SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 847

JUNE 1989

A NOTE ON A FORMAL DEVELOPMENT OF EVENT GRAPHS AS AN AID TO STRUCTURED AND EFFICIENT SIMULATION PROGRAMS

by

Lee W. Schruben
and
Enver Yucesan

This work was sponsored in part by National Science Foundation Grant ECS 8810517.

ABSTRACT

The event graph is a graphical technique for modeling and simulating discrete event systems. Recently, Som and Sargent addressed the issues of *(i)* anticipating logic errors due to simultaneously scheduled events and *(ii)* identifying and eliminating any unnecessary events using event graphs. They presented an elaborate algorithm to perform these tasks. In this note, we point out several shortcomings of their algorithm. With a simple example, we demonstrate that their algorithm can generate an infinite amount of computer code.

1. INTRODUCTION AND BACKGROUND

Event graphs are directed graphs with temporal and logical attributes. They were introduced by Schruben (1983) as a graphical technique to construct and analyze event-scheduling discrete event simulation models. On this graph, events are represented as vertices. Each vertex is associated with a set of changes to the state of the model. Relationships between events are represented as directed edges between pairs of vertices. Each edge is associated with a set of logical and temporal expressions. In other words, the edges determine under what conditions and after how long of a time delay one event will schedule or cancel further events.

Event graphs are also useful in analyzing simulation models. In particular, the analysis of an event graph can aid in the following simulation modeling tasks:

(i) identifying needed state variables,

(ii) determining a minimal set of events that must be scheduled at model initiation,

(iii) anticipating logic errors due to simultaneously scheduled events, and

(iv) eliminating unnecessary event routines.

Schruben offers "rules of thumb" to assist the modeler in the above tasks. Recently, Som and Sargent (1989) introduced a formal framework for event graphs and presented an elaborate procedure to address the last two issues listed above. However, the procedure has certain shortcomings which may result in an infinite expanded event

graph while trying to eliminate unnecessary event routines. These are discussed in the next section.

2. POTENTIAL PROBLEMS

2.1 Execution Order Priority Matrix

Two events are *order independent* if the resulting state of the model is independent of the *sequence* in which the events are executed. Overstreet (1982) shows that order dependencies is an *undecidable* problem. That is, there exists no general algorithm that would determine whether any two events are order independent; the question can only be answered on a case by case basis. Hence, Rule 3 of Schruben and Theorems 1 and 2 of Som and Sargent are the only available assistance for detecting such dependencies. Moreover, they represent sufficient but not necessary conditions.

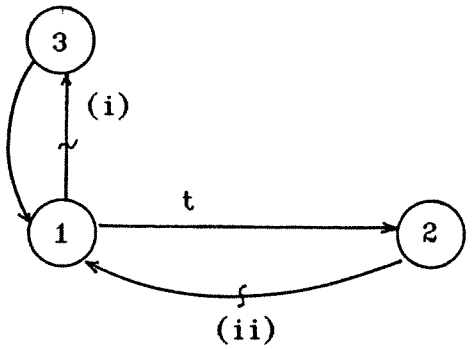
In the event execution order priority matrix of Som and Sargent, an entry of "*y*" in the i^{th} row and j^{th} column "means events i and j may occur simultaneously and the condition of Theorem 2 (event interaction) is not satisfied; however, they do not interact. Therefore, an execution order priority is not required, and events i and j may be executed in either order." As order dependencies is an unsolvable problem, it is impossible to determine whether such a relationship exists between events i and j. Thus, some of the entries of the priority matrix, which is a crucial input to their algorithm, remain undefined.

In a recent version of the computer implementation of event graph modeling, event execution priorities are computed *dynamically* as the

model is run [Schruben and Briskman, 1988]. This scheme appears to be completely general.

2.2 A Procedure for Identifying Super Events

Using Som and Sargent (SS) algorithm, it is possible to generate an infinite expanded event graph while trying to eliminate unnecessary event routines. Consider, for instance, the following event graph:



Suppose that the associated event execution priority matrix is as follows:

	1	2	3
1	n		
2	-1	n	
3	n	-1	n

Next, we apply the (SS) algorithm. In step 1, the root events in the expanded graph are determined. These are:

Primary Root Set = { 2 },

Root Set = { 2 }.

In step 2, the vertex set of the subgraph associated with each root vertex is constructed. The algorithm yields:

$$SG_2 = \{\ 2,\ 1,\ 3,\ 1,\ 3,\ \ldots.\ \}.$$

This is an infinite set of vertices! Moreover, property g of expanded graphs (which states: "in a subgraph, two event vertices cannot have the same name, i.e., no event vertex can have more than one copy) is violated. This, in turn, invalidates Theorem 5.

2.3 Rule 4(a) of Schruben

The Som and Sargent event reduction algorithm is essentially an attempt to automate Schruben's Rule 4(a) together with the given priority scheme. Furthermore, since the algorithm ignores the possibilities described by Rules 4(b) and 4(c) of Schruben, it fails to exploit *all* possible event reduction opportunities.

To show the parallelism between Schruben's Rule 4(a) and Som and Sargent algorithm, both procedures are applied to a single server queueing model. In the event graph, presented in Figure 1, there are three events: the arrival event, A, simply increases the queue size (Q = Q + 1). If the server is "available" (S = 1), then the begin_service event, B, is executed. This, in turn, decrements the queue size and changes the status of the server to "unavailable" (Q = Q - 1, S = 0). The end_service event, E, updates the status of the server back to "available" (S = 1). The customer interarrival times are represented by t_a while the service times are given by t_s. The associated event execution order priority matrix is given by:

	A	B	E
A	n		
B	1	n	
E	1	-1	n

The same expanded event graph is obtained when each procedure is applied to the single server queueing model. The expanded graph is presented in Figure 2.

Straightforward application of Schruben's rules of thumb to the example (interactive computer system model) in Som and Sargent's paper resulted in a SLAM II [Pritsker, 1984] model with 25% less executable code than the model generated by the (SS) algorithm. The execution speed of the two reduced models is not significantly different, even though both (reduced) models executed roughly 50% faster than the original one. The former model is highlighted in the appendix.

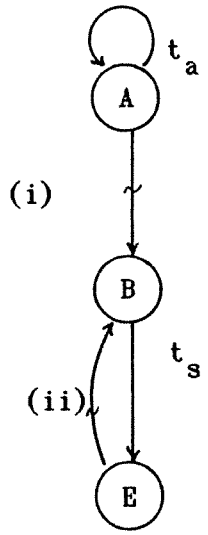


Figure 1. Single Server Queueing Model

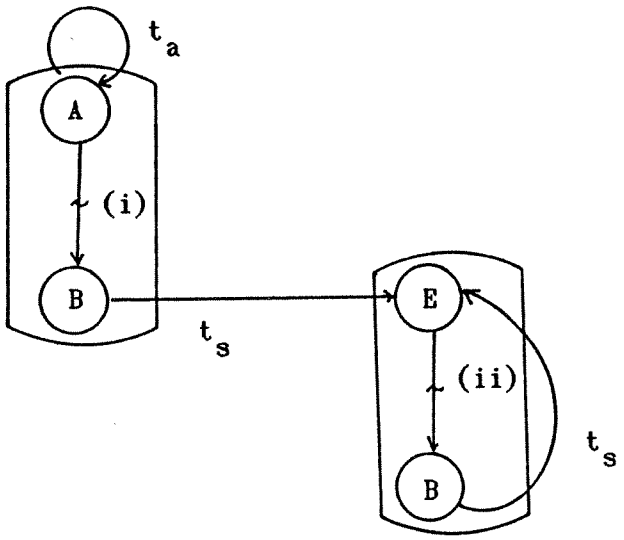


Figure 2. The Expanded Event Graph

3. CONCLUDING REMARKS

Even though Som and Sargent algorithm seems to be the automation of Schruben's Rule 4(a), it has several shortcomings. For instance, certain entries in the execution order priority matrix are not defined. Furthermore, step 2 of the expansion algorithm may generate an infinite event graph, hence an infinite computer code.

Clearly, more work is needed before the "rules" in Schruben's original paper can be called theorems. The application of these rules seems to be quite easy; however, as demonstrated by Som and Sargent (1989), general algorithms for automatic event reduction must be developed carefully.

APPENDIX: THE EVENT GRAPH USING SCHRUBEN'S RULES

Schruben's rules of thumb are applied to reduce the interactive computer model of Som and Sargent (1989). The resulting event graph is presented in Figure 3 and the event descriptions are given below. The definitions of the state variables can be found in the original paper. The edge conditions are:

(i) NM < Z

(ii) R = 1

(iii) (R = 1) & (QM > 0)

(iv) (R = 2) & (SC = 1)

(v) SD = 1

(vi) SC = 1

(vii) QC > 0

(viii) QD > 0

Event 3'

```
NT = NT + 1
NJ = NJ + 1
QM = QM + 1
IF ( NM < Z ) SCHEDULE event 6' AT TNOW
```

Event 6'

```
QM = QM - 1
NM = NM + 1
IF ( SC = 1 ) THEN
   SC = 0
   SCHEDULE event 10' at TNOW+t_c
ELSE
   QC = QC + 1
   ENDIF
```

Event 10'

```
IF ( QC > 0 ) THEN
   QC = QC - 1
   SCHEDULE event 10' AT TNOW + t_c
ELSE
   SC = 1
ENDIF
IF ( SD = 1 ) THEN
   SD = 0
   SCHEDULE event 13' AT TNOW + t_d
ELSE
   QD = QD + 1
ENDIF
```

Event 13'

```
IF ( QD > 0 ) THEN
   QD = QD - 1
   SCHEDULE event 13' AT TNOW + t_d
ELSE
   SD = 1
ENDIF
GENERATE R
IF ( R = 1 ) THEN
   NJ = NJ - 1
   NM = NM - 1
   TNJ = TNJ + 1
   IF ( TNJ ≥ 15000 ) STOP
   IF ( QM > 0 ) SCHEDULE event 6' AT TNOW
   NT = NT + 1
   SCHEDULE event 3' AT TNOW + t_t
ELSE
   IF ( SC = 1 ) THEN
      SC = 0
      SCHEDULE event 10' AT TNOW + t_c
   ELSE
      QC = QC + 1
   ENDIF
ENDIF
```

REFERENCES

[1] Overstreet, C.M. (1982) *Model Specification and Analysis for Discrete Event Simulations*. PhD Dissertation. Virginia Polytechnical Institute and State University. Blacksburg, VA.

[2] Pritsker, A.A.B. (1984) *Introduction to Simulation and SLAM II* John Wiley and Sons. New York

[3] Schruben, L.W. (1983) *Simulation Modeling With Event Graphs* Communications of the ACM. Vol. 29.11 pp.957-963

[4] Schruben, L.W. and Briskman, D. (1988) *Teaching Simulation with Σ* in the Proceedings of the 1988 Winter Simulation Conference. San Diego, CA (Abrams, Haigh and Comfort, eds.) pp.869-874

[5] Som, T.K. and Sargent, R.G. (1989) *A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs* ORSA Journal on Computing Vol.1.2 pp.107-125

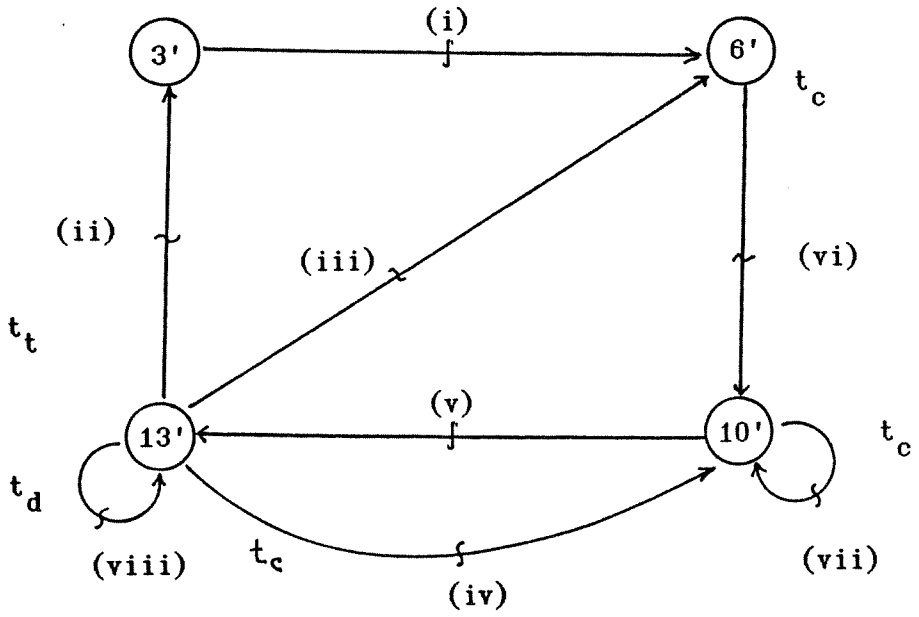


Figure 3. The Reduced Model