WIS-CS-193-73
Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Received August 7, 1973

VERIFICATION OF PROCESS STRUCTURES OF INTERACTING DIGITAL SYSTEMS

by

R. T. Johnson
and
D. R. Fitzwater

Computer Sciences Technical Report #193

August 1973

VERIFICATION OF PROCESS STRUCTURES OF INTERACTING DIGITAL SYSTEMS

by

R. T. Johnson and D. R. Fitzwater

ABSTRACT

A formal, universe for systems design has been developed, in which representations of interacting digital, systems are interpreted by a deterministic automaton acting on state information in the form of character strings. Since process structures in this universe can be described with regular languages, a new interpretation is defined in which regular languages represent state information. Computations under this new interpretation contain the previous ones (with some loss of detail), making it possible to prove assertions about the original string computations. An example is given to show the formulation, and algorithmic verification, of some interesting properties of two asynchronous communicating systems.

I. INTRODUCTION

A. Design Feedback

A system designer needs the kind of feedback from his design that is routinely supplied to a programmer by a compiler. The programmer may intend to write a Fortran program, while the compiler rejects it as not being written in Fortran at all. Further, upon testing the corrected program, the programmer may find that he did not say, in Fortran, what he intended to say. Feedback information from debugging runs is usually essential before he can be satisfied (perhaps, still, improperly) that he has written the intended program. The designer of a complex of asynchronous, interacting digital systems is faced with an even tougher problem, and currently has few tools to provide any feedback information.

The purpose of this paper is to present some aids for the system designer, and to build a base for more. The existence of these tools depends on the design's being expressed as a representation in a formal definition system, because only then can we be sure of the semantics intended. It would be very unwise to claim "proof" about a design written in a programming language, for instance, when even the meaning of each arithmetic operation depends on the hardware details of the specific computer involved.

Furthermore, the design representation must be effective, i.e. capable of being interpreted by a deterministic automaton. This is the only way that all the observable, dynamic behavior of the system complex can be studied. We have chosen the formal definition universe described in [1] because it fulfills these requirements, and promises to facilitate our task of deriving and verifying assertions about properties of the system complex.

A substantial amount of feedback information is immediately available from the interpreter of these formal definitions. Verifications of syntactic form and trial "debug" runs may be automatically produced for the designer. Such runs would rarely prove that the designed system complex has the desired properties: the designer (as well as the programmer) needs a more powerful tool. Indeed, because of race conditions between the asynchronous systems of the complex, no reasonable number of trial runs would be sufficient to instill much confidence that the desired properties were present.

Fortunately, the verification of many desirable properties is not dependent on the fine detail of computations presented by trial runs. A suitable abstraction of the computation (that can be shown to preserve the essential information) can provide the equivalent of an infinite number of trial runs. This current work is concerned with just such an abstraction, algorithmically provided. This work is presented more completely in [4]. We will discuss it here by defining a simple example system complex and tracing its analysis with respect to some process structure assertions.

B. An Example System Complex

We will define a small system complex containing two systems which operate asynchronously. The first system we call the Buffer. Its purpose is to receive and enqueue characters from some external source and transmit them on demand in a first-in - first-out manner. In various contexts this system would be called either a "store and forward" system or a "producer-consumer" system. The second system, called the Consumer, requests characters from the Buffer, one at a time, and performs some unspecified operation on "words" which it constructs from the characters.

The string representations of these systems is in Figure 1. For specific details of the representation and how systems in the representation are interpreted, see the discussions by Fitzwater and Smith [1]. These systems are extensions of Post systems designed to model digital system complexes with well-defined processes. All operators within a system work in parallel during a system process step and if their antecedent patterns match process states in the system state set ($\underline{\sigma}$), then their consequent strings all contribute to the next state set. The completion of the application of the operators to $\underline{\sigma}$ defines the end of a system step, and a new set of values will be assigned to $\underline{\sigma}$. During a system step the contents of the state set is invariant. Separate systems work asynchronously and communicate via messages on channels as shown in Figure 2. The Buffer System contains three processes. The first consists of an alternating sequence of r:0 and r:1 states used as a signal between Buffer and Consumer. r:0 is a signal that Consumer is awaiting a character so that one should be sent as soon as it is available. The r:1 signal accompanies each character to Consumer. Buffer will continue to enqueue characters until it receives the r:0 signal again.

The second process in Buffer is the channel process with process states "new" and "new $<\alpha>$" where "$<\alpha>$" is one of the characters to be enqueued. We have left the source of these messages unspecified. If the complex is to work as we expect, this source must neither send multiple messages simultaneously, nor send at a rate faster than Buffer can handle.

The third process in Buffer is the character queue holding received (but not requested) characters. It is intially the empty string.

The Consumer System also has three processes; r:0 and r:1 are again the synchronizing signals with the other system. Each new character is concatenated on the right of the "word" process. The unspecified restricted processor <process word> is also applied (a restricted processor is very similar to an ordinary system, and so <process word> could perform any self-contained algorithmic operation on the character string). The "use" process is a channel over which the requested characters are received.

Buffer ::=

{$\underline{\sigma:}$ r:0 $\underline{\text{and}}$ new $\underline{\text{and}}$ λ

$\underline{x:}$ ℓ m n o p q r s t u v w x y z 0 1

$\underline{\pi:}$ $r{:}0\ \underline{\text{and}}\ \bar{\Delta}\,\$ \rightarrow r{:}1 \rightarrow r{:}0$ $\underline{\text{or}}$

$r{:}0\ \underline{\text{and}}\ \bar{\Delta}\,\$ \rightarrow \bar{\Delta}_1 \rightarrow \text{use}$ $\underline{\text{or}}$

$r{:}0\ \underline{\text{and}}\ \text{new}\,\$\ \underline{\text{and}}\ \bar{\Delta}\,\$ \rightarrow \$_2\$_1$ $\underline{\text{or}}$

$r{:}0\ \underline{\text{and}}\ \text{new}\,\$\ \underline{\text{and}}\ \lambda \rightarrow \$_1$ $\underline{\text{or}}$

$r{:}1\ \underline{\text{and}}\ \text{new}\,\$\ \underline{\text{and}}\ \$ \rightarrow \$_2\$_1$ $\underline{\text{or}}$

$\text{new}\ \$ \rightarrow \text{new}$ $\underline{\text{or}}$

$r{:}\bar{\Delta} \rightarrow r{:}\bar{\Delta}_1$ }

Consumer ::=

{$\underline{\sigma:}$ r:0 $\underline{\text{and}}$ word $\underline{\text{and}}$ use

$\underline{x:}$ ℓ m n o p q r s t u v w x y z 0 1

π: $\text{word}\ \{\$\,\langle\text{process word}\rangle\}\ \underline{\text{and}}\ \bar{\Delta} \rightarrow \text{word}\ \$_1\bar{\Delta}_1$ $\underline{\text{or}}$

$r{:}\bar{\Delta} \rightarrow r{:}\bar{\Delta}_1$ $\underline{\text{or}}$

$r{:}\bar{\Delta} \rightarrow \text{use}$ $\underline{\text{or}}$

$r{:}1 \rightarrow r{:}0 \rightarrow r{:}1$ $\underline{\text{or}}$

$\text{word}\ \{\$\langle\text{process word}\rangle\}\ \underline{\text{and}}\ \text{use} \rightarrow \text{word}\ \$_1$ }

FIGURE 1: DEFINITION OF A TWO-SYSTEM COMPLEX.
$\underline{\sigma:}$ is the system state set.
$\underline{x:}$ is the vocabulary for value strings of variable pattern elements ($\$$ and $\bar{\Delta}$).
$\$$ will match any string over the vacabulary.
$\bar{\Delta}$ will match any character in the vocabulary.
$\underline{\pi:}$ is the set of operators separated by $\underline{\text{or}}$. Double arrow operators have the form antecedent $\rightarrow$ message $\rightarrow$ channel.

	Buffer States	Consumer States
	{r:0,new,λ}	{r:0,word,use}
	⋮	⋮
	{r:0,~~new~~ new m,λ}	{r:0,word,use}
	{r:0,new,m}	
	{~~r:0~~ r:1,new,λ}	{~~r:0~~ r:1,word,~~use~~ m}
	{r:1,~~new~~ new n,λ}	{~~r:1~~ r:0,word m,use}
	{~~r:1~~ r:0,~~new~~ new o,n}	
T	{r:0 r:1,~~new~~ new p,o}	
I	{r:1,~~new~~ new q,op}	{~~r:0~~ r:1,word m,~~use~~ n}
M	{r:1,new, pq}	{~~r:1~~ r:0,word mn, use}
E	{~~r:1~~ r:0,new,opq}	{r:0, word mn,use}
	{~~r:0~~ r:1,new,pq}	{~~r:0~~ r:1,word mn,~~use~~ o}
↓	⋮	⋮

etc.

FIGURE 2: A typical computation on the example system complex. Each horizontal line represents a new $\underline{\sigma}$. The struck-out (/) process states are those which act as channel names over which messages were received during the step. The completion times of system steps in each system are independent.

C. Verification of System Complex Assertions

This complex is simple enough that its correctness may be obvious. It does have sufficient structure so that the techniques we wish to illustrate may be applied to it. In the previous section we described certain expectations we have about how the system complex is to function. We would like to verify these properties in as automatic a fashion as possible.

The first step is to classify all of the process states in which we are interested into a finite set of regular languages. If two process states are in the same regular language, then we do not wish to distinguish between them in the subsequent analysis. The union of all the languages for the complex should contain all of the process states which we expect ever to appear in computations. This classification defines a mapping for each system representation from the state space in which each system state is a finite set of strings to the state space in which each system state is a finite set of regular languages. In the image space there are only a finite number of possible image states and consequently only a finite number of image computations. Furthermore, we can apply the operators of the system representations to the image states directly in such a way that the computations in the image space are images of the computations in the finite-languages state space. Horning and Randell [3] have a good tutorial article about general processes and state spaces, including the concepts of image states and image processes.

The computations in the image space are simpler, so the properties of the system complex we are interested in will show up more clearly.

All the processes within a system are synchronous, so that we have a chance of predicting from the initial state set those process states which will occur simultaneously.

In Table 1 , we present a set of image process states ($\{B_1,\ldots,B_5\}$ and $\{C_1,\ldots,C_5\}$) and a set of system states ($\{SB_1,SB_2,SB_3\}$ and $\{SC_1,SC_2\}$) for the Buffer and the Consumer, respectively. α is the set of characters $\{1,m,n,\ldots,z\}$. For this example, there are only two infinite languages in the complex: "α^*" in Buffer and "word α^*" in Consumer. In a more complex system, many of the image states would be infinite and would have more interesting structures than just arbitrary strings.

Since inter-system communication is by messages, the successors to states depend on the status of the message transmissions. Thus we must also provide a set of (message, channel) language pairs as follows:

M_1:(r:0,r:1)
M_2:(r:1,r:0)
M_3:(new α, new)
M_4:(α, use)

The first regular expression defines the image message language, while the second defines unique channel names.

We represent the state of the system complex as a vector (B,C) where B is a pair $(\sigma_B,\sigma_B{}''')$ and $C = (\sigma_C,\sigma_C{}''')$. σ_i is the current state set for SR_i $(i \in \{B,C\})$ and σ_i''' is the current status of message transmission to that system. In any given state vector σ_i''' will contain 0,1, or "many" of each of the (message, channel) language pairs for the system complex.

We will derive a number of properties which will verify some assertions about this system complex. The first assertion is that the specified image process and sytem states are closed under all system transformations. Another assertion is that there is never more than one character in transit from Buffer to Consumer at any time in any computation. One more assertion we wish to verify is that, apart from the indeterminacies induced by inter-system interactions, the computations in each system are deterministic: since we have specified enough structure in our image space so that the image computations within a system are deterministic, the computations themselves must be, at least so far as distinguishable by the regular language process states. It will also be clear from the image computation graph that the signals r:0 and r:1 serve to synchronize the communication of characters between Buffer and Consumer regardless of their relative rates of operation.

	Buffer System	Consumer System
Process State Languages	B_1: r:0	C_1: r:0
	B_2: r:1	C_2: r:1
	B_3: new + new α	C_3: word α^*
	B_4: $\alpha^* - \lambda$	C_4: use
	B_5: λ	C_5: α
System State Sets	SB_1: $\{B_1, B_3, B_5\}$	SC_1: $\{C_1, C_3, C_4\}$
	SB_2: $\{B_1, B_3, B_4\}$	SC_2: $\{C_2, C_3, C_5\}$
	SB_3: $\{B_2, B_3, B_4, B_5\}$	

TABLE 1: Hypothesized image states for example complex. The operators +, -, and * represent union, difference, and closure of regular expressions, respecitvely. λ represents the null string; α is $\{\ell, m, n, \ldots, z\}$.

II. APPLYING OPERATORS TO REGULAR LANGUAGES

We must describe the image computations in terms of operations on image process states. Each such computation involves the finite transformation of a regular expression, instead of the infinite number of transformations that would be required to process each sentence in the language as defined in [1]. We will discuss briefly these computations in the context of our example system complex.

We will first work out in detail how the operator:

$$\text{r:0 } \underline{\text{and}} \text{ new } \$ \ \underline{\text{and}} \ \bar{\Delta} \ \$ \rightarrow \$_2\$_1$$

matches in the state set SB_2 and contributes to the next state set. In order to perform the matching we construct a reduced finite state acceptor for the union of the languages in SB_2 and then apply the antecedents to the acceptor.

A. Matching a Pattern.

The union of the languages in SB_2 has the regular expression representation $r{:}0 + \text{new}\alpha + \alpha^* - \lambda$ where α is $\ell + m + \ldots + z$. The reduced acceptor for this language is shown in Figure 3.

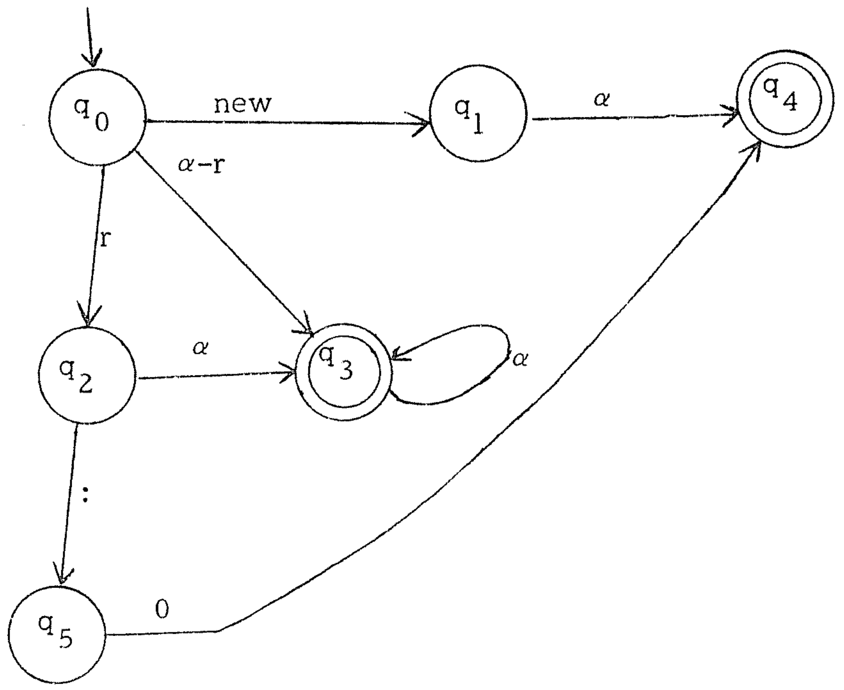

FIGURE 3: The reduced acceptor a for r:0 + new $\alpha + \alpha^* - \lambda$. A transition labeled with α stands for the set of all transitions under characters in l + m + . . . + z, and similarly for the transition labeled α - r.

The result of matching a pattern P in a regular language L is as follows: Let $M = (Q, \Sigma, q_o, F, \hat{q})$ be a reduced finite-state acceptor for L, where $Q = \{q_0, \ldots, q_m\}$ is the set of states, Σ is the character set, q_0 is the initial state, $F \subseteq Q$ is the set of final states, and $\hat{q} : Q \times \Sigma \rightarrow Q$ is the transition function. Let P be $p_1 \ldots p_n$ where each p_i, $1 \leq i \leq n$ is either a character, a single character variable (such as $\bar{\Delta}$), or a \$. Each <u>match</u> of P in M is a function

$$m' \colon \{p_1, \ldots, p_n\} \rightarrow Q \times Q$$

such that $m'(p_1)_1 = q_0$ ($m'(p)_i$ denotes the ith component of $m'(p)$),
$m'(p_i)_2 = m'(p_{i+1})_1$, $1 \leq i \leq n-1$, and
$m'(p_n)_2 \in F$.

Also, if $m'(p_i) = (q_r, q_s)$, then there must be a string accepted by the acceptor $(Q, \Sigma, q_r, \{q_s\}, \hat{q})$ such that p_i matches the string.

For each match m' of P in M, the result of the match of P in L is a function

$$m \colon \{p_1, \ldots, p_n\} \rightarrow \Sigma^*$$

defined by $m(p_i) =$

$\{s \mid s$ is accepted by $(Q, \Sigma, m'(p_i)_1, \{m'(p_i)_2\}, \hat{q})$ and p_i matches $s\}$ for each $1 \leq i \leq n$.

This may be clarified by the example in Table 2.

	m_{11}'	m_{11}
r	(q_0,q_2)	$\{r\}$
:	(q_2,q_5)	$\{:\}$
0	(q_5,q_4)	$\{0\}$

	m_{21}'	m_{21}	m_{22}'	m_{22}
new	(q_0,q_1)	$\{new\}$	(q_0,q_1)	$\{new\}$
\$	(q_1,q_1)	$\{\lambda\}$	(q_1,q_4)	α

	m_{31}'	m_{31}	m_{32}'	m_{32}	m_{33}'	
$\overline{\Delta}$	(q_0,q_2)	$\{r\}$	(q_0,q_2)	$\{r\}$	(q_0,q_3)	$\alpha - \{r\}$
\$	(q_2,q_2)	$\{\lambda\}$	(q_2,q_3)	$\alpha\alpha^*$	(q_3,q_3)	α^*

TABLE 2: Matches of Example Operator on the Union of the languages in SB_1 .

B. Constructing the Consequent

For each sequence of matches of the antecedents of an operator in a regular language, a consequent language is constructed as the concatenation of the languages associated by the matches together with the constants in the consequent pattern.

The consequent language constructed by an operator applied to a regular language is the union of all languages constructed for each match.

For example, the consequent language constructed by the example operator is

$$\{\lambda\}\{\lambda\} + \{\lambda\}\alpha + \alpha\alpha^*\{\lambda\} + \alpha\alpha^*\alpha + \alpha^*\{\lambda\} + \alpha^*\alpha = \alpha^*.$$

For the entire set of operators of Buffer applied to SB_1 , the consequent language is $\alpha^* + \alpha^* +$ new + r:0 = r:0 + new + α^* . The first operator matches and sends the two messages

$$\{r{:}1\} \rightarrow \{r{:}0\} \text{ and } \alpha \rightarrow \{use\} .$$

C. Closure of Process State Languages

In general, the language produced by the operators applied to a regular language need not be equal to any of the process state languages or to any union of them. The process state languages were to be a classification of the process states of the SR, so that each sentence in this constructed language must be in at least one of the process state languages. If this were not so, the asserted image process states would not be closed under the system transformations.

We can now derive some valuable information about the processes supported by the system, by defining the successors to an image process state to be all image process states that have a non-null intersection with the consequent language. When this has been done for all operators in the system we have not only verified the closure hypothesis but also defined a finite graph at the specified level of resolution that characterizes the system processes. Such graphs easily provide information about many process properties. The designer need only cast his assertions in the form of hypothesized regular languages.

III. STEPS OF THE SYSTEM COMPLEX

We can now develop a concept of process applicable to the entire system complex. We must find a way to define the state of a complex and a step from one state to another. There is no notion of relative system process step speeds in our formal universe and so each possible race condition result must be shown. Our approach is based on a technique developed in [2] . We can simulate message reception and determine which of the process state languages overlap the constructed language. In our example, when Buffer makes a step with initial state SB_2, then the constructed language is $r{:}1 + new + \alpha^*$ since the message $\{r{:}1\} \rightarrow \{r{:}0\}$ has been transmitted and such transmissions are "instantaneous" within a single system. This constructed language overlaps B_2, B_3, B_4 and B_5. Thus, there is at least one representative string from each of the languages in SB_2, i.e. $SB_3 \subseteq \{B_2,B_3,B_4,B_5\}$. Thus, SB_3 is the successor state set to SB_2 when Buffer completes a step.

Information about messages must also be retained and used to construct the language which determines the successor states in each SR . We do this by including with the state of each SR at each "time" during a computation, a record of which messages have been transmitted to that SR . Since delays in transmissions mean that many instances of the same message can be _en route_ to a system at the same time and we wish to guarantee a bound on the number of attainable states, we only record whether 0,1, or "many" copies of the same message has been transmitted. We can use this mechanism to provide a source for the messages received in Buffer on the channel "new" by postulating that in the state for the system complex, "many" copies of the message "new α" have been transmitted to "new".

B. System Complex Processes

For the initial state of the complex, Buffer has SB_1 as its system state and "many" copies of message M_3 have been transmitted to it. Consumer has SC_1 as its system state and no pending messages. If Buffer completes a step before Consumer, the "next" state of the complex will have either SB_1 or SB_2 as the state set of Buffer, and either 0, 1, or "many" copies of M_3 still pending for Buffer. SC_1 still is the system state for Consumer. On the other hand, if Consumer completes a step first, then the system complex remains in the initial state.

We summarize the attainable states of the complex in Table 3 and the successor relations between these states in Figure 4.

TABLE 3: Obtainable States of System Complex.
The example system complex is started in state 1 and only the given states may occur.

State Number	State of Buffer		State of Consumer	
	System state	Messages pending	System state	Messages pending
1	SB_1	many M_3	SC_1	none
2	SB_2	many M_3	SC_1	none
3	SB_3	many M_3	SC_1	M_2 and M_4
4	SB_3	many M_3	SC_2	none
5	SB_3	M_1 and many M_3	SC_1	none
6	SB_1	M_3	SC_1	none
7	SB_2	M_3	SC_1	none
8	SB_3	M_3	SC_1	none
9	SB_3	M_3	SC_2	M_2 and M_4
10	SB_3	M_1 and M_3	SC_1	none
11	SB_1	none	SC_1	none
12	SB_2	none	SC_1	none
13	SB_3	none	SC_1	none
14	SB_3	none	SC_2	M_2 and M_4
15	SB_3	M_3	SC_1	none

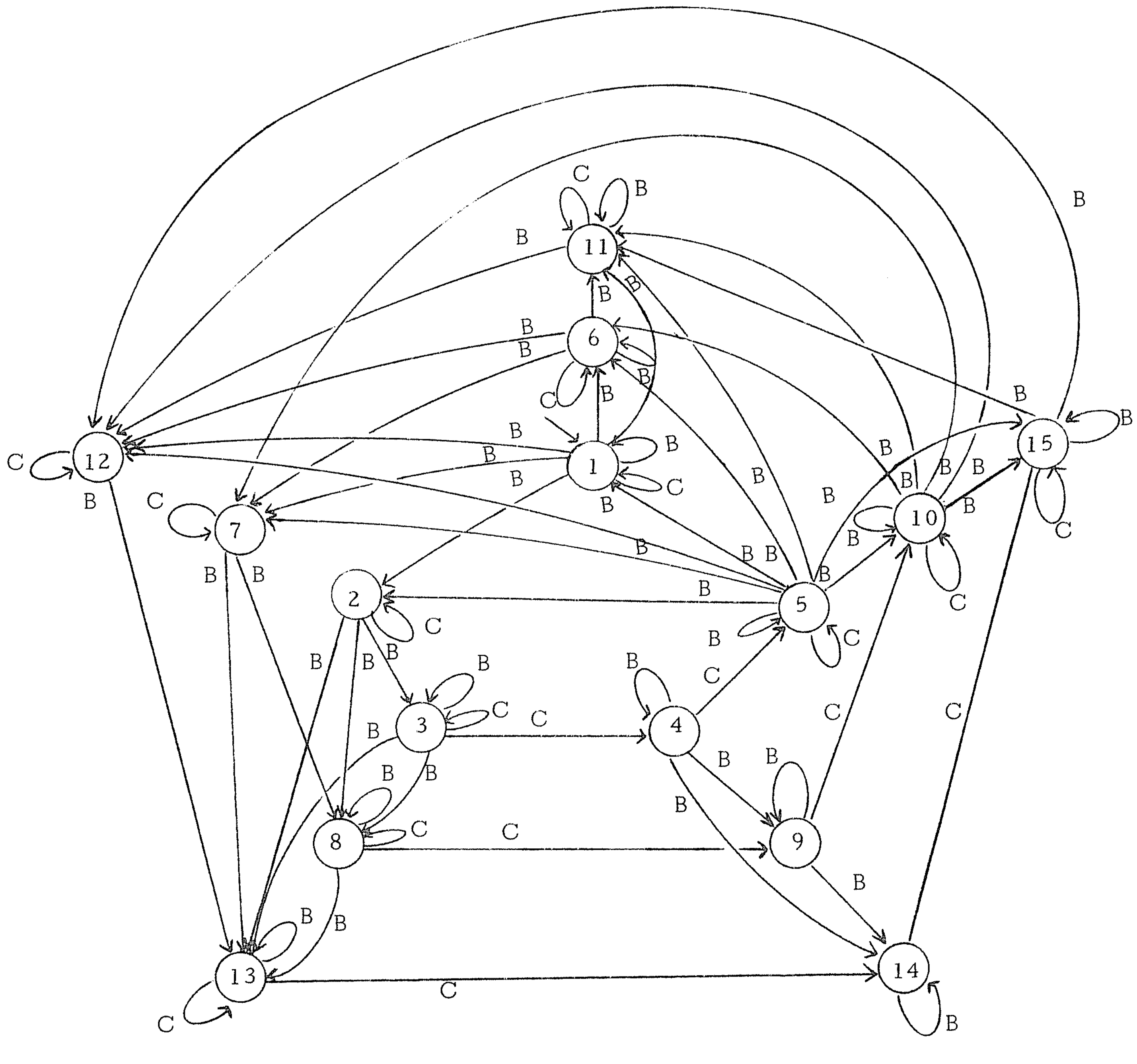

FIGURE 4: State successor relationships for example system complex. A label on an arc means that if the system of that "name" ("B" for Buffer, "C" for Consumer) completes a step the complex can be in the state of the end of the arc.

C. Projection of System Complex Processes

We can obtain quite a bit of information about the system from the table and the graph, but for some purposes they are still too complex. For example, quite a bit of the complexity of the graph seems to depend only on the number of copies of M_3 in the pending message set of Buffer. If we take a suggestion from Horning and Randell [3] and perform a projection on the states to "forget" all information about M_3, we get an induced graph homomorphism on the state-successor graph. The resulting state structures and image graph are in Table 4 and Figure 5, respectively. The projected graph that is orthogonal to the graph in Figure 5 is given in Figure 6.

TABLE 4: States in projected complex.

	State of Buffer		State of Consumer	
State Name	System state	Messages pending	System state	Messages pending
{1,6,11}	SB_1	none	SC_1	none
{2,7,12}	SB_2	none	SC_1	none
{3,8,13}	SB_3	none	SC_1	M_2, M_4
{4,9,14}	SB_3	none	SC_2	none
{5,10,15}	SB_3	M_1	SC_1	none

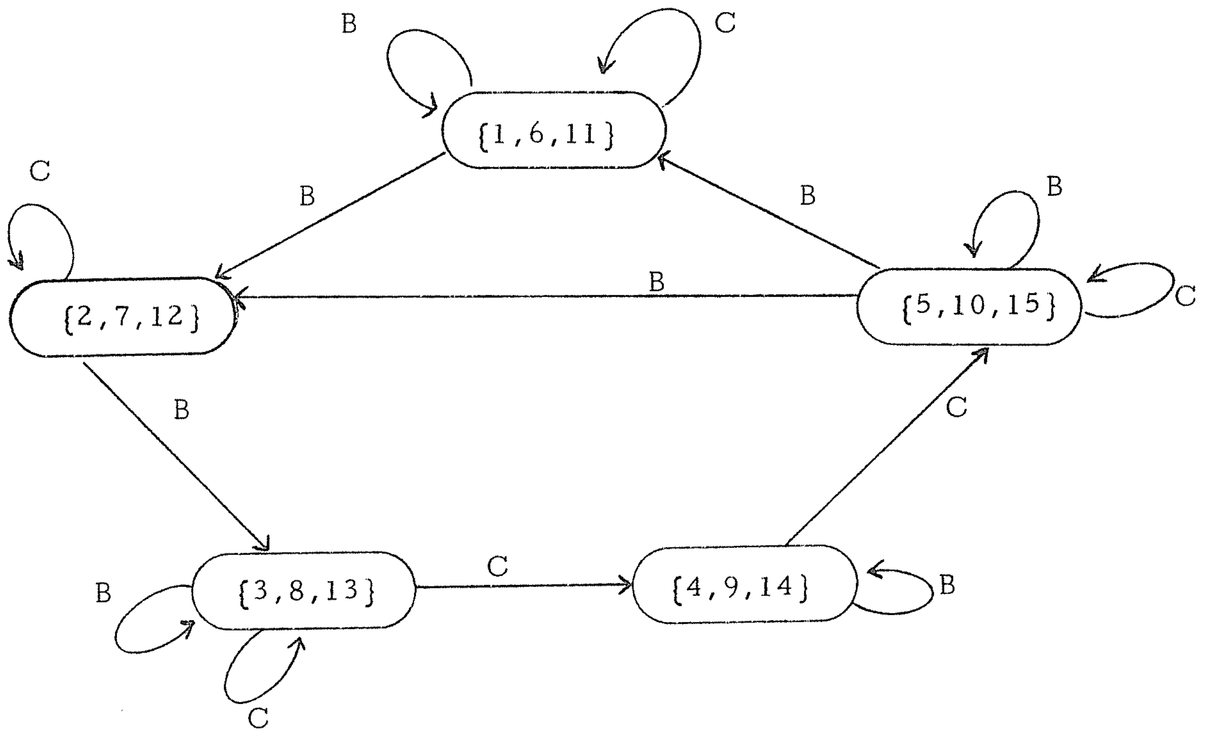

FIGURE 5: Projected image graph of state successors. All dependence on M_3 has been eliminated in this projection of the graph in Figure 4.

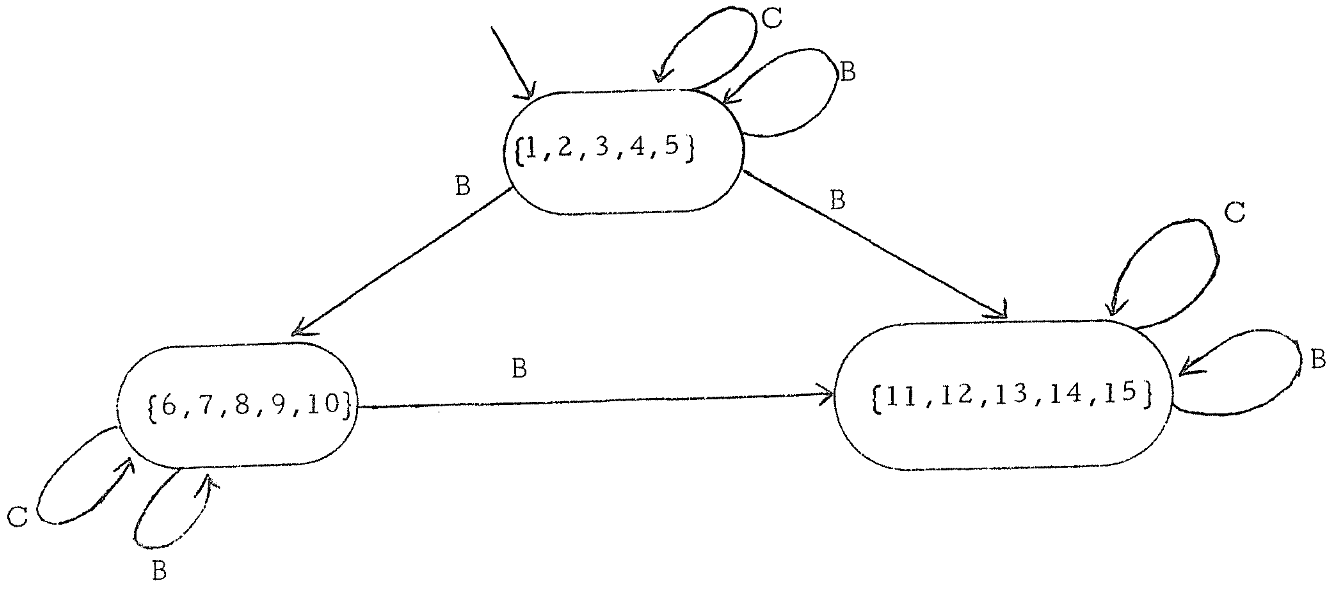

FIGURE 6: Orthogonal projection image graph.
The projection of the graph in Figure 4 which preserves only information about M_3

We note that, using the terminology of Horning and Randell, the process described by the graph of Figure 4 is the synchronous combination of these latter two graphs. Thus, all the information inherent in Figure 4 is in the other two graphs in a more compact and comprehensible form.

D. Verification of Properties

The closure of the image process and system complex states under the transformations of the system complex have been demonstrated in Figure 4. That there is never more than one character in transit from Buffer to Consumer at any time is obvious by inspection of Table 3. The state transitions in Figure 4 are deterministic in spite of the relative rates of the two systems. The role of the signals r:0 and r:1 is clearly shown in the projection of Figure 5.

In Figure 6, there is a transition from the node representing many copies of M_3 ({1,2,3,4,5}) to the node representing no copies of M_3 ({11,12,13,14,15}). If the transmission of M_3 was sequential then this means that some of these messages can get lost because the local reception buffer will only retain one copy during a step. Thus, if we do not want any messages lost, Buffer's steps must be fast enough so that no two of the messages described by M_3 can arrive during a single step.

IV. SUMMARY AND CONCLUSIONS

As we mentioned, the example we presented was simple. At least part of that simplicity arose because we designed it with the subsequent regular-language analysis in mind. In fact, one of the major goals of this continuing research is to provide feedback to designers which can provide a basis for design decisions leading to analyzable systems.

Given the finite graph of states of the system complex and their successor relations, there are many possible analyses which can be performed. The main point of this paper is that, given a system complex and a classification of the process states and messages, such a graph can be generated automatically and algorithmically. We have skipped over many of the details required for being convincing about the validity of the graph as a representation of the processes in the system complex. Some of these raise issues which are quite complex , for example, those associated with computing restricted processor results when the state sets are regular languages and those associated with message handling. Also, we have stated here rather glibly that there is a well-defined relation between the computations in which the state sets are finite sets of strings and those in which the state sets are regular languages. In [4], we have clarified and settled most of these issues.

There are computer implementations of simulators both of the finite language model and the regular language model. Both are written in FORTRAN V on the Univac 1108 and are available from the authors.

REFERENCES

[1] Fitzwater, D. R., and Smith, Pamela Z. "A Formal Definition Universe for Complexes of Interacting Digital Systems." Computer Sciences Technical Report #184, University of Wisconsin, Madison, Wisconsin, 1973.

[2] Gilbert, P., and Chandler, W. J. "Interference Between Communicating Parallel Processes." Comm. ACM 15, 6 (June 1972), pp. 427-437.

[3] Horning, J J., and Randell, B. "Process Structuring." Computing Surveys 5, 1 (March 1973), pp. 5-30.

[4] Johnson, Robert T. Proving Assertions about the State Structure of Formally-defined, Interacting, Digital Systems. Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1973.

BIBLIOGRAPHIC DATA SHEET	1. Report No. WIS-CS-193-73	2.	3. Recipient's Accession No.

4. Title and Subtitle Verification of Process Structures of Interacting Digital Systems	5. Report Date August 1973
	6.
7. Author(s) R. T. Johnson and D R. Fitzwater	8. Performing Organization Rept. No. 193
9. Performing Organization Name and Address The University of Wisconsin, Computer Sciences Department, 1210 W. Dayton St.	10. Project/Task/Work Unit No.
	11. Contract/Grant No.
12. Sponsoring Organization Name and Address	13. Type of Report & Period Covered
	14.

15. Supplementary Notes

16. Abstracts

A formal universe for systems design has been developed, in which representations of interacting digital systems are interpreted by a deterministic automaton acting on state information in the form of character strings. Since process structures in this universe can be described with regular languages, a new interpretation is defined in which regular languages represent state information. Computations under this new interpretation contain the previous ones (with some loss of detail), making it possible to prove assertions about the original string computations. An example is given to show the formulation, and algorithmic verification, of some interesting properties of two asynchronous communicating systems.

17. Key Words and Document Analysis. 17a. Descriptors

Digital Systems
Formal Systems
Language Semantics
Asynchronous Communication
Design Automation
Process Structuring
Extensions to Post Systems
Design Debugging Aids

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 23
	20. Security Class (This Page UNCLASSIFIED	22. Price