Report Number: WUCS-79-9

1979-01-01

Behavioral Abstraction of Communicating Sequential Processes

Authors: Takayuki D. Kimura

It is shown that behavioral semantics of Hoare's Parallel Commands can be formally specified by an extension of the regular expression, augmented by the shuffle operation and the inverse shuffle operation. As a corollary of the above, it is also shown that the problems of behavioral equivalence and deadlock-detection are solvable for the Parallel Commands.

BEHAVIORAL ABSTRACTION OF COMMUNICATING SEQUENTIAL PROCESSES

Takayuki Kimura

WUCS-79-9

January 1979

Department of Computer Science

Washington University

St. Louis, Missouri 63130

(314) 889-6122

ABSTRACT

It is shown that the behavioral semantics of Hoare's Parallel Commands can be formally specified by an extension of the regular expression, augmented by the shuffle operation and the inverse shuffle operation.

As a corollary of the above, it is also shown that the problems of behavioral equivalence and deadlock-detection are solvable for the Parallel Commands.

Key Words: Behavioral Semantics, Communicating Sequential Processes, Parallel Commands, Shuffle Operation, Inverse Shuffle Operation, Event Graph, Deadlock Detection.

1. Introduction

With the recent advancement of LSI technology and distinguished trends towards more distributed computation, the study of semantic models of parallel programming languages have shifted from those with shared memory communication mechanism to those with 'thin-wire' message oriented communication mechanism. [20]

More than a decade ago, the macromodule project at Washington University (St.Louis) successfully investigated the feasibility of designing and implementing asynchronous concurrent systems based on the idea of process coordination through thin-wire communication at the architecture level. [17]

Some programming languages adhere to the similar philosophy. Examples are Actor System of Hewitt [5], APPLE of Kimura [9], Data Flow Language of Dennis [13], PLITS at Rochester [1], Smalltalk at Xerox PARC [7], and Parallel Commands of Hoare [6]. Because of the nature of its communication mechanism, a process society specified by such a language can be characterized in a significant way by its communication behavior, i.e. its message transmission activities.

Riddle proposed an extension of the regular expression, called the 'message transfer expression' [18], later called the 'event expression' [16], for a formal description of such behavior. Harbermann's path expressions [4], Kimura's SC-expressions [10], and Shaw's flow expressions [19] are intended to serve similar purposes.

Some results on the power of such expressions are also known [10][16]. One fundamental discovery in this area of research is that the shuffle operation of the formal language theory [2], denoted here by +, characterizes the behavioral concurrency of two independent (non-communicating) activities. Recently, Kimura showed [11], by generalizing the synchronization mechanisms of event(flow) expressions, that the inverse of the shuffle operation ($\sim$) is sufficient to describe the behavior of a network of communicating concurrent activities.

In this paper we will apply this result to define the formal behavioral semantics of Hoare's Parallel Commands. As a corollary, we will show the decidability of deadlock detection for the PC.

Throughout this paper we try to avoid unnecessary formalism.

In Section 2, we give a definition of the concept of 'behavior' for communicating sequential processes. In Section 3, an extention of the regular expression is defined, and an event graph is introduced as the schema of a parallel command. The results of [11] are quoted and used to formulate a composition rule for event graphs. Section 4 shows the decidability of deadlock detection after formalizing the concept of deadlock. Section 5 gives future problems.

2. Behavior of Communicating Sequential Processes

By the 'behavioral semantics' of a parallel command, we mean the set of all possible sequences of event occurrences that can be generated by, and can be observed from the outside of, the process society specified by the parallel command. By an 'event occurrence', we mean an an instantaneous execution of a communication action (input/output command) in the society.

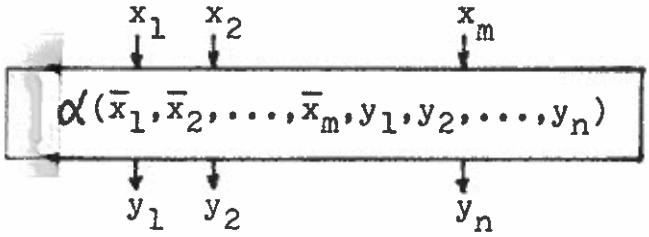
We assume that (1) no two event occurrences are simultaneous [15], (2) all processes are close enough in time-space so that for any interval (of time-space), the total number of event occurrences is finite, and there always exists a total ordering on them [12], and (3) no event occurs between the executions of an input command and the corresponding output command.

Even though the assumption (1) does not agree with Hoare's statement: "(input and output) Commands which correspond are executed simultaneously, .." [6], because of the assumption (3), the disagreement is not significant.

We identify an input command by an alphabet with an upper bar, and an output command by an alphabet without a bar. Input commands with the same source process and the same target variable type are identified as same. Similarly, output commands with the same destination process and the same type value are identified as same. We designate the behavior of a process society P, denoted by $|P|$, by a set of finite sequences of alphabets. Each alphabet represents an input or output command interacting with the outside of the society. Note that in our form-

alization the internal communication activities of the society do not appear in the behavior of the society.

Schematically, the behavior of a process society can be represented by:



where α is an expression that specifies a set of finite sequences of $\{\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_m,y_1,y_2,\ldots,y_n\}$. An execution of an input command is denoted by $\bar{x}_i$ and an output command by y_j. When there exists more than one such society with the lines for 'corresponding' input/output commands interconnected, we obtain what we call an event graph representing another hierarchy of process society with inter-social communication structure. Event graphs are discussed in Section 3.3.

Two examples follow:

Example 1: SQUASH (from Hoare [6])

```
X::*[c:character; west?c --->                          *(λ w̄
     [c≠asterisk ---> east!c                            (λ e
     ▯c=asterisk ---> west?c;                           ∪ λ w̄
        [c≠asterisk ---> east!asterisk; east!c          (λ ee
        ▯c=asterisk ---> east!upward arrow              ∪ λ e
   ]]]                                                  )))
```

$|X| = \{\lambda, \bar{w}e, \bar{w}\bar{w}e, \bar{w}\bar{w}ee, \ldots\}: (\bar{w}(e \cup \bar{w}(ee \cup e)))^*$

where λ denotes the null sequence and * is the closure of concatenation. Figure 1 gives an event graph for X.

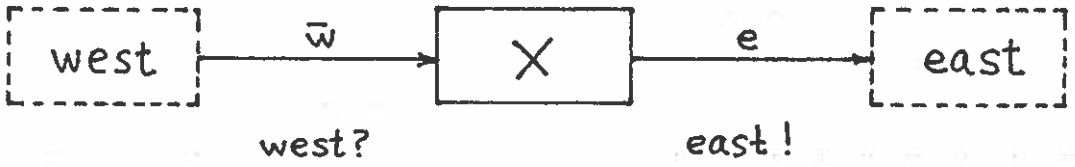
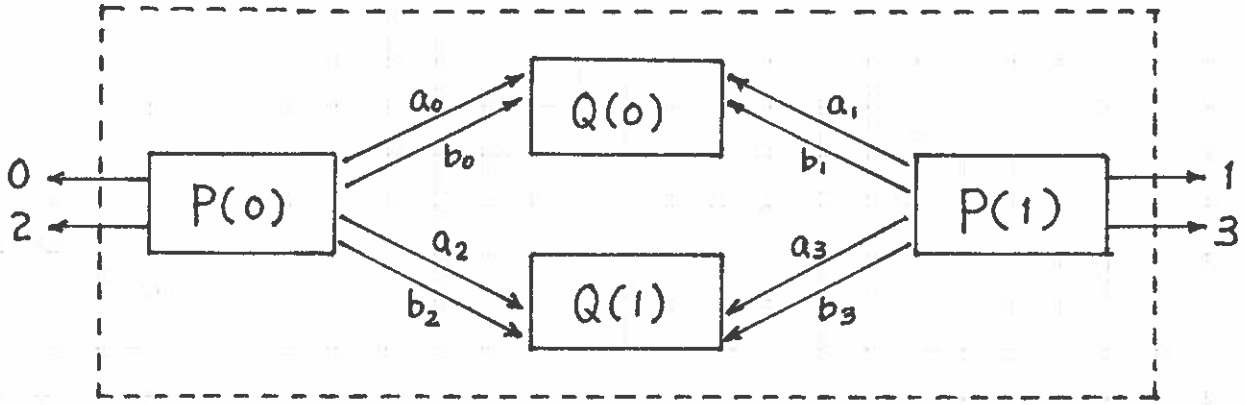


Figure 1: Event Graph of X




```
BS::[P(0)::Q(0)!"a";out!0;Q(1)!"a";out!2;          (a0 0 a1 2
      Q(0)!"b";Q(1)!"b"  ||                         b0 b1) +
   P(1)::Q(1)!"a";out!1;Q(0)!"a";out!3;           (a3 1 a2 3
      Q(1)!"b";Q(0)!"b"  ||                         b3 b2) +
   Q(0)::*[P(0)?"a" ---> P(0)?"b"  []             *(ā0 b̄0
           P(1)?"a" ---> P(1)?"b"  ]  ||             ā2 b̄2) +
   Q(1)::*[P(0)?"a" ---> P(0)?"b"  []             *(ā2 b̄2
           P(1)?"a" ---> P(1)?"b"  ]                 ā3 b̄3)
]
```

Figure 2: Event Graph of BS

Example 2: Binary Semaphore

```
BS::[P(i:0..1)::Q(i)!"a";out!i;Q(1-i)!"a";out!(i+2);
                Q(i)!"b";Q(1-i)!"b" ||
     Q(i:0..1)::*[(j:0..1)P(j)?"a" ---> P(j)?"b"]
    ]
```

$|P(0)|$: $a_0 0 a_2 2 b_0 b_2$

$|P(1)|$: $a_3 1 a_1 3 b_3 b_1$

$|Q(0)|$: $(\bar{a}_0\bar{b}_0 \cup \bar{a}_1\bar{b}_1)^*$

$|Q(1)|$: $(\bar{a}_2\bar{b}_2 \cup \bar{a}_3\bar{b}_3)^*$

$|BS|$: $0213 \cup 1302$

P(0)::Q(0)!"a" ≡ a_0

P(1)::Q(0)!"a" ≡ a_1

P(0)::Q(1)!"a" ≡ a_2

P(1)::Q(1)!"a" ≡ a_3

Similarly for $b_0 - b_3$.

An event graph of BS is given in Figure 2. It represents schematically the communication structure of the process society BS. We call such a graph as the event graph of a parallel command. A systematic derivation of $|BS|$ from $|P(0)|$, $|P(1)|$, $|Q(0)|$ and $|Q(1)|$, based on the event graph, is given in Section 3.3 in a more general form.

Notes: (1) BS becomes deadlocked after the activity:

$a_0\bar{a}_0 0 a_3\bar{a}_3 1$

(2) $|BS|$ is generated by the following activities:

$$\left\{ \begin{array}{l} a_0\bar{a}_0 0 a_2\bar{a}_2 2 b_0\bar{b}_0 b_2\bar{b}_2 a_3\bar{a}_3 1 a_1\bar{a}_1 3 b_3\bar{b}_3 b_1\bar{b}_1 \\ a_3\bar{a}_3 1 a_1\bar{a}_1 3 b_3\bar{b}_3 b_1\bar{b}_1 a_0\bar{a}_0 0 a_2\bar{a}_2 2 b_0\bar{b}_0 b_2\bar{b}_2 \end{array} \right\}$$

3. Extended Regular Expression (ERE) and Event Graphs

In this section we extend the regular expression [14] by adding the shuffle and its inverse operation. Then we define an event graph as a labeled directed graph with multiple arcs. The intended interpretation of an event graph is the communication behavior of a process society specified by a parallel command. Each node represents a sub-society (mostly a process) whose behavior is denoted by the ERE associated with the node. Each arc represents a communication channel (a pair of corresponding input/output commands) between two sub-societies.

From the ERE's of individual nodes, representing the internal communication behavior of the society, it is possible to construct an ERE for the entire graph representing the interaction of the society with its outside environment. A formula for such construction is given in Section 3.3.

3.1 Extended Regular Expressions (ERE)

Let Σ be an arbitrary finite alphabet, and let $L_1, L_2 \subseteq \Sigma^*$. The shuffle (+) and its inverse operation ($\sim$) are defined as:

$$L_1+L_2 \overline{\Delta} \{ x_1y_1x_2y_2\cdots x_ny_n \in \Sigma^* \mid x_1x_2\cdots x_n \in L_1 \wedge y_1y_2\cdots y_n \in L_2 \},$$

$$L_1\sim L_2 \overline{\Delta} \{ x_1x_2\cdots x_n \in \Sigma^* \mid x_1y_1x_2y_2\cdots x_ny_n \in L_1 \wedge y_1y_2\cdots y_n \in L_2 \}.$$

It is known that the family of regular languages are closed under both + and $\sim$. [3]

We define Extended Regular Expression as follows:

(1) All regular expressions are EREs.

(2) If A and B are EREs, denoting |A| and |B|, then (A+B) and (A$\sim$B) are also EREs, denoting |A|+|B| and |A|$\sim$|B| respectively.

Example 3:

$|((ab+cd)\sim cb)| = \{abcd, acbd, acdb, cabd, cadb, cdab\} \sim \{cb\} = \{ad\}.$

Notes: (1) ERE denotes a regular set only.

(2) The closure of the shuffle operation is not included in ERE, because, first, it is not necessary for uninterpreted schemata for parallel commands (in which no recursion is allowed), second, it is known that once the closure operation is allowed, the power of expressions (with $\sim$) becomes equivalent to that of Turing machine [16]. This denies solvability of many analysis problems.

3.2 Construction of ERE for a Sequential Command

We say a command is sequential if it does not contain a parallel command. It is straightforward to show that the behavioral semantics of a sequential command is a regular set, and therefore can be specified by a regular expression.

Table I gives a translation table from a sequential command to a regular expression. Note that only input/output commands are translated into the finite base alphabets $\sum$: an input command for $\bar{a}$ and output command for a, where $a, \bar{a} \in \sum$.

Example 4: DISASSEMBLE (from Hoare [6])

```
west::*[cardimage:(1..80)character;cardfile?cardimage --->
          i:integer; i:=1;
          *[i<80 ---> X!cardimage(i); i:=i+1];
          X!space
        ]
```

$|west| = (\bar{c}w^*w)^*$

TABLE I: Sequential Command Behavior Translation Table

Command Syntax	ERE
SKIP	λ
A := B	λ
<guard list>	λ
A?B	$\bar{a}$
A!B	a
α ---> β	$\alpha\beta$
$\alpha ; \beta$	$\alpha\beta$
$*[\alpha]$	$(\alpha)^*$
$\alpha \,[]\, \beta$	$\alpha \cup \beta$
$(i:m..n)(\alpha_i$ ---> $\beta_i)$	$\bigcup_{i=m}^{n} (\alpha_i \beta_i)$

Example 5: ASSEMBLE (from Hoare [6])

```
east::lineimage:(1..125)character;                                  λ
      i:integer; i:=1;                                              λλ
      *[c:character; X?c --->                                       *(λ ē
          lineimage(i):=c;                                           λ
          [i<124 ---> i:=i+1                                        (λλ
          [] i=125 ---> lineprinter lineimage; i:=1                 ∪λpλ
        ]]                                                          ))
      [i=1 ---> skip                                                (λλ
      [] i>1 ---> *[i<125 ---> lineimage(i):=space;i:=i+1];         ∪λ*(λλλ)
         lineprinter!lineimage                                      p
      ]                                                             )
```

$|east| = (\bar{e}(\lambda \cup p))^*(\lambda \cup p)$

Example 1 of Section 2 also illustrates a derivation of a regular expression for SQUASH.

3.3 Construction of ERE for a Parallel Command

We represent a process society by a directed graph, called an event graph, which is similar to Kahn's Schema [8]. It is a directed graph with multiple arcs. All arcs are labeled differently by a finite set of symbols Σ. Each node is also labeled by an ERE with $\Sigma \cup \bar{\Sigma} \cup \Omega$ as its base alphabet, where $\Sigma \cap \Omega = \phi$. The alphabet $\bar{\Sigma}$ represents input commands for internal communication, Σ represents output commands for internal communication, and Ω represents input/output commands interfacing with the outside of the society. For example, Figure 2 illustrates

an event graph for the BS process society. Note that the communication behavior of a process society is completely characterized by such an event graph.

In order to incorporate an event graph as one node in another super graph, we must have a rule for constructing an ERE (global) from the ERE's of individual (local) nodes. Such a composition rule is given in Kimura [11]. The following theorem is a version of the results adopted for event graphs.

Theorem:1:

Let $(\{A_i \mid 1 \le i \le m\}, \{a_j \mid 1 \le j \le n\})$ be an event graph, where A_i is an ERE for the i-th node, and a_j is a label for the j-th arc. Then, the ERE for the event graph is:

$$(\overset{m}{\underset{i=1}{+}} A_i) \sim (\overset{n}{\underset{j=1}{+}} (a_j \bar{a}_j)^*)$$

$$= (A_1 + A_2 + \ldots + A_m) \sim ((a_1\bar{a}_1)^* + (a_2\bar{a}_2)^* + \ldots + (a_n\bar{a}_n)^*)$$

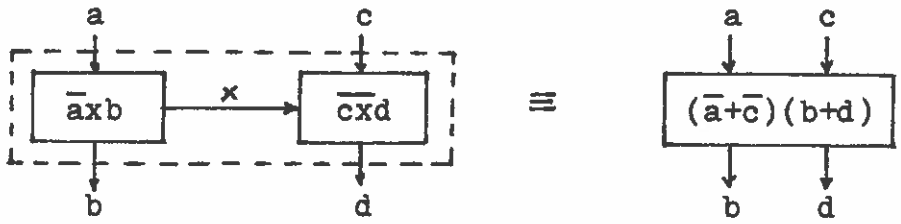
where $+$ is the shuffle operation, and $\sim$ is the inverse shuffle operation.

(End of Theorem)

If there is no communication between the node processes, then the society behavior is an arbitrary interleaving of the individual process activities, with $(\overset{}{\underset{i}{+}} A_i)$ representing such behavior. The sequence $(a_j\bar{a}_j)$ represents a logical requirement for the communication channel j that an execution of the output command must preceed that of the corresponding input command, and that no other action should occur in between. Therefore, since each communication channel is independent with others, $\overset{}{\underset{j}{+}}(a_j\bar{a}_j)^*$ represents the logical constraint for communication behavior in the entire society. The inverse shuffle operation extracts

only such communication activities that satisfy the constraint, from the uncontrolled behavior of the society $(\underset{i}{+}A_i)$.

Example 6:

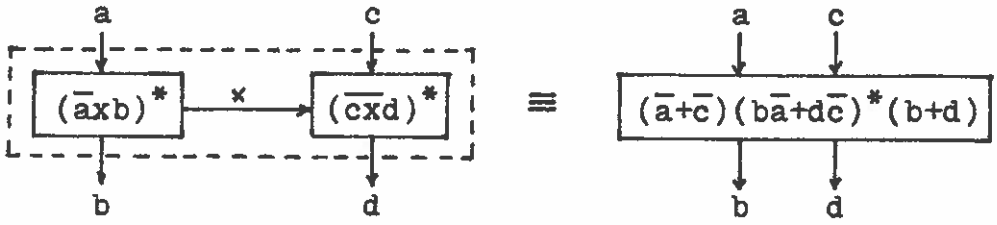


$$(\bar{a}xb+\bar{c}xd)\sim(x\bar{x})^* = (\bar{a}+\bar{c})(b+d)$$

$\Sigma = \{x\}$

$\Omega = \{a, b, c, d\}$

Example 7:



$$((\bar{a}xb)^*+(\bar{c}xd)^*)\sim(x\bar{x})^* = (\bar{a}+\bar{c})(b\bar{a}+d\bar{c})^*(b+d)$$

There are two types of nodes in an event graph: one group of nodes which interface with the outside, called the frontier nodes, and the other group of nodes which communicate with other members of the society only, called the internal nodes.

The following theorem is a corollary of the above theorem 1.

Theorem 2: For a given event graph,

let $\{A_i\}$ be the set of ERE's for the frontier nodes, and

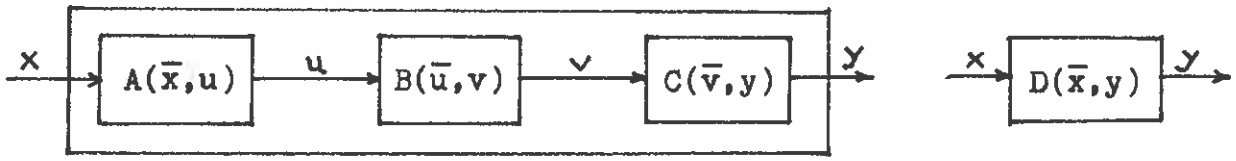
$\{B_j\}$ be the set of ERE's for the internal nodes.

Then, the ERE for the event graph is: $(\underset{i}{+}A_i)\sim(\underset{j}{+}B_j)$.

(End of Theorem 2)

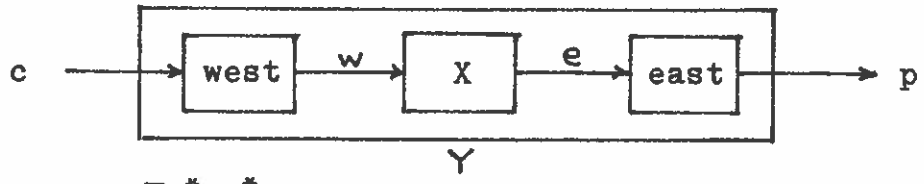
In short, the behavior of internal nodes restrict the behavior of frontier nodes, and this defines the system behavior.

A special case of the theorem is as follows:



where $D(\bar{x},y) \equiv (A(\bar{x},u)+C(\bar{v},y))\sim B(\bar{u},v)$.

Example 8: Conway's Problem (from Hoare [6])



west: $(\bar{c}w^*w)^*$ (Example 4)

X: $(\bar{w}(e\cup\bar{w}(ee\cup e)))^*$ (Example 1)

east: $(\bar{e}(\lambda\cup p))^*(\lambda\cup p)$ (Example 5)

$$Y(\bar{c},p) = (\bar{c}w^*w)^*+(\bar{e}(\lambda\cup p))^*(\lambda\cup p)\sim(\bar{w}(e\cup\bar{w}(ee\cup e)))^*$$

Example 9: Binary Semaphore (Figure 2)

P(0) and P(1) are the frontier nodes.

Q(0) and Q(1) are the internal nodes.

By the theorem 2,

$$|BS| = ((a_0 0 a_2 2 b_0 b_2)+(a_3 1 a_1 3 b_3 b_1))$$
$$\sim((\bar{a}_0\bar{b}_0\cup\bar{a}_1\bar{b}_1)^*+(\bar{a}_2\bar{b}_2\cup\bar{a}_3\bar{b}_3)^*) = \{0213,1302\}.$$

In general, our composition rule for an ERE for a parallel command is as follows:

(1) Construct an ERE for each process command.

If a process command is nested inside another one, then recursively apply the steps (1) - (3) inside-out.

(2) Identify the frontier process commands and the internal process commands.

(3) Using the theorem 2, compute $(+Ai) \sim (+B_j)$.

4. Deadlock Detection

We conceive the deadlock problem as a problem on communication protocol; i.e.[a] problem about an agreement among the society members on how to communicate with each other. Where no communication exists, no deadlock exists. A deadlock occurs in a process society S with respect to a communication protocol C, when there exists a partial activity of S admissible under C that can not be completed an activity of S admissible under C. In other words, S is deadlock-free under C, if any partial activity of S legal under C can be completed to a legal activity of S.

In order to define formally the concept of deadlock[v]event for graphs representing parallel commands, we need another language operator @. The prefix operator, @, is defined as:

$@L \overline{\Sigma} \{x \in \Sigma^* \mid xy \in L \text{ and } y \in \Sigma^*\}$, where $L \subseteq \Sigma^*$.

This operator represents partial activities, and it is known that the family of regular languages are closed under @. [2]

Definition: Let $E = (\{A_i \mid 1 \le i \le m\}, \{a_j \mid 1 \le j \le n\})$ be an event graph, and let $A \overline{\Sigma} (\dot{+}_i A_i)$, $B \overline{\Sigma} (\dot{+}_j (a_j \bar{a}_j)^*)$.

E is deadlock-free iff $@A \sim B \subseteq @(A \sim B)$

(End of Definition)

Note: $A \sim B$ is the behavior of the graph.

Example 10: BS is not deadlock-free.

$A = (a_0 0 a_2 2 b_0 b_2) + (a_3 1 a_1 3 b_3 b_1) + (\bar{a}_0 \bar{b}_0 \cup \bar{a}_1 \bar{b}_1)^* + (\bar{a}_2 \bar{b}_2 \cup \bar{a}_3 \bar{b}_3)^*$

$B = (a_0 \bar{a}_0)^* + (a_1 \bar{a}_1)^* + (a_2 \bar{a}_2)^* + (a_3 \bar{a}_3)^* + (b_0 \bar{b}_0)^* + (b_1 \bar{b}_1)^* + (b_2 \bar{b}_2)^* + (b_3 \bar{b}_3)^*$

Consider a partial activity: $\alpha = a_0 \bar{a}_0 0 a_3 \bar{a}_3 1$.

Then, since $a_0\bar{a}_0 0 a_3\bar{a}_3 1 a_2 2 b_0\bar{b}_0 a_1 3 b_3\bar{b}_3 b_1 \in A$, $\alpha \in @A$ and $\alpha \sim B = \{01\}$. However, $01 \notin @\{0213, 1302\} = \{\lambda, 0, 02, 021, 0213, 1, 13, 130, 1302\}$.

That is to say, $01 \in @A \sim B$ but $01 \notin @(A \sim B)$.

Therefore BS is not deadlock-free by definition.

Theorem 3:

It is decidable whether an event graph is deadlock-free or not.

Proof: The family of regular languages are closed under $\cdot, \cup, *, +,$ and $\sim$. The subset inclusion problem is known to be decidable for regular languages.

Q.E.D.

By the same argument, we can claim:

Theorem 4:

It is decidable whether two event graphs are equivalent or not.

As far as parallel commands are concerned, we can say that in one level of abstraction (i.e. uninterpreted behavioral abstraction) their equivalence and deadlock-freeness problems are decidable. The next level of abstraction will involve some interpretation of message contents transmitted through communication channels.

5. Discussion and Conclusion

The following characteristics of the parallel command make it possible to formalize its behavior by a regular set as we did in this paper:

(1) No recursion is allowed.

(2) Communication is through a 'thin-wire' (no shared memory).

(3) No memory element exists in a communication link (no buffer).

A similar formalism can be constructed for G.Kahn's program schemata [8], first without recursion. However, since its communication channel is a fifo queue (unbounded), a specification of communication behavior through such a channel requires the shuffle closure operation (#); i.e. in stead of $(x\bar{x})^*$ for each communication channel, we must use $(x\bar{x})^{\#}$, which is a Dyck language, for interaction between an output command x and the corresponding input command $\bar{x}$. Therefore the behavior of such a society will not be regular anymore, and the decidability results may or may not be applicable. The composition rule (Theorem 2) is valid for Kahn's schema, too.

There are two problems immediately related to this work.

(1) Calculus for ERE.

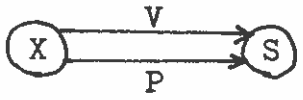
The nature and power of + and # have been studied to some extent in the past [10], but not for $\sim$. An algorithm for translating an ERE to a regular expression must be discovered.

(2) Interpreted behavioral abstraction.

Our formalism is completely uninterpreted. Some degree of interpretation must be included in a behavioral abstraction.

Consider the following command for integer semaphore:

```
S::val:integer; val:=0;
   *[X?V() ---> val:=val+1
    []val>0; X?P() ---> val:=val-1
    ]
```



We want to say that the behavior of S is $(VP)^{\#}$, but our formalism gives $|S| = (V \cup P)^{*}$. In this example, interpretation of val:integer makes the difference.

References:

(1) Feldman,J.A., A programming methodology for distributed computing (among other things), TR9, CS Dept., University of Rochester, 1977.

(2) Ginsburg,S.,The Mathematical Theory of Context-free Languages, McGraw-Hill, 1966.

(3) Ginsburg,S., Spannier,E.H., Mappings of languages by two-tape devices, JACM 12,3, July 1965.

(4) Habermann, A.N., Path expressions, CS Dept., Carnegie-Mellon University, June 1975.

(5) Hewitt,C., Atkinson,R., Parallelism and synchronization in actor systems, 4th ACM Symp. on POPL, Calif.,1977.

(6) Hoare,C.A.R., Communicating sequential processes, CACM 21,8, August 1978.

(7) Ingalls,D., The smalltalk-76 programming system, 5th ACM Symp. on POPL, Arizona, 1978.

(8) Kahn,G., The semantics of a simple language for parallel programming, IFIPS Proceedings, 1974.

(9) Kimura,T., APPLE: A parallel programming language for process structuring and interprocess communication, 15th NBS-ACM Technical Symp., June 1976.

(10) Kimura,T., An algebraic system for process structuring and interprocess communication, 8th ACM SIGACT Symp., 1976.

(11) Kimura,T., Formal description of communication behavior, submitted to Johns Hopkins Conference on Information Sciences and Systems, Baltimore, Md., March 1979.

(12) Lamport,L., Time, clocks, and ordering of events in a distributed system, CACM 21,7, July 1978.

(13) Rumbaugh,J.E., A parallel asynchronous computer architecture for data flow programs, Ph.D. Thesis, MIT, May 1975.

(14) McNaughton,R., Yamada,H., Regular expressions and state graphs, IRE Trans. Electron. Comp., vol EC-9, 1960.

(15) Miller,R.E., Yap,C.K., On formulating simultaneity for studying parallelism and synchronization, 10th ACM SIGACT Symp., 1978.

(16) Ogden, W.F., Riddle,W.E., Rounds,W.C., Complexity of expressions allowing concurrency, 5th ACM Symp. on POPL, Arizona, 1978.

(17) Orstein,S.M., Stucki,M.J., Clark,W.A., A functional description of macromodules, SJCC vol.30, 1967.

(18) Riddle,W.E., The modelling and analysis of supervisory systems, Ph.D. Thesis, Stanford University, March 1972.

(19) Shaw,A.C., Software descriptions with flow expressions, IEEE Trans. on Software Engineering, vol. SE-4,3, 1978.

(20) Metcalfe,R.M., Strategies for interprocess communication in a distributed computing system, Proc. of the Symposium on Computer-Communications Networks and Teletraffic, Polytechnic Press, New York, 1972.