$ of asynchronously and concurrently acting determinate operators; a set $I=\{i_1, i_2, \ldots\}$ of input objects with memory, the state of which is changed only from outside the DS and can be accessed by the operators from O only for "reading"; a set $S=\{s_1, s_2, \ldots\}$ of system objects with memory, the state of which is updated by the operators from O and can be accessed from outside the DS for "reading" (implicit output of the distributed system).

Each operator $o\in O$ has a set of input objects from $I\cup S$ (denoted by domain n^o) and a set of output objects from S (denoted by range_o). The action of each $o\in O$ consists in obtaining information from the current states of its input objects and updating with the computed results the current states of its output objects. Concerning the rest of the system, the acts of reading, computing, and writing, performed by an operator on its invocation, represent an indivisible elementary act — firing of the operator.

Going back to the control system R (Fig. 1.), the respective domains and ranges of the operators are:

$$\begin{array}{lll} \text{domain}_A=\{V, B\}, & \text{domain}_C=\{B\}, & \text{domain}_G=\{I\} \\ \text{range}_A=\{B\} & \text{range}_C=\{B\} & \text{range}_G=\{B\}. \end{array}$$

The consecutive states $\bar{m}_i$, $i=1, \ldots, t_m$ of any object $m\in I\cup S$ form its local discrete history $h_m=\langle \bar{m}_1, \ldots, \bar{m}_{tm}\rangle$. The length of this local history

$\underline{l h_m} = t_m$ determines m' s current local moment. An object may have empty local history $\langle\rangle$ ($\underline{l h_m} = 0$) or infinite local history (its total history) $\langle \overline{m}_1, \overline{m}_2, \ldots \rangle$ ($\underline{l h_m} = \infty$). The history h of the whole system is a map from the set of objects to their local histories: $h(m) = h_m$. Also, each operator acts in its local discrete time, defined by the sequence of elementary transitions, performed by that operator.

The interaction of operators, resulting from their asynchronous and parallel activities, manifests itself in the order in which they access their respective shared input and output objects for reading and writing. This synchronization of real-life activities will be abstracted (modelled) by two partial functions — the "read" synchronizer r and the "write" synchronizer w:

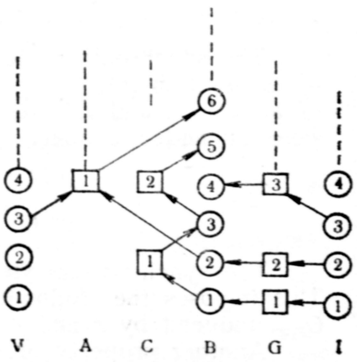

Fig. 2. An initial segment of the control system R history

(D1) $$r(o, t, m) = t', \quad o \in O, \quad t \in N, \quad m \in \text{domain}_o, \quad t' \in N$$

(D2) $$w(o, t, m) = t', \quad o \in O, \quad t \in N, \quad m \in \text{range}_o, \quad t' \in N,$$

where $N = \{1, 2, \ldots\}$.

D1 associates with each operator $o \in O$ the accessible for reading states of each of its input objects (state number t' in history $h(m)$; $h(m)[t']$) in the t-th local for o moment (on its t-th invocation), $t = 1, 2, 3, \ldots$ Similarly D2 associates with each operator the accessible for writing states of its output objects for each invocation.

D1 and D2 will be illustrated by an initial segment of a possible history of the control system R (Fig. 2).

Fig. 2 models the following particular course of events: The sensor system G is turned on and it initializes the data base $B(h(B)[1])$. The control system C starts computing the control, using the initial state of B. In the meantime G updates B with some new information ($h(B)$ [2]). The astronaut detects some deviation on the tv-screen $V(h(V)[3])$ and, using the current state of B, decides to perform a control act. While he is doing all that, the faster computer C issues two control commands (registered in $h(B)$ [3] and $h(B)$ [5]), and G updates once more $B(h(B)$ [4]) . . . This goes on and on until the space rocket lands safely back on the Earth. Of course, there are systems intended to run forever (or for indefinitely long time), hence their histories are considered to have infinite lengths.

Having in mind the definitions of r and w (D1 and D2), the described behaviour of R is formalized by transcribing the arrows on Fig. 2 into the following partial functions ("read" and "write" synchronizers):

$$.r(A, 1, V) = 3, \; w(A, 1, B) = 6,$$

$$.r(A, 1, B) = 2,$$

$$.r(C, 1, B) = 1, \; w(C, 1, B) = 3,$$

$$r(C,\ 2,\ B)=3,\ w(C,\ 2,\ B)=5,$$

$$r(G,\ 1,\ I)=1,\ w(G,\ 1,\ B)=1,$$
$$r(G,\ 2,\ I)=2,\ w(G,\ 2,\ B)=2,$$
$$r(G,\ 3,\ I)=3,\ w(G,\ 3,\ B)=4.$$

The synchronizers define a directed graph $G_{r,w}$ in the global history space of the system $(O \cup I \cup S)\times N$ with nodes $(e,\ t)$ (where e is either an operator or an object, and t is the local for e moment) and edges $((e,\ t),\ (e',\ t'))$ between all nodes connected by the synchronizers:

$$e \in I \cup S,\ e' \in O \quad \text{and} \quad r(e',\ t',\ e)=t \quad \text{or}$$
$$e \in O,\ e' \in I \cup S \quad \text{and} \quad w(e,\ t,\ e')=t'.$$

The synchronizers should reflect the physical nature of synchronization. This imposes the following restriction on r and w: (P1) the directed graph $G_{r,w}$, induced by r and w (see Fig. 2), should be acyclic.

Another property of w must hold true to exclude non-deterministic states from the object histories — simultaneous updates of an object are not possible

(P2) $$\forall o_1,\ o_2 \in O,\quad \forall t_1,\ t_2 \in N,\quad \forall m \in \text{range}_{o_1} \cap \text{range}_{o_2},$$
$$(w(o_1,\ t_1,\ m)=w(o_2,\ t_2,\ m) \Rightarrow o_1=o_2\ \&\ t_1=t_2).$$

In this setting each operator $o \in O$ is considered ready to fire (enabled when the current (in respect to o) states of all its input objects from domain^o are already produced and when all its output objects from range_o have reached their current (in respect to o) states (are ready to be updated). This will be formalized by the auxiliary function φ:

(D3) $$\varphi(o,\ h)=\begin{cases} t & \text{if } \forall m \in \text{domain}_o\ r(o,\ t,\ m)\leq \underline{l}\,h(m)\ \& \\ & \quad \forall m \in \text{range}_o\ w(o,\ t,\ m)=\underline{l}\,h(m)+1\,; \\ 0 & \text{if there is no such } t. \end{cases}$$

Note that $\underline{l}\,\langle\,\rangle=0$ and that P1 guarantees that $\varphi(o,\ h)$ is always defined and is unique.

$\varphi(o,\ h)$ has as a value the current for o local moment if o is enabled with h as a current history. Let's see now what happens with the control system R if its current history is:

$$h(V)=\langle \bar{V}_1,\ \bar{V}_2\rangle,\quad h(I)=\langle I_1,\ I_2,\ I_3\rangle,\quad h(B)=\langle \bar{B}_1,\ \bar{B}_2\rangle.$$

Then with the synchronization on Fig. 2:

$$\varphi(A,\ h)=0,\quad \varphi(G,\ h)=0,\quad \varphi(C,\ h)=1$$
$$(r(C,\ 1,\ B)=1<\underline{l}\,h(B)=2\ \&\ w(C,\ 1,\ B)=\underline{l}\,h(B)+1=3),$$

which means that only C is enabled and it will fire for the first time. When C fires the history of B will be extended with a new state $\bar{B}_3$ so G will become enabled to fire for the third time.

2. The Function of a DS.

Let's define more formally the value domains used by now in the style of [8]:

—$IH=I \underset{m}{\rightarrow} U^*$ — the domain of histories of input objects where U^* denotes the domain of tuples with lengths ranging from o to ∞ and elements from the domain U;

— $SH=S \underset{m}{\rightarrow} U^*$ — the domain of histories of system objects;

— $H=IH \underline{\cup} SH$ — the combined domain of histories;

— U — the domain of object states (left further undefined);

—$R=(O\text{—}N\text{—}I/S)\, m\, N'$ — the domain of read synchronizers;

— $W=(O\text{—}N\text{—}S) \underset{m}{\rightarrow} N'$ — the domain of write synchronizers; where I is the set of input objects, S — the set of system objects and O is the set of operators.

It is already possible to define the domain of transitions of system states:

$$T=(IHR\tau W) \rightarrow SH \rightarrow SH.$$

If $\tau \in T$ then τ maps an input object history ih, supplied with read (r) and write (w) synchronizers, into a function $F=\tau(ih, r, w)$ which performs the state transition of system objects by extending their histories with the new states, resulting from the firing of the enabled operators.

If the current combined history is $h=ih \cup sh$ and the operator o is enabled, let's denote by $\psi(h, o, m) \in U$ the state of the object $m \in \text{range}_o$, resulting from firing o. To produce $\psi(h, o, m)$ for all output objects from range_o, o uses the states of its input objects from domain_o, defined by the read synchronizer: $h(m)[r(o, t, m)]$.

The full definition of F is:

(D4) $$F(sh)=[m \rightarrow sh(m) \frown \bar{m} \mid m \in S], \quad \text{where}$$

$$\bar{m}=\begin{cases} \langle\psi(h, o, m)\rangle & \text{if } \exists o \in O(m \in \text{range}_o \ \&\ \varphi(o, h)>0), \\ \langle\,\rangle & \text{otherwise.} \end{cases}$$

The notation in D4 is from [8]. $[d \rightarrow v(d) \mid P(d)]$ is a map which maps each object d, such that $P(d)$ is true, into $v(d)$; "$\frown$" denotes tuple concatenation. Note that $u^* \frown \langle\,\rangle=u^*$ for any tuple $u^* \in U^*$.

$F(sh)$ maps any object $m \in S$ into its current history $sh(m)$ eventually extended with a new state $\psi(h, o, m)$ if m is an output object of some enabled operator o. Note that F is well defined due to the properties P1 and P2 of the synchronizers.

This, together with the determinism of the operators, makes F determinate, too.

Now it is necessary to introduce appropriate partial orders in the domains $U^*(\prec)$ and $H(\subseteq)$:

(D5) $$\text{for any } q_1, q_2 \in U^*, \ q_1 \prec q_2 \quad \text{iff} \quad q_1=\langle\,\rangle \quad \text{or}$$

$$\underline{l}\, q_1 \leq \underline{l}\, q_2 \ \&\ q_1[i]=q_2[i], \quad i=1, \ldots, \underline{l}\, q_1;$$

(D6) $$\text{for any } h_1, h_2 \in H, h_1 \subseteq h_2 \text{ iff } h_1 = [\,] \text{ or}$$
$$\underline{\text{dom}}\, h_1 \subset \underline{\text{dom}}\, h_2 \,\&\, \forall m \in \underline{\text{dom}}\, h_1\; h_1(m) \prec h_2(m).$$

D5 orders two tuples if the first one is a prefix of the second one and D6 reduces $\subseteq$ to $\prec$. The zero-length tuple $\langle\,\rangle$ is the smallest with respect to $\prec$ element of U^* and the null history (the empty map) $[\,]$ is the smallest with respect to $\subseteq$ element of H. The restriction of $\subseteq$ on IH and SH is obvious and will be denoted also by $\subseteq$.

To avoid some notational difficulties, let's have $h(m) = \langle\,\rangle$ if $m \notin \underline{\text{dom}}\, h$ (unlike [8] where $h(m)$ is considered undefined in this case).

From the definition of U^* and from D5 follows that any chain $q_1 \prec q_2 \prec \ldots$ in U^* has a lowest upper bound (lub) denoted by $\cup_{i=1}^{\infty} q_i$ or $\cup \{q_i \mid i \in N\}$. So $\prec$ is a complete partial order (cpo) and, having in mind D6, $\subseteq$ is a cpo, too.

If X and Y are any domains with cpo's $\prec_X$ and $\subseteq_Y$, then a function $f: X \rightarrow Y$ will be called monotonic if

(D7) $$\forall x_1, x_2 \in X (x_1 \prec_x x_2 \Rightarrow f(x_1) \subseteq_Y f(x_2))$$

and will be called continuous if it is monotonic and for each chain $x_1 \prec x \prec \ldots$ in X

(D8) $$f(\bigcup_{i=1}^{\infty} x_i) = \bigcup_{i=1}^{\infty} f(x_i).$$

Note that the lub $\cup_{i=1}^{\infty} f(x_i)$ in D8 always exists because f is monotonic and $\subseteq_Y$ is a cpo.

In the approach of Kahn [5] and its variant, presented by Keller in [6], the crucial point is that the transition function should be continuous (or at least monotonic) so that it would have a minimal fixed point. The unrestricted function F, as defined by Keller in [6], page 342, is not even monotonic, so he somewhat loosely restricts F only on selfproduced histories, starting from input histories and empty system histories (see also Kahn's equations Σ_p in section 3 of [5]).

Instead of restricting the defined in this paper F to the chain $[\,] \subseteq F([\,]) \subseteq F^2([\,]) \subseteq \ldots$ where it is continuous and has a minimal fixed point $F^*([\,]) = \cup_{i=0}^{\infty} F^i([\,])$, let's take a more operational view, giving the same results, and define directly the overall function f of a DS:

(D9) $$f(ih, r, w) = \bigcup_{i=0}^{\infty} F^i([\,]) = F^*([\,]).$$

The type of f is $(IH\ R\ W) \rightarrow SH$.

f is well defined because

(P3) $$F^i([\,]) \subseteq F^{i+1}([\,]), \quad i = 0, 1, \ldots$$

which follows from the definition D4 of F ($\subseteq$ is a cpo in SH).

The operational interpretation of D9 is that the result, associated to a particular input history and particular synchronizers is the history of states, produced by the operation of the DS (modelled by the successive application of F) starting from null system history.

Note that the transition function F (D4) registers the effects of firing all enabled operators (fires an operator the moment it becomes enabled). Any other transition function $\breve{F}$, delaying the firing of some operators, will be consistent with F and will be weaker than it in the following sense:

(P4) $$\breve{F}^n(sh) \subseteq F^n(sh), \quad n=0,\ 1, \ldots$$

If F is replaced in D9 with such a weaker function $\breve{F}$ which is fair (i. e. guarantees the firing of an enabled operator after a finite number of selfapplications) then

$$f(ih,\ r,\ w) = F^*([\ ]) = \breve{F}^*([\ ]).$$

3. Continuity of f. Theorem. *$f(ih, r, w)$ is continuous in its first argument ih.*

Proof. The following notation will be used in the proof:

— F_i is the defined by D4 function for input history ih_i;

— $sh_i^k = F_i^k([\ ])$, $k=0,\ 1, \ldots$ (hence $f(ih_i\ r,\ w) = F_i^*([\ ]) = \cup_{k=0}^{\infty} sh_i^k$);

— $h_i^k = ih_i \cup sh_i^k$ (the combined history of input and system objects).

First the monotonicity of f will be proved. Let $ih_1,\ ih_2 \in IH,\ ih_1 \subseteq ih_2$. The proof will be carried out by induction, showing

(P5) $$h_i^k \subseteq h_2^k, \quad k=0,\ 1, \ldots \quad \text{and}$$

(P6) $$z(k), \quad k=0,\ 1, \ldots,$$

where $Z(k)$ stands for the predicate $\forall s \in S(\underline{l}\, sh_1^k(s) < \underline{l}\, sh_2^k(s) => f(ih_1,\ r,\ w) = sh_1^k(s))$. P6 means that if at any moment the history of an object s is extended by F_2 for the greater input ih_2 and not extended by F_1 for the lesser input ih_1 then s has already reached its full (finite length) history for ih_1.

Obviously $sh_1^0 \subseteq sh_2^0$ and $z(o)$ are true.

Let's assume that $sh_1^i \subseteq sh_2^i$ and $z(i)$ are also true, $i=0. \ldots,\ n$.

Let $s \in S$.

If $\underline{l}\, sh_1^n(s) < \underline{l}\, sh_2^n(s)$ then from $z(n)$ directly follows that $sh_1^{n+1}(s) \prec sh_2^{n+1}(s)$ and that $f(ih_1,\ r,\ w)(s) = sh_1^{n+1}(s) = sh_1^n(s)$.

Let now $\underline{l}\, sh_1^n(s) = \underline{l}\, sh_2^n(s)$ and $w(o,\ t,\ s) = \underline{l}\, sh_1^n(s)+1$ for some o and some t.

If o is enabled to fire for $ih_1(\varphi(h_1^n,\ o)=t)$, then o will fire for ih_2, too (from the properties of the synchronizers and from the inductive assumption: $\varphi(h_2^n,\ o)=t)$. Hence $sh_1^{n+1}(s) = sh_2^{n+1}(s)$.

Let now $\varphi(h_1^n,\ o)=t-1$, i. e. o still cannot fire for the combined history of input and system objects h_1^n so $sh_1^{n+1}(s) \prec sh_2^{n+1}(s)$ in all cases. The interesting case, however, is when o can fire for h_2^n, i. e. $\varphi(h_2^n,\ o)=t$. This means that either the history $ih_1(m)$ of an input object m is too short for o to fire, or the history $sh_1^n(m)$ of a system object is too short. If $m \in I$ then its history is fixed so o will wait forever on the input ih_1. If, on the other hand, $m \in S$ then $z(n)$ prevents m from having its history extended ($\varphi(h_1^n,\ o) < \varphi(h_2^n,\ o)$ implies $\underline{l}\, sh_1^n(m) < \underline{l}\, sh_2^n(m)$), so o will never fire again relying only on ih_1. Hence $f(ih_1, \overline{r}, w)(s) = \overline{sh}_1^n(s)$.

$$sh_1^{n+1}(s) \prec sh_2^{n+1}(s) \quad \text{and} \quad (\underline{l}\, sh_1^{n+1}(s) < \underline{l} sh_2^{n+1}(s) \Rightarrow$$

$f(ih_1, r, w)(s) = sh_1^{n+1}(s))$ being true for each $s \in S$ means that $sh_1^{n+1} \subseteq sh_2^{n+1}$ and $Z(n+1)$ are true. Hence P5 and P6 are true.

From P5 it follows that

$$f(ih_1, r, w) = \bigcup_{i=0}^{\infty} sh_1^i \subseteq \bigcup_{i=0}^{\infty} sh_2^i = f(ih_2, r, w),$$

i. e. f is a monotonic function.

Let now $ih_0 \subseteq ih_1 \subseteq \ldots$ be a chain in IH.

Let $ih = \cup_{i=0}^{\infty} ih_i$ and $sh^k = \cup_{i=0}^{\infty} sh_i^k$.

$\forall k \supset j_k sh^k = sh_{jk}^k$ will be proved by induction.

$sh^c = sh_0^0 (= [\ ])$.

Let's assume that $\forall i \leq n \exists_{ji} sh^i = sh_{ji}^i$.

$sh^{n+1} = F(sh^n)$ by definition. When F is applied to sh^n, only a finite portion ih^+ of the total input ih is used (there is only a finite number of operators). Having in mind $ih = \cup_{i=0}^{\infty} ih_i$, this means that there is $ih_{j_{n+1}}$ such that $ih_{jn} \subseteq ih_{j_{n+1}}$ and $ih^+ \subseteq ih_{j_{n+1}}$. Hence, if F^+ denotes the defined by D4 function for the input ih^+,

$$sh^{n+1} = F(sh^n) = F^+(sh^n) = (F(sh^n) = F^+(sh^n) \subseteq F_{j_{n+1}}(sh^n) \subseteq F(sh^n))$$

$$F_{j_{n+1}}(sh^n) = \qquad \text{(the inducti ve assumption)}$$

$$F_{j_{n+1}}(sh_{jn}^n) = (ih_{jn} \subseteq ih_{j_{n+1}} \subseteq ih, \qquad sh^n = sh_{jn}^n \subseteq sh_{j_{n+1}}^n \subseteq sh^n)$$

$$F_{j_{n+1}}(sh_{j_{n+1}}^n) = sh_{j_{n+1}}^{n+1}.$$

So $sh^k = sh_{jk}^k \subseteq \cup_{n=0}^{\infty} sh_{jk}^n = f(ih_{jk}, r, w) \subseteq \cup_{i=0}^{\infty} f(ih_i, r, w)$, which leads, taking the lub $\cup_{k=0}^{\infty} sh^k = f(ih, r, w)$, to

$$f(\bigcup_{i=0}^{\infty} ih_i, r, w) = f(ih, r, w) \subseteq \bigcup_{i=0}^{\infty} f(ih_i, r, w).$$

From the monotonicity of f directly follows

$$f(\bigcup_{i=0}^{\infty} ih_i, r, w) \supseteq \bigcup_{i=0}^{\infty} f(ih_i, r, w) \quad \text{or finally}$$

$$f(\bigcup_{i=0}^{\infty} ih_i, r, w) = \bigcup_{i=0}^{\infty} f(ih_i, r, w).$$

This ends the proof that f is a continuous function in its first argument.

Let's now introduce a prefix-like cpo $\preccurlyeq$ in the domains R and W of the synchronizers:

(D10) for any $r_1, r_2 \in R$ $r_1 \preccurlyeq_R r_2$ iff $\forall o \in O \forall m \in I \cup S$,

$$((\forall t \in N\ r_1(o, t, m) = r_2(o, t, m)) \quad \text{or}$$
$$(\exists t^* \in N(\forall t \leqq t^* r_1(o, t, m) = r_2(o, t, m)) \;\&$$
$$(\forall t > t^*\ r_1(o, t, m) = \infty))).$$

The same kind of definition goes for $\leqslant_w$ in W, too.

Using the technique of the above theorem proof, it is easy to show that f is also continuous in its synchronizers with respect to $\leqslant$.

4. Conclusions. The main result of this paper is the theorem, stating that the function of a DS with shared memory is continuous in its arguments. The only significant restriction on the class of such DS is the requirement that their operators should be determinated.

The driving idea in the paper is the synchronization of actions by the synchronizer functions. They abstract in an elegant way the real-life synchronization and generalize different synchronization mechanisms (e. g. Keller's partial order of events in [6], Hewitt and Baker's combined order in [3], the flow of data in data flow systems, etc.)

The synchronizers serve another purpose, too. Passing them to the DS function as arguments captures the synchronization non-determinism of the DS behaviour (the "merge" type of non-determinism). This type of non-determinism is determined by the class of possible (feasible) synchronizers.

If the number of operators, reading from or writing into each object in the DS, is restricted to one, then the objects can behave like channels (lines) between operators when a suitable synchronization (ensuring one-way FIFO sequencing) is supplied. Therefore, the message-passing type of DS can be treated as a special kind of DS with shared memory, so some of the results about message-passing systems are reproduced in this paper as a side effect.

The study of structural manipulation of distributed systems with shared memory (composition of DS and DS with recursion) remains as a future work on the DS function properties. Another, more pragmatic direction of future work is the application of the proposed mathematical constructs and properties in the area of software engineering (e. g. development of software design verification, etc. techniques).

REFERENCES

1. J. Dean Brock, W. B. Ackerman. Scenarios: A Model of Non-determinate Computation. Formalization of Programming Concepts, Proc. of the International Colloquium Peniscola, Spain. *LNCS*, **107**, 1981, 252—259.
2. Formal Description of Programming Concepts. Proc. of the IFIP Working Conference on FDPC'77. North-Holland (1978).
3. C. E. Hewitt, H. Baker. Actors and Continuous Functionals, in [2], 367-387.
4. C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, **21**, 1978, No 8.
5. G. Kahn. The Semantics of a Simple Language for Parallel Programming. Proc. IFIP'74 1974, 471-475.
6. R. M. Keller. Denotational Models for Parallel Programs with Indeterminate Operators, in [2], 337-363.
7. R. M. Keller. Some Theoretical Aspects of Applicative Multiprocessing. Proc. Mathematical Foundations of Computer Science '80. LNCS, **88**, 1980, 58-74.
8. The Vienna Development Method: The Meta-Language. LNCS, **61**, 1978.

SBNIPI "Interprograma"
P. O. Box 795, Sofia 1000, Bulgaria

Received 22. 3. 1982