

IMPLEMENTATION OF SENSITIVITY ANALYSIS

IN

DISCRETE EVENT DIGITAL SIMULATION

by

Yasemin Birgil

B.S. in I.E., Boğaziçi University, 1981

Bogazici University Library



39001100315848

14

Submitted to the Institute for Graduate Studies in  
Science and Engineering in partial fulfillment of  
the requirements for the degree of  
Master of Science  
in  
Industrial Engineering



178218

Boğaziçi University

1983

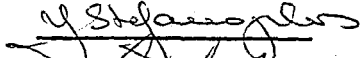
We hereby recommend that the thesis entitled  
"Implementation of Sensitivity Analysis in Discrete  
Event Digital Simulation" submitted by Yasemin Birgil  
be accepted in partial fulfillment of the requirements  
for the degree of Master of Science in Industrial  
Engineering, Institute of Science and Engineering,  
Boğaziçi University.

Examining Committee

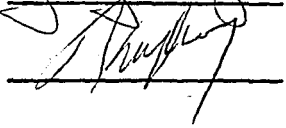
Dr. M. Akif Eyler (Thesis Advisor)

  
\_\_\_\_\_

Doç. Dr. Yorgo Istefanopulos

  
\_\_\_\_\_

Dr. Ali Rıza Kaylan

  
\_\_\_\_\_

Date: 23/6/1983

## TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
ÖZET	vii
LIST OF FIGURES	viii
CHAPTER I. INTRODUCTION	1
CHAPTER II. MODELING DISCRETE EVENT DYNAMIC SYSTEMS USING ACTIVITY CYCLE DIAGRAMS	6
1. Activity Cycle Approach	7
2. Activity Cycle Diagram of a Restaurant, IKILER	12
CHAPTER III. ACSIM: A SIMULATION PROGRAM BASED ON ACTIVITY CYCLES	16
1. Idea of Structured Programming	16
2. Structure of ACSIM	19
3. Simulation of IKILER using ACSIM	24
CHAPTER IV. ESTIMATION OF PARAMETER SENSITIVITIES USING ACSIM	27
1. Sensitivity Calculation for Systems Represented by Activity Cycles	29
2. Addition of Routines into ACSIM to Estimate Parameter Sensitivities	35
3. Experimentation with IKILER	37

CHAPTER V.	IMPLEMENTATION OF SAMPLE PATH ANALYSIS ON A TIME-SHARED COMPUTER SYSTEM	39
	1. Statement of the Problem	39
	2. Modeling the System	41
	3. Experimental Results	44
CHAPTER VI.	DISCUSSION	47
REFERENCES		49
APPENDIX A	DYNAMICS OF ACSIM	51
APPENDIX B	SYNTAX DIAGRAM FOR ACSIM	55
APPENDIX C	ACSIM PROGRAM LISTING	58

## ACKNOWLEDGEMENTS

This study was conducted under the supervision of Dr. M. Akif Eyler to whom I wish to express my sincere thanks for his invaluable guidance, encouragement and support.

I am also grateful to Doç. Dr. Yorgo Istefanopulos and Dr. Ali Rıza Kaylan for serving on my thesis committee.

In particular, I would like to thank Mr. Armağan Birgil for his support in typing and preparation of the form of this document.

Y.B.

## ABSTRACT

Discrete event dynamic systems have been analyzed by use of either analytical or simulation models. Both approaches have certain drawbacks. In this thesis a new approach which aims to reduce the number of computer runs required for the sensitivity analysis in discrete event digital simulations is studied.

The proposed approach estimates the sensitivity coefficients of the system output with respect to various parameters from the results of a SINGLE simulation run. It can derive answer to the question "How would the total simulation time change if we repeated the experiment under the same conditions, except for small perturbations in the sample durations of one of the activities?" for some or all of the activities in the system while observing the experiment. The efficiency of this approach lies in the fact that it has a computational advantage of  $M$  to 1, where  $M$  is the total number of activities for which the above question is asked.

## ÖZET

Ayrık olaylı dinamik sistemlerin çözümlemesinde analitik modeller veya benzetim modelleri kullanılmaktadır. Bu tezde ayrık olaylı sistemlerin duyarlılık çözümlemesinde benzetim zamanını önemli ölçüde azaltan yeni bir yaklaşım sunulmuştur.

Önerilen yaklaşım, sistem çıktısının sistemdeki bazı veya tüm parametrelere göre duyarlılığını tek bir benzetim koşumu ile kestirmektedir. Diğer bir deyişle, "Sistemde yalnız bir faaliyet süresinde küçük bir değişiklik yaparak bu deneyi tekrarlasaydık sistem çıktısı nasıl değişirdi?" sorusuna deneyi gerçekleştirerek görmeye gerek kalmadan sistemi gözleme zamanı içinde cevap verir. Bu yaklaşımın verimliliği bu soruyu tek bir benzetim koşumu içinde sistemdeki tüm faaliyetler için ayrı ayrı cevaplayabilmesindedir. Yani  $M$  adet faaliyet için duyarlılık çözümlemesi istendiğinde hesaplama zamanı bu yaklaşımla  $M:1$  oranında azalacaktır.

## LIST OF FIGURES

	<u>Page</u>
Figure 1.1. Outline of the thesis	3
Figure 2.1. Activity Cycle for a new born baby	10
Figure 2.2. Activity Cycle for a mother	10
Figure 2.3. Activity Cycle Diagram for the functioning of a new born baby	10
Figure 2.4. Steps of Activity Scanning	11
Figure 2.5. Activity Cycle Diagram of IKILER	14
Figure 3.1. Flow diagram of ACSIM	20
Figure 3.2. ACSIM input for the simulation of IKILER	25
Figure 3.3. Results of simulation of IKILER, ACSIM output	26
Figure 4.1. Propagation of local gains through an activity	33
Figure 4.2. Results of sensitivity analysis in IKILER	38
Figure 5.1. A time-shared computer system	39
Figure 5.2. Activity Cycle Diagram of the time-shared computer system	42
Figure 5.3. ACSIM input for the estimation of parameter sensitivities in the time-shared computer system	43
Figure 5.4. Comparative results of Sample Path Analysis on the time-shared computer system	45
Figure 5.5. Estimated sensitivity coefficients, ACSIM output of the time-shared computer system	46



## I. INTRODUCTION

Discrete event dynamic systems, such as traffic flow in a canal system, material flow in a production system, messages in a communication network and generally nonstandard queueing networks, evolve according to occurrences of distinct events. All state changes occur only at a countable number of time epochs which are referred as the event times. These events may be an arrival of a boat or a message, shutdown of a production machine, completion of a car-wash service and can be deterministic or stochastic. Although states of the system change due to an event occurrence, some events may not actually result in a change in the state of the system. The complex interactions of discrete events among the entities make it difficult to analyze the discrete event dynamic systems.

In general, two approaches to these problems exist. The first approach is analytical, typically represented by the queueing theory. The major drawbacks of this approach can be listed as follows: a large number of restrictive assumptions must be satisfied for the theory to be valid, the situation may be too complex to build a mathematical formulation, and analytical techniques may not be sufficient for handling the mathematical formulation even if it has been achieved.

The second approach, simulation, is used when analytical techniques are not applicable. Simulation is one of the most widely used techniques in operations research and it represents one of the major tools for brute force analysis of discrete event systems. Simulation models are designed to sample the characteristics of the system they represent by observing the system over time and subsequently gathering and deducing pertinent information. The analyst can experiment with such a system and study its performance while changing the system parameters and decision rules at will

Although a discrete event simulation could conceptually be done by hand calculations, the amount of data to be stored and manipulated for most real world systems make it apparent that discrete event simulations should be done on a digital computer. In addition, simulation of complex systems would require a large amount of computer time. Therefore the main problem with this method is the computational burden, especially in case of complex systems. Also, repeated simulation of such a system over various parameter ranges to see the effect of changes could be very costly or even infeasible. The purpose of this thesis is to outline and implement a new approach which significantly reduces the problem of computational time.

The proposed approach derives the sensitivity coefficients of system output with respect to various parameters from the result of a SINGLE simulation run. In other words, it can derive answer to the question "How would the system output change if we repeated the experiment under exactly the same conditions, except for a small perturbation in sample values of one of the parameters?" while observing the experiment and without having to repeat any experiment for the specified perturbation. The efficiency of this approach lies in the fact that it has a computational advantage of  $M$  to 1 where  $M$  is the total number of parameters for which the above question is asked.

This approach has been implemented in a previous work for buffer storage design in production lines with successful results (1). Later, it was applied to discrete event dynamic systems using activity cycles world view (2). The simulation of the systems in that particular study was carried out using ECSL, an activity oriented simulation program (3). Successful solutions of these problems have led to extensions to other nonstandard queueing networks (4-10), and these have been collected in a technical report (11).

In the remainder of this chapter an outline of the thesis is given with references to the related chapters, a summary of this outline can be seen in Figure 1.1. with the numbers in parenthesis showing respective chapters.

A basic problem faced by all practitioners of simulation is choosing a method for viewing complex systems in a manner that reduces the complexity and enables one to logically analyze the components of the system. There are many ways to approach this problem, including the flow of entities through the system, events that take place in the system, and the activity cycles of the entities in the system. Event is an occurrence which causes a change in a system entity. In the event scheduling approach, emphasis is on the occurrence of individual events, each time the simulation clock is advanced the next event is determined by predetermined instructions (12).

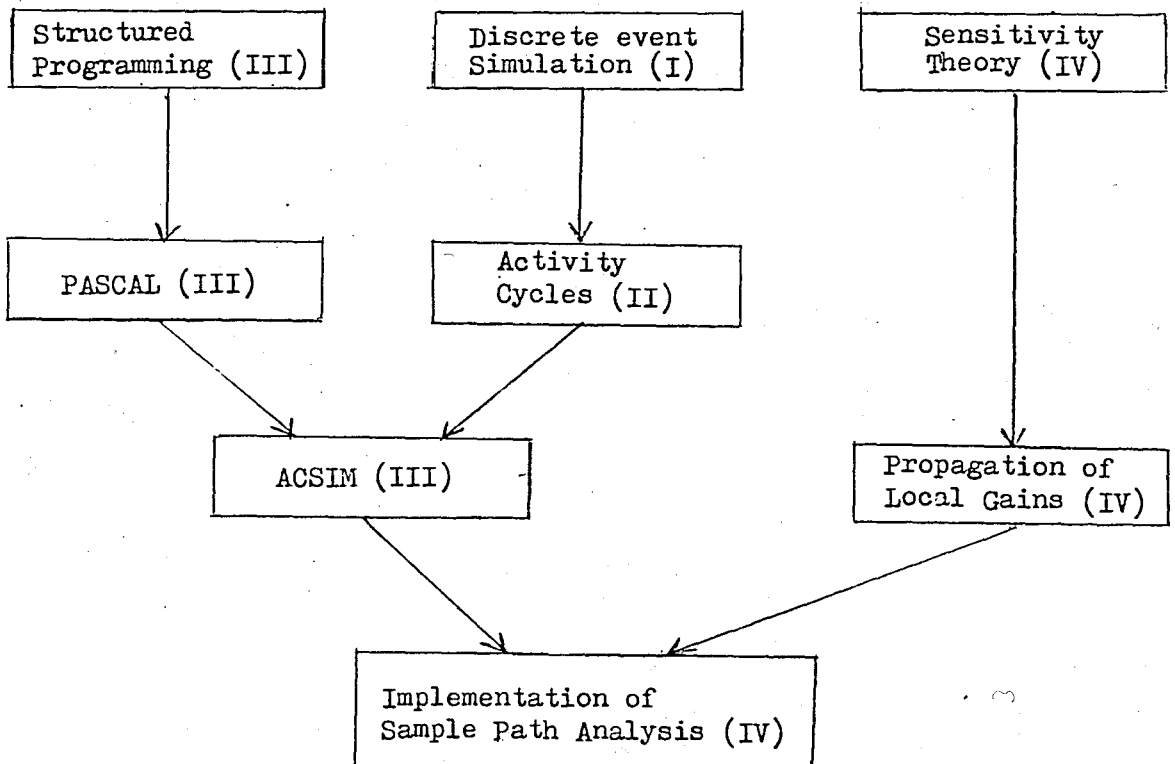


Figure 1.1. Outline of the Thesis

Events are filed in ascending order of the scheduled occurrence times. A significant amount of book keeping effort is required to keep the scheduled event in its proper sequence. To insert a new event in the list, a sequential search is performed until the appropriate location is found. As the simulated events occur, they are removed from the head of the list and the simulated clock is advanced to the smallest event time and event is executed. One of the difficulties of this approach is its division of the logic of an operating system into small parts, for instance the return of a part from a machine is stated three times, one for the part, one for machine and one for the server.

Activity is the active state of an entity where the state of the entity is being changed. It is composed of two parts: testing the conditions, performing the actions. Whenever a simulation time is advanced the next step is determined by a search procedure. A cyclic scanning of activities insures that all possibilities are examined and necessary actions are performed. This method will be used because of its graphic clarity and logical appeal. Chapter II contains a brief explanation of the activity cycle approach and presents the activity cycle diagram of a restaurant system, IKILER.

Process oriented approach carries out the progress of an entity through the system from its arrival until its departure. A process is a set of events or a collection of activities related to an entity. This approach combines the run-time efficiency of event scheduling with the modeling efficiency of activity scanning.

Digital computer is the main computing device for executing a simulation model. In this thesis also lots of work is overcome by use of a digital computer, UNIVAC 1108 Computer in Boğaziçi University, and ACSIM, a simulation program based on activity scanning (13). ACSIM has been designed

by applying the concepts of structured programming and using PASCAL as the programming language. Chapter III discusses the beneficial aspects of the structured programming, describes the structure of ACSIM and presents the results of the simulation of IKILER.

Chapter IV describes the method for sensitivity calculation in discrete event dynamic systems represented by activity cycles. A formal view of the approach is given and the new routines included into ACSIM for this estimation procedure are explained. The method is applied to IKILER and comparative results of this approach and brute force analysis are stated in the last section of the chapter.

A complete example problem, time-shared computer system is studied in Chapter V. First the problem is defined, then the system is modeled using activity cycle diagrams and finally sensitivity coefficients of the total simulation time with respect to durations of each activity in the system are estimated all in one simulation run. The results are compared with the results of brute force analysis and the level of perturbations up to which the deterministic similarity holds are examined.

Chapter VI discusses the use and efficiency of the new approach.

## II. MODELING DISCRETE EVENT DYNAMIC SYSTEMS USING ACTIVITY CYCLE DIAGRAMS

The first task at the start of a simulation study is the choice of a way for viewing and modeling the system of interest. In the United States much of the simulation study has been based on an event-scheduling, SIMSCRIPT, or process-interaction, GPSS, SIMULA, orientation. In contrast, England and Australia have tended to use the activity cycle approach (14).

In the activity cycle approach, the activity is the basic unit composed of two parts: checking the conditions and performing the actions. Whenever a simulation time is advanced, all the activities are scanned for possible performance. If all conditions for an activity are satisfied, state-changing and time-setting instructions are executed. A cyclic scanning of activities insures that all possibilities are examined and all necessary actions are performed. When an activity scan is not employed as is the case in GPSS, SIMSCRIPT, and SIMULA, all system events must be predetermined and scheduled.

An activity scan is efficient for highly interactive processes that involve a fixed number of entities. Event-scheduling is efficient for less interactive processes that involve large number of entities. A process-oriented language reduces the number of statements a programmer has to write, since many event subprograms can be combined in one process routine. Each modeling scheme can be advantageous in some systems and disadvantageous in others. There are no rules for selecting one scheme over another for a given system.

In this thesis activity scanning approach is chosen as the basis for modeling the systems because of its logical appeal and graphic clarity.

This approach has the advantage of viewing complex systems with an apparent ease of communication. Use of internal clocks for each entity, explained in section 2.1., makes it possible to scan the activities without need to use predetermined lists.

## 2.1. Activity Cycle Approach

The basic elements of the activity cycle approach are the entities, the activities, and the queues.

i. **Entities:** A system is considered to be composed of entities which are the elements whose behavior over time will be examined. In a restaurant the entities of interest might be customers, servers, and tables. In a computer system the entities might be central processing unit (CPU), terminals, disks, tapes, and jobs.

Entities of a system may have attributes which describe and distinguish them. A customer in restaurant might have a demand of two or three servings, or a job in a computer system might have a request from disk or tape, and those are the attributes distinguishing them from the others.

It would be useful to group various entities together into classes on some logical basis. A basis for classification could be grouping the entities whose behavior follow identical protocols. For example, servers in a restaurant could be classified as head waiters and waiters, first group taking the orders from the customers and passing them to the waiters, and the second one taking the orders from head waiters and serving the customers.

In an activity cycle, there are basically two groups depending on the state of the entities. "Activities" which are the active states of the entities form the first group, and "Queues" which are the idle states of the entities form the second group. Each entity has an internal

clock showing the time when it will or has become available in its respective queue. The internal clock of an entity is updated when the entity is involved in an activity.

ii. Activities: Entities come together for a certain activity to take place. For example, customer and head waiter would be present for the order activity to start. Activities are considered as the active states of the entities where the states of the entities are being changed. The duration of an activity is determined according to a specified distribution, and all the entities involved in the activity experiences the same duration. Customer and head waiter coming from their respective queues are activated while the order is taken and their internal clocks are reset to the completion time of the activity order.

Conceptually, it is useful to view each active state as having an idle state as its immediate predecessor.

iii. Queues: Queues are the idle states of the entities. Each queue belongs to a single class of entity. An entity class may have one or more queues in an activity cycle diagram. For example, tables in a restaurant might be in two idle states as "full" and "empty" or jobs in a computer system might be "waiting for disk", or "waiting for CPU", or "waiting for tape". In general, an entity in a queue is awaiting the availability of the other entities in their respective queues for the succeeding activity to start. The duration of an idle state for an entity can be zero or anything depending on the other entities coming from different active states, in other words waiting time in a queue changes from an entity to another.

The queues and the activities form one or more closed loops in which the entities circulate. Many of the activities in a system would require more than one class of entity. Therefore preceding queues of such



activities would be more than one. An activity would start if and only if all the input entities are present in their respective queues. If any of these entities is not in its queue, the other entities are forced to remain in queue until that entity becomes available.

When an entity is involved in an activity the internal clock of the entity is advanced to the completion time of that activity and its next state is specified. However, this entity is actually placed in this queue when the system clock or simulation time is equal to its internal clock. System clock is updated after all possible conditions are examined and actions are performed. It is advanced to the smallest internal clock of the active entities. The entities with internal clock values less than or equal to the system clock are in their idle states, waiting in queues, and the entities with internal clock greater than or equal to the system clock are in their active states.

Activity cycle diagrams present the complete logic for the operation of the system and summarize the behavior patterns of the entities in a clear graphic form. In such a diagram, activities are denoted by rectangles, the queues by circles. Each entity class has a certain closed path which is distinguished by use of different directed lines.

As an illustration, consider a system may be the simplest and the most common one for all of the human beings. This system, working for the functioning of a new born baby, is composed of two entities, a baby and a mother. At the start of life a baby has basically two activities: to sleep and to suck. Assume that the mother has only one activity, namely to feed her baby. The activity cycle diagrams of the entities in the system is shown in Figure 2.1., and Figure 2.2., that of baby and the mother respectively.

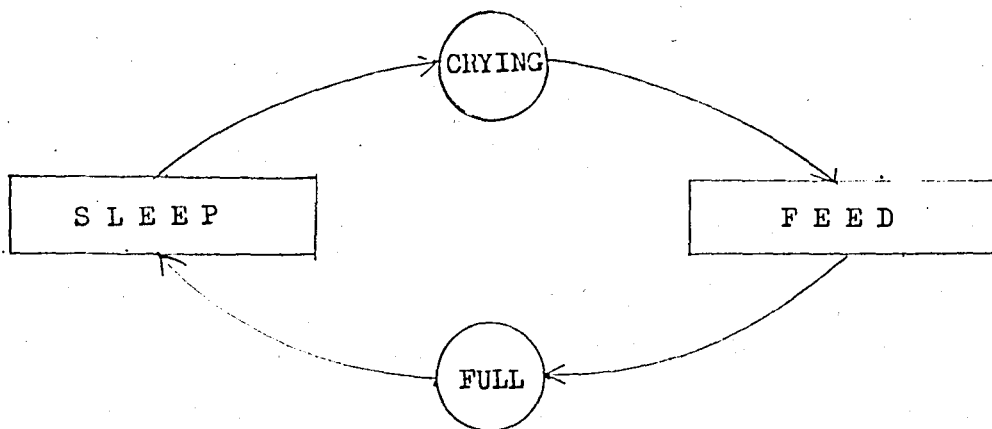


Figure 2.1. Activity Cycle for a new born baby

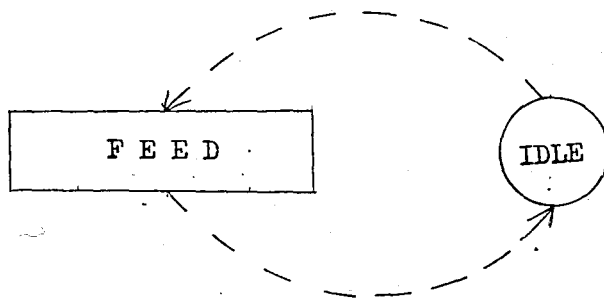


Figure 2.2. Activity Cycle for a mother

Note that the mother and the baby have a common activity FEED. Therefore the baby must be CRYING and the mother must be IDLE for the FEED activity to start. The operation of this system is most easily visualized by combining the activity cycles of the two as in Figure 2.3.

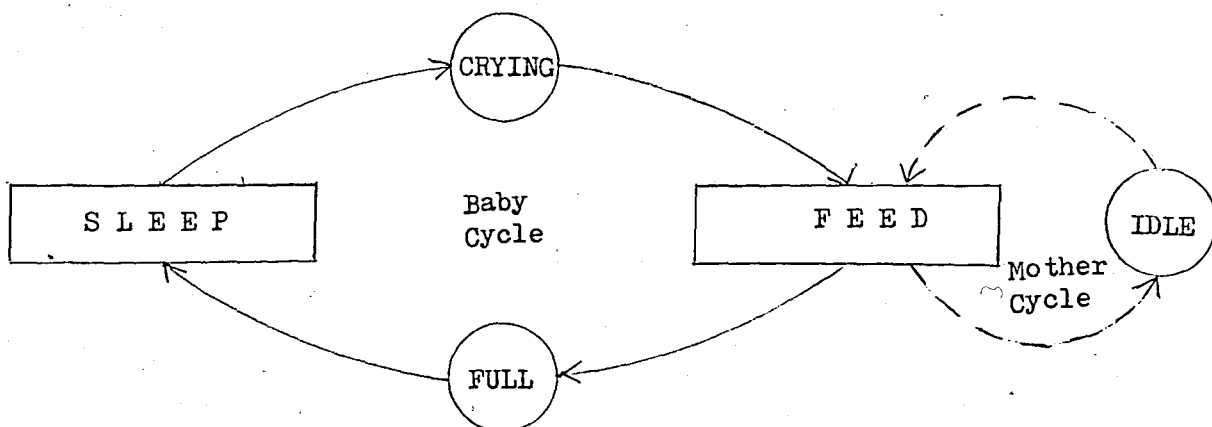


Figure 2.3. Activity Cycle Diagram for the functioning of a new born baby.

Assume that SLEEP and FEED durations are deterministic with 180 and 10 minutes, respectively. The simulation of this simple system by use of activity scanning could be explained as follows. Assume that simulation ends when the baby is fed twice. Initially, the baby is CRYING and the mother is IDLE. The system clock and the internal clocks of the baby and the mother are set equal to zero. Since the conditions for FEED are satisfied, the actions are executed. Namely, next idle states are specified as FULL and IDLE, the internal clocks of the entities are advanced to the end of FEED, now they are both 10. The other activity SLEEP is tested but not started since the baby is not FULL yet. Now all the activities are scanned, it is time to update the system clock. The system clock is set to the smallest internal clock of the active entities. For this case it becomes 10. It is time to move those entities with their internal clocks equal system clock to the specified idle states. Now baby is FULL and the mother is IDLE again.

The activity scanning goes on until the end of simulation. The steps of simulation for this system is presented in Figure 2.4.

System Clock	Internal clock of BABY	Internal clock of MOTHER	State of BABY	State of MOTHER
0	0	0	CRYING	IDLE
0	10	10	FEED	FEED
10	10	10	FULL	IDLE
10	190	10	SLEEP	IDLE
190	190	10	CRYING	IDLE
190	200	200	FEED	FEED
200	200	200	FULL	IDLE

Figure 2.4. Steps of Activity Scanning

It should be noted as an important characteristic that an activity cycle diagram is independent of the number of entities in the system. The diagram in Figure 2.3. could be used for a mother and her baby, or for a mother and her twins, or for two baby-farmers and seven babies sucking at feeding-bottles in a baby-farm.

An activity cycle diagram is also independent of the time required to perform the activities. The same diagram could be used for any entity in a specific class. The diagram in Figure 2.3. could be used for any baby and mother considering that it is independent of SLEEP and FEED durations which might change from baby to baby and from a mother to another.

To complete the diagram for simulation purposes, the number of each type of entity and the distribution for the duration of each activity must be added, and initial states of the entities must be specified. Then, this complete diagram provides the basic input for ACSIM.

## 2.2. Activity Cycle Diagram of a Restaurant, IKILER

A restaurant system, IKILER, is developed as the basic sample problem to clarify the concepts mentioned in this thesis. It will be studied step by step and will be used for the illustrations of the third and fourth chapters, too.

Assume that IKILER is a restaurant having 20 tables, a head waiter, and 3 waiters. The arriving customers enter the system if a table is available, otherwise they leave. The interarrival times of customers are independent identically distributed exponential random variables with mean 10 minutes. The head waiter takes the orders from the customers and passes them to the waiters. The duration of order taking is a uniform random variable between 2 and 4 minutes. The duration it takes to pass the orders to waiters is again a uniform random variable between 1 and

3 minutes. The waiters take the orders from the head waiter and pass them to the kitchen, when the food is ready they serve the customers. The duration for the preparation of food, the cook duration, is a normal random variable with mean 15 and standard deviation 3 minutes. The service duration is a uniform random variable between 3 and 5 minutes. After service completion customers have their dinner. Dinner duration is a normal random variable with mean 40 and standard deviation 8 minutes. At the end of dinner the head waiter receives the payments. The duration it takes for receiving the payments is a uniform random variable between 3 and 7 minutes.

The dynamics of IKILER would be well understood while observing its activity cycle diagram shown in Figure 2.5. The arrival process is generated by an artificial generator, DOOR, which becomes OPEN according to the interarrival distribution of the customers. When the DOOR is OPEN and a CUSTOMER group is OUT, the activity ARRIVE starts and the customer enters the system, and joins the queue PAUSE waiting for the availability of a table. If a TABLE is not available, in other words if there is no table in the idle state EMPTY, CUSTOMER leaves the system. LEAVE is an activity with zero duration. If there is an EMPTY TABLE, the CUSTOMER is involved in the activity SIT with a deterministic duration of one minute together with that table. After the completion of SIT the TABLE is FULL, and the CUSTOMER SITTING, and waiting to give the orders. ORDER activity starts when the HEAD WAITER is IDLE and CUSTOMER is SITTING. At the end of ORDER the HEAD WAITER is IDLE again and the CUSTOMER is WAITING for the next coming activity. ORDERPASS starts when the head waiter is IDLE, a waiter is FREE, and customer is WAITING. After ORDERPASS a dummy queue, INDUML is introduced. This is to declare that ORDERPASS implies the next activity COOK. After COOK, customer is HUNGRY while waiting for SERVICE,

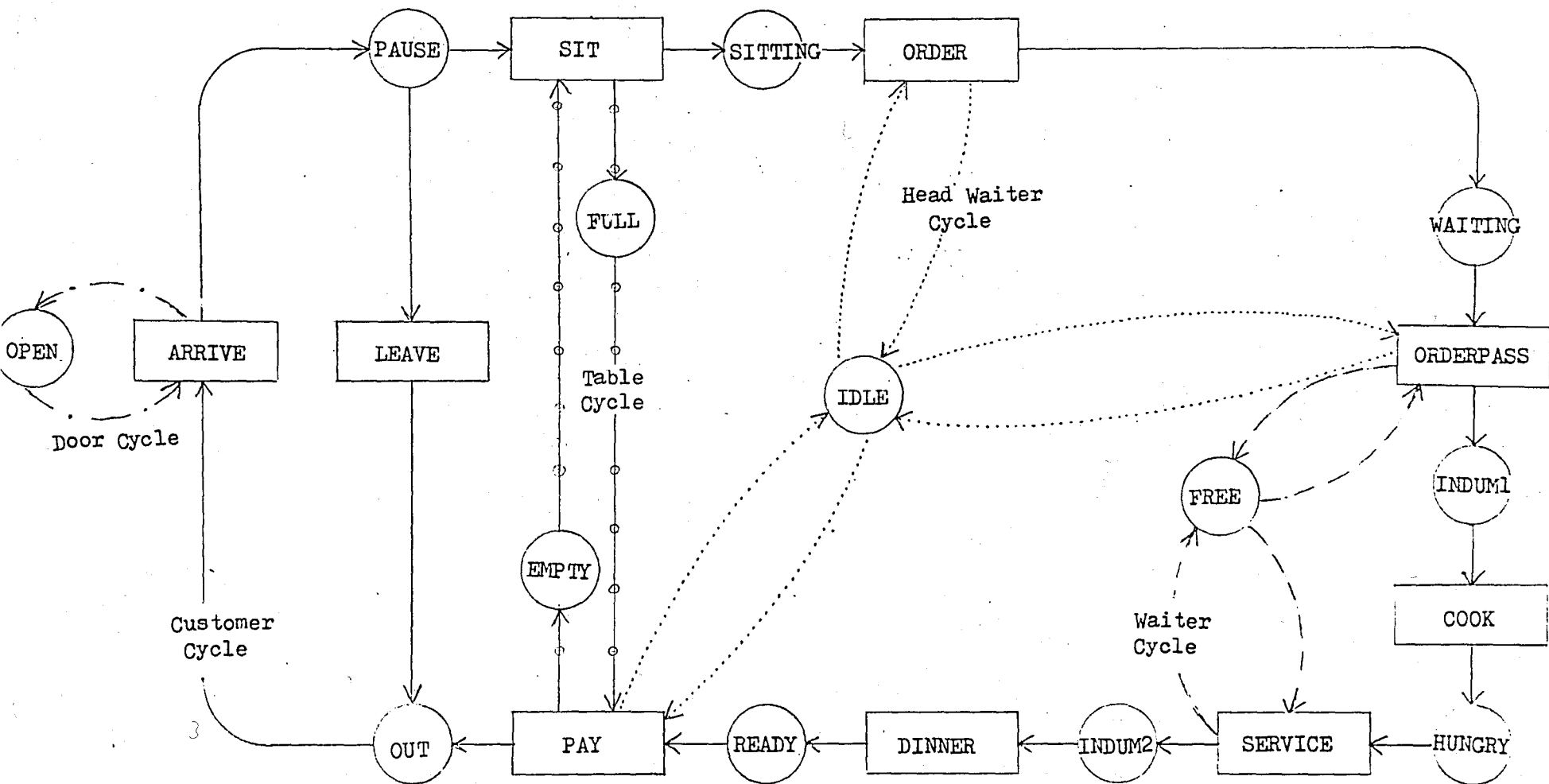


Figure 2.5. Activity Cycle Diagram of IKILER

and the SERVICE starts as soon as a WAITER is FREE. Since the SERVICE implies the activity DINNER, again a dummy idle state INDUM2 is introduced in between. After DINNER customers are READY to PAY, waiting for the IDLE HEAD WAITER. When the activity PAY is completed table becomes EMPTY, head waiter becomes IDLE, and customers go OUT to represent a new group.

Table becomes FULL as soon as the activity SIT is completed and stays in this idle state during the activities ORDER, ORDERPASS, COOK, SERVICE, DINNER, and PAY. Instead of forming such a big cycle for the table and introducing dummy queues between all these activities, a single queue FULL is created and preceded the activity PAY. It would be clear that with this diagram a table would have to wait as FULL during all the activities mentioned above and goes into idle state EMPTY after the completion of PAY.

The elements of the activity cycle diagram could be summarized as follows: There are five entity classes in IKILER, namely HEAD WAITER, WAITER, TABLE, DOOR, and CUSTOMER. The idle states of CUSTOMER are PAUSE, SITTING, WAITING, INDUM1, HUNGRY, INDUM2, READY, and OUT, the idle states of TABLE are FULL and EMPTY. For the entity classes HEAD WAITER, DOOR, and WAITER there are only single queues; IDLE, OPEN, and FREE, respectively. The activities in the system can be listed as ARRIVE, SIT, ORDER, ORDERPASS, COOK, SERVICE, DINNER, PAY, and LEAVE.

### III. ACSIM: A SIMULATION PROGRAM

#### BASED ON ACTIVITY CYCLES

Although a computer is not a necessary tool for carrying out a simulation experiment, it certainly speeds up the process, eliminates the computational difficulties, and reduces the probability of error.

In this thesis lots of work is overcome by use of a digital computer, UNIVAC 1108 Computer in Boğaziçi University. ACSIM, a structured simulation program employing activity scan is used in simulation of the discrete event dynamic systems. The programming language PASCAL has been used in ACSIM.

This chapter includes a brief explanation of the idea of structured programming, gives a summary of ACSIM structure, and presents the results of simulation of IKILER by use of ACSIM.

#### 3.1. Idea of Structured Programming

The earliest computer programs were no more than lists of the primitive instructions that the computer could execute directly. Until the late 1950's programming consisted of the detailed encoding of long sequences of instructions into numbers in binary, octal, hexadecimal form (15). The programmers at this stage had to consider all details of the machine including its processor organization and its instruction set. As time went by, more complicated programs were written and these programs became unmanageable. To a machine the execution of a program containing a few thousands instructions presented no problem. However, it was practically impossible for a programmer to discover the principles of



such a complicated program and even difficult to explain his or her own program. The reason was that these programs lacked structure. The representation of a complex program as an unstructured, linear sequence of commands was the most inappropriate form for the human inspector to comprehend and to express.

These shortcomings led to the development of high level programming languages which were designed not according to the limitations of current technology but according to the habits and the capabilities of man to express his thoughts. In the following years these developments led to an increasing interest in the art of computer programming.

The presence and application of structure became the principle tool in helping the programmer to synthesize systematically and to maintain an overall comprehension of complicated programs. All computer programs could be expressed in terms of four basic structures (16). These are the sequence, the decision, the loop, and the procedure. The sequence is a group of instructions executed one after the other. The decision is a structure that enables the action of the program to be influenced by the data. Many languages introduce the decision structure with the word 'IF', such as:

```
IF a>b THEN c:=a-b
      ELSE c:=b-a
```

The loop structure is used to execute an instruction or a sequence of instructions several times. An example to one of the forms of loop structure can be given as follows:

```
REPEAT
      c:=c+i
UNTIL c>=cmin
```

The procedure enables one to replace a group of instructions by a single instruction. Use of procedures not only makes a computer

program shorter and easier to write but also gives a hierarchical structure. Furthermore it allows a program to be the product of a team-work with each procedure written by different programmers.

The programming language PASCAL uses all of these techniques of structuring. They are incorporated in such a simple way that PASCAL is very easy to learn and use, and also a well-written PASCAL program is easy to understand.

The layout of a program text must match the structure of the program. A PASCAL program is structured in levels, and the level of a statement is indicated by indenting the statements. A simple example would clarify these ideas.

```
PROGRAM addintegers;

    VAR i,sum:integer;

    BEGIN i:=0;
          sum:=0;
          REPEAT
              i:=i+1;
              sum:=sum+i
          UNTIL i=50;
          write(sum)

    END.
```

PASCAL programs start with the key word PROGRAM after which the name of the program is given. VAR is the key word employed before the declaration of the variables used in the program. The main program is stated between the key words BEGIN and END.

ACSIM, the simulation program used in this thesis has been designed considering all these beneficial aspects of structured programming and the language PASCAL in particular.

### 3.2. Structure of ACSIM

ACSIM is a general purpose discrete event digital simulation program based on activity cycles. Once the activity cycle diagram of a system is initiated it becomes almost automated to turn it into an input for ACSIM. ACSIM is good for quick and little-detailed simulations without need to invest much time for modeling and coding.

Structured statements of the main program of ACSIM is given below and the equivalent flow diagram is shown in Figure 3.1.

```

BEGIN
    initialize;
    REPEAT
        move entities;
        scan activities;
        IF none of the activities could start
            THEN update time;
    UNTIL the end of simulation;
    report
END.

```

The dynamics of ACSIM would be quite clear after the refinements to explain the underlined statements above:

```

initialize:
    create the given number of entities, activities, and queues;
    set system clock;
    for each entity
        set internal clock.

```

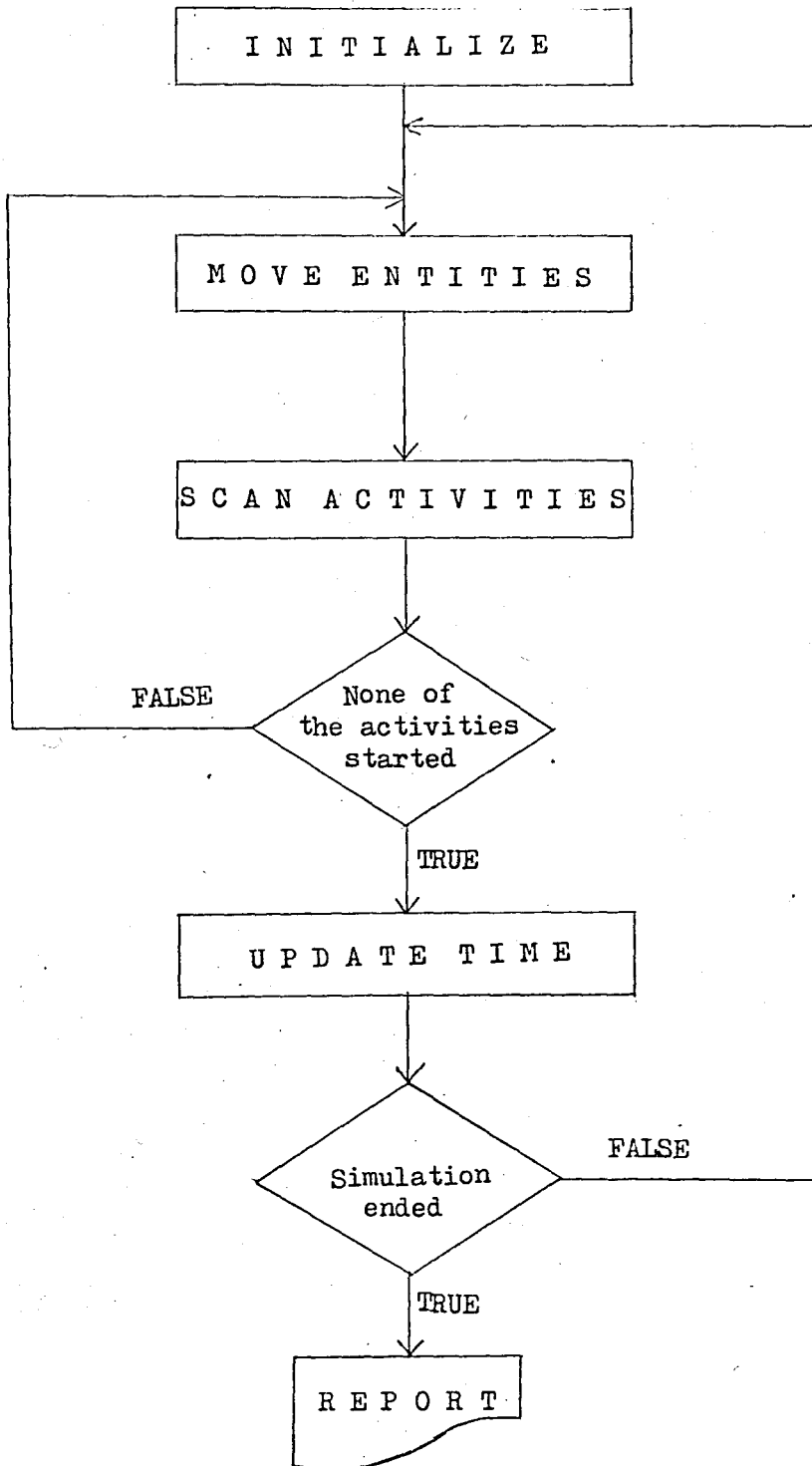


Figure 3.1. Flow diagram of ACSIM

move entities:

```

for each active entity
  if internal clock equals system clock
    then move it into its respective queue.

```

scan activities:

```

for each activity
  begin
    check conditions;
    if all the conditions are satisfied
      then perform actions
  end.

```

update time:

```

advance the system clock to the smallest internal
  clock of the active entities.

```

report:

```

print the system clock;
for each activity
  print the number of times it has started;
for each queue
  print the number of entities in the queue.

```

One more step of refinements is necessary to explain how the conditions are tested and if satisfied which actions are taken.

perform actions:

```

for each entity to be activated
  begin
    take entity from its queue;
    specify its next state;
    if an attribute is specified then set its value;
    update the internal clock of the entity
  end.

```

check conditions:

```

if the number of entities in queue is less than requirement
  then condition is not satisfied
  else if no attribute is specified
    then begin
      condition is satisfied;
      specify the entities to be activated
    end
  else begin
    search for an entity with desired attribute;
    if found
      then begin
        condition is satisfied;
        specify the entities to be activated
      end
    else condition is not satisfied
  end.

```

ACSIM allows the activity durations to be given according to a certain probability distribution functions with integer arguments as well as deterministic durations.

Allowable expressions to specify the durations and corresponding interpretations are as follows :

```

CON(n) : Constant integer with value n,
XPO(t) : Exponential distribution with mean t,
UNI(a,b) : Uniform distribution between a and b,
NOR(m,s) : Normal distribution with mean m and standard
           deviation s.

```

In any of the above distributions, activity durations are generated so that they have integer values.

The format used to convert the activity cycle diagram of a system to an ACSIM input could easily be understood by a simple illustration since

there is an apparent correspondence between the two.

The format as applied to the diagram in Figure 2.3. is shown below:

```

BABYFARM:          143
FEED  BABY CRYING? MOTHER IDLE?
      AFTER UNI(12,18):
      BABY FULL, MOTHER IDLE;
SLEEP BABY FULL?
      AFTER NOR(40,6):
      BABY CRYING;
BEGIN 4 CRYING, 3 FULL, 2 IDLE,
END   FEED 17.

```

The first line identifies the system name and the initial seed to be used in random number generation. Then activities are listed one by one separated by semicolons. Each activity block contains the name of the activity, the conditions to be satisfied for the activity to start, the distribution for the duration of the activity, and the actions to be taken in case the activity starts. Conditions are recognized by question marks. After the keyword AFTER the distribution is specified. Finally the actions are stated and a comma is used as the separator between the actions. After all the activities are listed in this format, the initial states of the entities are given after the key word BEGIN, and the terminating condition of the simulation is given after the key word END.

In the above example the name of the system is Babyfarm and initial seed is 143. Initially, 4 babies are CRYING, 3 babies are FULL, and 2 baby-farmers are IDLE. The first activity is FEED. The conditions, availability of a CRYING baby and an IDLE mother (baby-farmer), must be satisfied for this activity to start. FEED duration is generated according to uniform distribution between 12 and 18 minutes. The actions to be taken

in this activity are to specify the next states for the entities involved and to reset their internal clocks. The internal clocks are set to the completion time of FEED, and next states are specified as FULL and IDLE for the baby and the mother, respectively. The second activity is SLEEP. SLEEP starts as soon as a baby is FULL, and takes time according to the normal distribution with mean 40 and standard deviation 6 minutes. After completion of SLEEP baby is CRYING again. The simulation ends when the 17th FEED activity is completed.

Two extensions to this simple world view have been implemented in ACSIM;

1. Each entity may carry an integer attribute that is set in some actions and inspected in some conditions.
2. An activity may require more than one entity from a queue.  
For example, a part in a job-shop may require two workers to be carried outside the shop.

The program listing of ACSIM is given in Appendix C, and a complete syntax diagram for the ACSIM input is given in Appendix B.

### 3.3. Simulation of IKILER using ACSIM

The queueing system IKILER illustrated in Figure 2.5. is simulated using ACSIM. Figure 3.2. shows the ACSIM input as applied to the activity cycle diagram of IKILER. Since ORDERPASS implies COOK, COOK duration is included in ORDERPASS, and the queue INDUM1 together with activity COOK is eliminated. The same idea applies in the case of SERVICE and DINNER. These activities are combined under the name DINNER. The SERVICE duration is considered in DINNER, and the queue INDUM2 is eliminated. In other



words, instead of

```
SERVICE  CUSTOMER HUNGRY? WAITER FREE?
          AFTER UNI(3,5): CUSTOMER INDUM2, WAITER FREE;

DINNER   CUSTOMER INDUM2?
          AFTER NOR(40,8): CUSTOMER READY;
```

a better format

```
DINNER   CUSTOMER HUNGRY? WAITER FREE?
          AFTER UNI(3,5): WAITER FREE,
          AFTER NOR(40,8): CUSTOMER READY;
```

is used.

Specifications of system parameters are given in section 2.2. Initially, 4 customers are SITTING, 5 customers are WAITING, 11 tables are EMPTY, the head waiter is IDLE, and 3 waiters are FREE? Obviously 9 tables are FULL, occupied by the customers in the system. The door is OPEN to start

```
IKILER:                                     14

ARRIVE  CUSTOMER OUT? DOOR OPEN?
        AFTER 0: CUSTOMER PAUSE,
        AFTER XPO(10): DOOR OPEN;

SIT     CUSTOMER PAUSE? TABLE EMPIY?
        AFTER 1: CUSTOMER SITTING, TABLE FULL;

PAY     CUSTOMER READY? TABLE FULL? HEADWAITER IDLE?
        AFTER UNI(3,7): CUSTOMER OUT, TABLE EMPIY, HEADWAITER IDLE;

DINNER  CUSTOMER HUNGRY? WAITER FREE?
        AFTER UNI(3,5): WAITER FREE,
        AFTER NOR(40,8): CUSTOMER READY;

ORDERPASS CUSTOMER WAITING? WAITER FREE? HEADWAITER IDLE?
        AFTER UNI(1,3): WAITER FREE, HEADWAITER IDLE,
        AFTER NOR(15,3): CUSTOMER HUNGRY;

ORDER   CUSTOMER SITTING? HEADWAITER IDLE?
        AFTER UNI(2,4): CUSTOMER WAITING, HEADWAITER IDLE;

LEAVE   CUSTOMER PAUSE?
        AFTER 0: CUSTOMER OUT;

BEGIN   4 SITTING, 5 WAITING, 20 OUT, 9 FULL, 11 EMPIY,
        1 IDLE, 3 FREE, 1 OPEN,
        PAY 60.

END
```

Figure 3.2. ACSIM input for the simulation of IKILER

the arrival process. Since this is a closed queueing model there would be enough number of customers OUT, ready to enter the system at each opening of the door. Simulation would terminate at the end of 60th PAY.

The system was simulated with the given specifications and initial conditions. The system clock was 630 when the 60th PAY was completed. The simulation output showing the activity counters, The states of the entities and the content of the queues are presented in Figure 3.3.

```

IKILER
SIMULATION ENDED AT PAY          60
SIMULATION ENDED AT CLOCK = 630.
ARRIVE HAS STARTED 60 TIMES
SIT HAS STARTED 60 TIMES
PAY HAS STARTED 60 TIMES
DINNER HAS STARTED 63 TIMES
ORDERPAS HAS STARTED 63 TIMES
ORDER HAS STARTED 59 TIMES
LEAVE HAS STARTED 0 TIMES
    
```

```

ENTITIES AT 630
1 CUSTOMER: 1 PAY          3 DINNER
             19 OUT        5 SITTING      1 WAITING
2 DOOR      : 1 ARRIVE
             NO DOOR      IN ANY QUEUE
3 TABLE    : 1 PAY
             10 EMPTY     9 FULL
4 HEADWAIT: 1 PAY
             NO HEADWAIT IN ANY QUEUE
5 WAITER    : NO ACTIVE WAITER
             3 FREE
    
```

```

QUEUES AT 630
1 CUSTOMER OUT          19: 4 13 18 14 17 20 15 21 16 19 22 24 25 26
2 DOOR OPEN            0:
3 CUSTOMER PAUSE       0:
4 TABLE EMPTY        10: 39 40 41 42 43 44 45 46 47 48
5 CUSTOMER SITTING     5: 10 11 2 3 12
6 TABLE FULL          9: 30 31 32 33 34 35 36 37 38
7 CUSTOMER READY       0:
8 HEADWAIT IDLE        0:
9 CUSTOMER HUNGRY      0:
10 WAITER FREE         3: 53 51 52
11 CUSTOMER WAITING    1: 1
    
```

Figure 3.3. Results of simulation of IKILER, ACSIM output

#### IV. ESTIMATION OF PARAMETER

##### SENSITIVITIES USING ACSIM

As stated in the Introduction, the question "How would the system output change if we repeated the experiment under exactly the same conditions, except for small perturbations in sample values of one of the parameters?" will be answered without having to repeat any experiments at all. Furthermore this question could be answered for all of the system parameters in one run, while simulating the system.

The system output is the total simulation duration. It may be a time value specified beforehand to terminate the simulation, or it may be the time required to complete given number of activations of an activity, the output activity.

The system output,  $T_s$ , is a function of the system parameters, i.e. the activity durations.

$$T_s = f(p_{11}, \dots, p_{1K_1}, \dots, p_{j1}, \dots, p_{jk}, \dots, p_{jK_j}, \dots, p_{M1}, \dots, p_{MK_M})$$

where

$p_{jk}$  = duration of the kth activation of activity j,

$K_j$  = total number of activations of activity j,

$M$  = total number of activities in the system.

If a small perturbation is created in the k-th activation of activity j,  $A_{jk}$ , then the system output in the perturbed system,  $T'_s$ , will be given by:

$$T'_s = f(p_{11}, \dots, p_{j1}, \dots, p_{jk} + \Delta p_{jk}, \dots, p_{jK_j}, \dots, p_{MK_M})$$

where  $\Delta p_{jk}$  is the amount of perturbation introduced in  $A_{jk}$ .

The sensitivity of the system output with respect to the duration  $p_{jk}$

is given by

$$(4.1) \quad \frac{\partial T_s}{\partial p_{jk}} = \lim_{\Delta p_{jk} \rightarrow 0} \frac{f(p_{11}, \dots, p_{jk} + \Delta p_{jk}, \dots, p_{MK_M}) - f(p_{11}, \dots, p_{jk}, \dots, p_{MK_M})}{\Delta p_{jk}}$$

$\frac{\partial T_s}{\partial p_{jk}}$  is called the sensitivity of the system output to the individual duration  $p_{jk}$ .

The sensitivity of the system output with respect to durations of all individual activations of a particular activity  $j$  would be

$$\frac{\partial T_s}{\partial p_{jk}} \quad k=1, \dots, K_j$$

The resultant sensitivity of the total simulation time to the sample durations of activity  $j$ ,  $\frac{\partial T_s}{\partial p_j}$ , would simply be the addition of the sensitivities with respect to individual durations. This is true for all the activities in the system.

$$(4.2) \quad \frac{\partial T_s}{\partial p_j} = \sum_{k=1}^{K_j} \frac{\partial T_s}{\partial p_{jk}} \quad j = 1, \dots, M$$

Here the basic assumption is that individual perturbations are so small not to cause nonlinear effects in the system even when they are applied in all the activations of a particular activity.

The efficiency of this study is due to the fact that the sensitivity coefficients of the system output with respect to all the activity durations can be estimated in a single run. Furthermore, the CPU time it takes to estimate  $M$  sensitivity coefficients is almost the same with the CPU time that brute force analysis takes to estimate only one sensitivity coefficient.

This estimation procedure is done by use of Perturbation Propagation Analysis. Perturbation Analysis can be carried out for a number of performance measures of interest. In this study, the sensitivity

coefficients of the total simulation time with respect to the durations of some or all of the activities in discrete event dynamic systems are analyzed.

Propagation of a perturbation refers to the way in which the change in activity durations can propagate through the discrete event system, affect various activities, and eventually cause a change in the total simulation duration.

In section 4.1. first a descriptive then a formal view of perturbation analysis for systems represented by activity cycles are presented. The procedures included into ACSIM to estimate the sensitivity coefficients are explained in section 4.2. The sensitivity of total simulation time with respect to the durations of the activities in IKILLER are estimated using ACSIM and comparative results are also included in the last section of the chapter.

#### 4.1. Sensitivity Calculation for Systems Represented by Activity Cycles

Suppose that a perturbation is introduced in some of the system parameters. This will cause to put on gain or loss to the entities involved. For example, if an activity is finished one unit early then the entities involved in this activity will be available in the succeeding queues one unit early. As a result they all will have a local gain of one unit. Such a perturbation will be eventually propagated through the other activities or will be cancelled, namely either realized or lost by the system. Once a perturbation is introduced in the system its propagation to the other entities is followed via the rules which are called the Propagation Rules (2). They could be stated as follows:

- i. Gain of an entity is propagated either partially or as a whole to all other entities involved in an activity if this entity arrives last, namely if it is the critical entity for this particular activation of the activity.
- ii. Gain of an entity could only be affected at points in time when an activity in which the entity is involved occurs.

Clearly, if local gains are not eliminated through the system this would be referred as realization of gain for the entire system as opposed to the realization of this gain for a single activity.

The Perturbation Analysis approach could be formalized as follows. First of all it should be noted that the system without any perturbations is called the nominal system, and the one in which the perturbations are observed is called the perturbed system. All quantities pertaining to the nominal path are nonprimed and those pertaining to the perturbed path are primed.

Assume that a perturbation is created prior to the  $k$ -th activation of the activity  $j$ ,  $A_{jk}$ , and its size is small. Gain of an entity  $i$ ,  $E_i$ , at the start of  $k$ -th activation of this particular activity can be given as a function  $G_i(A_{jk})$  as follows:

$$G_i(A_{jk}) \triangleq T_i(A_{jk}) - T_i'(A_{jk}) \quad i \in S_{jk}$$

where

$T_i(A_{jk})$  = The arrival time of  $E_i$  to its queue  $Q_i$  preceding activity  $j$ , prior to its  $k$ -th activation in the nominal system,

$T_i'(A_{jk})$  = The arrival time of  $E_i$  to its queue  $Q_i$  preceding activity  $j$ , prior to its  $k$ -th activation in the perturbed system,

$S_{jk}$  = The set of all the entities involved in  $A_{jk}$   
(Assuming they are invariant for both systems).

$G_i(A_{jk})$  measures the difference in arrival of  $E_i$ ,  $i \in S_{jk}$  prior to  $A_{jk}$  between the nominal path and the perturbed path. It could be positive, negative or zero corresponding to the cases of local gain, local loss or no gain, respectively.

Since an activity will start as soon as all its input entities are available in their respective queues, the starting time of the  $k$ -th activation of activity  $j$  in the nominal system,  $T(A_{jk})$ , is given by

$$T(A_{jk}) = \text{Max}_{i \in S_{jk}} T_i(A_{jk})$$

Let  $E_c$  denote the critical entity which arrives last with

$$T(A_{jk}) = T_c(A_{jk}).$$

Since it determines the starting time of  $A_{jk}$ ,  $E_c$  is called the critical entity for this particular activation  $A_{jk}$ .

Similarly the starting time of  $A_{jk}$  in the perturbed system,  $T'(A_{jk})$ , is given by

$$\begin{aligned} T'(A_{jk}) &= \text{Max}_{i \in S_{jk}} T'_i(A_{jk}) \\ &= \text{Max}_{i \in S_{jk}} ( T_i(A_{jk}) - G_i(A_{jk}) ), \end{aligned}$$

and the critical entity for  $A_{jk}$  in the perturbed system is  $E_c$ , with

$$T'(A_{jk}) = T'_c(A_{jk}) = T_c(A_{jk}) - G_c(A_{jk}).$$

Comparing with the nominal system it is seen that the entities,  $E_i$ ,  $i \in S_{jk}$ , will all have a local gain given by  $( T(A_{jk}) - T'(A_{jk}) )$  at the end of  $A_{jk}$ , assuming of course that no perturbation is applied in the duration of  $A_{jk}$ . These final gains of all the entities involved in  $A_{jk}$  can be written in terms of the local gains of the critical

entities

$$\begin{aligned}
 (4.3) \quad T(A_{jk}) - T'(A_{jk}) &= T_c(A_{jk}) - (T_{c'}(A_{jk}) - G_{c'}(A_{jk})) \\
 &= G_{c'}(A_{jk}) + (T_c(A_{jk}) - T_{c'}(A_{jk})) \\
 &= G_{c'}(A_{jk}) + W_{c'}(A_{jk})
 \end{aligned}$$

where

$W_{c'}(A_{jk})$  = the waiting time of  $E_{c'}$  for  $A_{jk}$  in the nominal system.

Consider two possibilities that could occur:

i. The critical entity remains invariant in the nominal and the perturbed systems. This means  $E_c = E_{c'}$ . If we consider this case in equation 4.3, final gain becomes

$$T(A_{jk}) - T'(A_{jk}) = G_{c'}(A_{jk}) = G_c(A_{jk}).$$

In other words, the critical entity forces its gain as a whole to all the entities involved in  $A_{jk}$ . All the entities coming with gains  $G_i(A_{jk})$ ,  $i \in S_{jk}$  would leave activity  $j$  with gain  $G_c(A_{jk})$ .

ii. The critical entity for  $A_{jk}$  is different in the two systems. This means  $E_{c'} \neq E_c$ . In this case final gain is

$$T(A_{jk}) - T'(A_{jk}) = G_{c'}(A_{jk}) + W_{c'}(A_{jk}).$$

It should be noted that

$$G_c(A_{jk}) \leq G_{c'}(A_{jk}) + W_{c'}(A_{jk}) \leq G_{c'}(A_{jk}).$$

Thus the final gain is determined by the critical entities in the two systems with a value between their original local gains.

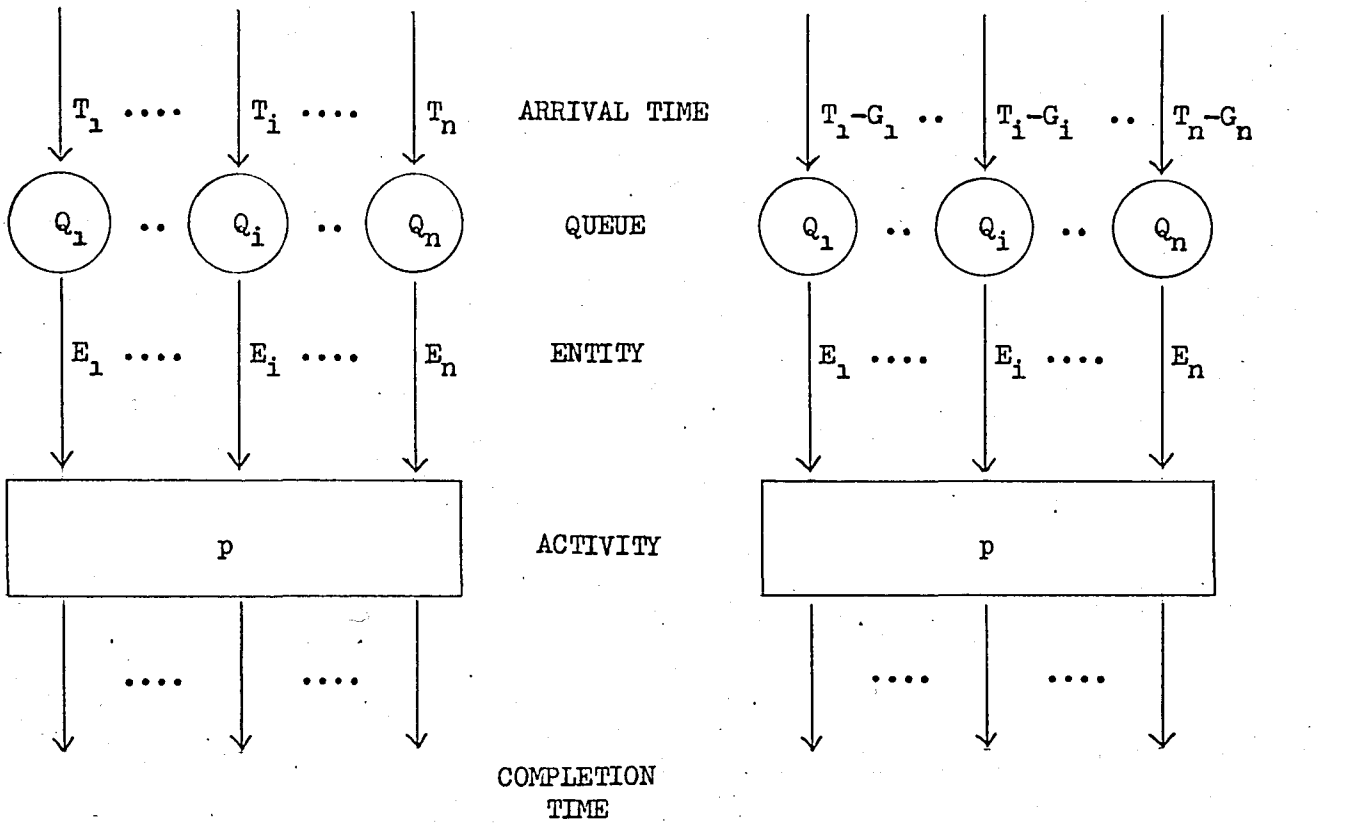
The summary of this formal view is shown in Figure 4.1., assuming that the variables are functions of a particular activation of an activity, although not stated explicitly.

Up to now it is assumed that  $A_{jk}$  itself would have no perturbation. But there could be a small perturbation,  $\Delta p_{jk}$ , in the duration of this particular activation. In this case the duration of  $A_{jk}$  in



NOMINAL PATH

PERTURBED PATH



$$T + p = \text{Max} ( T_1, \dots, T_i, \dots, T_n ) + p$$

$$= T_c + p$$

$$T' + p = \text{Max} ( T_1 - G_1, \dots, T_i - G_i, \dots, T_n - G_n ) + p$$

$$= (T_c, -G_c) + p$$

$$\text{final gain} = (T + p) - (T' + p)$$

$$= T_c - (T_c, -G_c)$$

$$= G_c + W_c$$

Figure 4.1. Propagation of local gains through an activity

in the perturbed system is  $p_{jk} + \Delta p_{jk}$  where  $p_{jk}$  is the duration of the activity in the nominal system. In general, final gains of the entities at the end of  $A_{jk}$  is given by  $FG(A_{jk})$ , where

$$\begin{aligned} FG(A_{jk}) &= ( T(A_{jk}) + p_{jk} ) - ( T'(A_{jk}) + p_{jk} + \Delta p_{jk} ) \\ &= T_c(A_{jk}) + p_{jk} - ( T_c(A_{jk}) - G_c(A_{jk}) + p_{jk} + \Delta p_{jk} ) \\ &= G_c(A_{jk}) + ( T_c(A_{jk}) - T_c'(A_{jk}) ) - \Delta p_{jk} \\ &= G_c(A_{jk}) + W_c(A_{jk}) - \Delta p_{jk} \end{aligned}$$

for all  $E_i$ ,  $i \in S_{jk}$ .

$\Delta p_{jk}$  is either positive or negative or zero corresponding to the cases where the activity duration is increased or decreased or unchanged, respectively.

It would be clear that gain of an entity will be affected only when it is involved in an activity.

To be successful in the estimation of the sensitivity coefficients by use of Perturbation Propagation Analysis, the perturbations must be so small that for each entity the sequence of activities remains invariant in both the nominal and the perturbed system. This is called the Deterministic Similarity of two systems(11). Stating it once more; the perturbations would be small enough not to change the order of the flow of entities with respect to the nominal path. For example, assume that  $A_{j+1,1}$  and  $A_{jk}$  need an entity  $E_i$  from the same queue and  $A_{jk}$  starts before  $A_{j+1,1}$  in the nominal path. A large perturbation may cause  $A_{j+1,1}$  to start earlier than  $A_{jk}$ , therefore it switches  $E_i$  from  $A_{jk}$  to  $A_{j+1,1}$ , and causes local gains not predictable by the propagation analysis.

#### 4.2. Addition of Routines into ACSIM to Estimate Parameter Sensitivities

The sensitivity of the system output with respect to the individual durations is defined in section 4.1. as

$$\frac{\partial T_s}{\partial p_{jk}} = \lim_{\Delta p_{jk} \rightarrow 0} \frac{\Delta T_s(\Delta p_{jk})}{\Delta p_{jk}}$$

where  $\Delta T_s(\Delta p_{jk})$  is the change in the total simulation duration of the nominal path caused by the perturbation  $\Delta p_{jk}$  in  $A_{jk}$ .

Assuming that this perturbation is small enough this sensitivity coefficient can be written as

$$(4.4) \quad \frac{\partial T_s}{\partial p_{jk}} = \frac{\Delta T_s(\Delta p_{jk})}{\Delta p_{jk}} :$$

Substituting equation 4.4 into equation 4.2 the following result is obtained.

$$\frac{\partial T_s}{\partial p_j} = \sum_{k=1}^{K_j} \frac{\Delta T_s(\Delta p_{jk})}{\Delta p_{jk}}$$

The perturbations  $\Delta p_{jk}$  is taken to be equal for each activation of an activity,  $\Delta p_{jk} = \Delta p_j$   $j = 1, \dots, K_j$ . This equality results in the following simplification:

$$\frac{\partial T_s}{\partial p_j} = \frac{1}{\Delta p_j} \sum_{k=1}^{K_j} \Delta T_s(\Delta p_{jk}) = \frac{\Delta T_s(\Delta p_j)}{\Delta p_j} .$$

$\Delta T_s(\Delta p_j)$  is defined as the change in the total simulation time of the nominal system caused by the perturbations applied in all the

activations of activity  $j$  with the same amount of  $\Delta p_j$ .

ACSIM supplies the change in the system output with respect to the perturbations in each activity,  $\Delta T_s(\Delta p_j)$   $j = 1, \dots, M$ , at the end of a single simulation run by means of new routines. An array of local gains for each activity is kept for each entity in the system. This can be thought as a matrix of local gains,  $GAIN(i,j)$ , whose entries are showing the gain of an entity  $i$  at any time if the perturbations are applied in the duration of activity  $j$ .

The resultant system gain at the end of simulation is given by

$$\Delta T_s(\Delta p_j) = GAIN(I,j) \quad j = 1, \dots, M$$

for each of the activities in the system, where  $I$  is any active entity whose internal clock is equal to the system clock at the end of simulation.

The sensitivity coefficients of the system output with respect to sample durations of each activity is simply found as

$$\frac{\partial T_s}{\partial p_j} = \frac{GAIN(I,j)}{\Delta p_j} \quad j = 1, \dots, M$$

at the end of a single simulation run.

Since these results are valid for this particular simulation run, this analysis is named as Sample Path Analysis(11).

ACSIM keeps track of local gains with respect to each activity for each entity in the system. Local gains are updated when the entities are involved in an activity. The basic rule used in ACSIM for propagation of local gains through an activity is to pass the gain of the critical entity to all the other entities involved in the activity and to

subtract the amount of perturbation from these gains in case there exists a perturbation in this activation.

The critical entity is found while checking the conditions for the start of an activity. If all the conditions are satisfied then the entity whose internal clock equals the system clock is the critical entity. The dynamics of this estimation procedure could be followed in the new structure of ACSIM given in Appendix A.

#### 4.3. Experimentation with IKILER

The approach described in this chapter was applied to IKILER illustrated in Figure 2.5. The total simulation time was 630 minutes when the 60th PAY activity was completed. The sensitivity of the total simulation time with respect to the durations of ARRIVE, ORDER, pay, ORDERPASS and DINNER was estimated, all in one run.

The perturbations in all the activities were 0.001 minute decrease in the sample durations. The highest sensitivity coefficient is observed with respect to the duration of ARRIVE. It is found to be 39. This result was compared with that of brute force analysis. Namely, one more simulation run was performed in which all the activations of ARRIVE were actually decreased by 0.001 minute, and the simulation was terminated at the end of 60th PAY again. The system clock appeared to be 629.961 minutes. The sensitivity coefficient was calculated as

$$\frac{629.961 - 630.000}{- 0.001} = 39 .$$

This is the same result with the one estimated by ACSIM. Then four more simulations were performed to compare the results in case of the other four activities, by the same way above. All the results estimated

by ACSIM are equal to the results of brute force analysis. It should be noted that ACSIM results were taken at the end of a single simulation run, however brute force analysis required six runs, one for the nominal path and five more for the perturbed paths in case of ARRIVE, DINNER, ORDER, ORDERPASS, and PAY. These results are summarized in Figure 4.2.

A new set of five more runs were taken with perturbation values increased to 0.01 minute, to see the effect of that much perturbations in the estimation procedure. As a result, the sensitivity coefficients with respect to the activities ORDER, ORDERPASS, DINNER and PAY still remained the same, but a small discrepancy was observed in case of ARRIVE. This is due to the nonlinearity caused by 0.01 minute perturbations which resulted in the changes in the paths of some entities. The results of these experiments are also included in Figure 4.2.

ACTIVITY	SENSITIVITY COEFFICIENT		
	ACSIM	BRUTE FORCE ANALYSIS	
		0.001 min.	0.01 min.
ARRIVE	39	39	30
ORDER	17	17	17
ORDERPASS	16	16	16
DINNER	0	0	0
PAY	22	22	22

Figure 4.2. Results of sensitivity analysis in IKILER

V. IMPLEMENTATION OF SAMPLE PATH ANALYSIS  
ON A TIME-SHARED COMPUTER SYSTEM

5.1. Statement of the Problem

Consider a company with a time-shared computer system consisting of a single central processing unit(CPU), one disk drive, one tape drive, and twelve terminals as shown in Figure 5.1.(17). The operator of each terminal thinks for an amount of time which is an exponential random variable with mean 75 seconds and then sends a message to the CPU. The arriving jobs join a single queue in front of the CPU and are served in first in-first out manner. If the CPU is idle, the job immediately begins service. If the CPU is idle, the job immediately begins service.

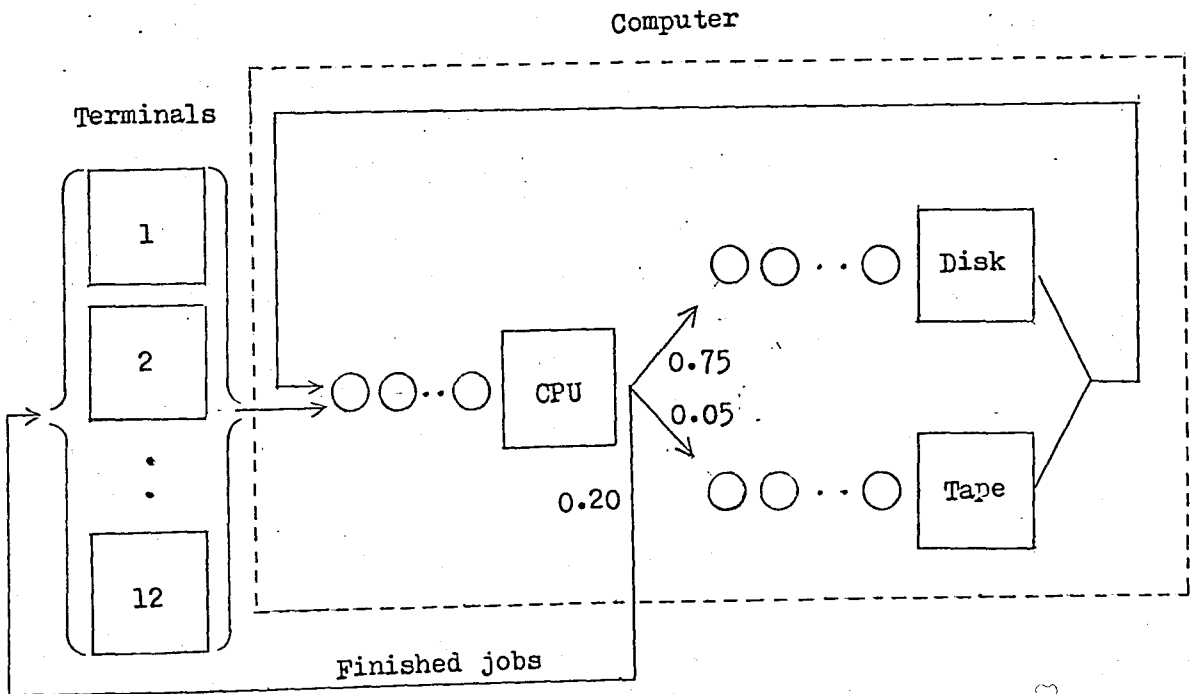


Figure 5.1. A time-shared computer system

Each job occupies the CPU for an amount of time which is a uniform random variable between 1 and 2 seconds. Upon leaving the CPU, a job is either finished, with probability 0.20, independent of the system state, and returns to its terminal to begin another think time, or requires data from the disk drive, with probability 0.75, or needs some data stored on tape, with probability 0.05. After obtaining the data either from disk or tape the job joins the queue in front of the CPU again.

If a job leaving the CPU is sent to the disk drive it may have to join a FIFO queue there until the disk drive is free. The service time at the disk drive is uniformly distributed between 1 and 5 seconds. Similarly if it is sent to tape drive it may have to join a FIFO queue until the tape drive is available. The service duration of the tape drive is an exponential random variable with mean 14 seconds. All service times and think times are independent, and all jobs are initially in the think state at their terminals.

The goal is to estimate the sensitivity of the time required to complete the 6-th request from the tape with respect to the service durations of the CPU, the disk drive, the tape drive and terminal think time.

Later, this sensitivity coefficients could be used to solve the problem of congestions in the computer system occurring through interactions with secondary storage devices, such as disks and tapes. Namely, if the sensitivity coefficient with respect to the duration of tape drive turns out to be considerably high, a solution to the problem of congestion would be to try to decrease the service time of the tape drive. This could be maintained by rearranging the locations of the stored data, since the total service time of a request from a tape depends on the location addressed by the request previously served(18).



## 5.2. Modeling the System

The time-shared computer system is modeled using activity cycles. The activity cycle diagram of this system is shown in Figure 5.2.

The activities of the system are; thinking time at the terminal, processing of the job at the CPU, requesting data from disk, and requesting data from tape, and these activities are named after their location as TERMINAL, CPU, DISK, and TAPE, respectively. The entity classes of the system are the jobs, the CPU, the disk drive and the tape drive, which are named as JOB, CPUF, DISKDRV, TAPEDRV, respectively. Two dummy activities DUMDISK and DUMTAPE are used to take the jobs into the queues in front of the disk and tape.

Initially all the jobs are in the terminal think state. At the end of think duration the jobs are taken from TERMINAL into the queue WAITING, in front of the CPU. CPU would start service if CPUF is FREE and a JOB is WAITING. To be able to determine the flow of jobs in the system an attribute for each job is created while leaving the CPU. Attribute DUNI, having a Discrete UNIFORM value between 1 and 100 is set at the end of service at CPU before joining the queue READY. A READY JOB with an attribute value  $DUNI > 25$  would take place in the activity DISK, since it is given that 75 percent of the jobs leaving CPU requests data from the disk drive. After taking place in the dummy action DUMDISK, with zero duration, the job is taken into the succeeding queue INQDISK, in front of the activity DISK. Disk would start service if DISKDRV is IDLE and a JOB is in INQDISK. Similarly if  $20 \leq DUNI \leq 25$  then the READY JOB is first activated in DUMTAPE, with zero duration, then taken into the queue INQTAPE in front of the tape. TAPE is activated if TAPEDRV is FREE and a JOB is in INQTAPE. Finally, if  $DUNI \leq 20$  it implies that the JOB is

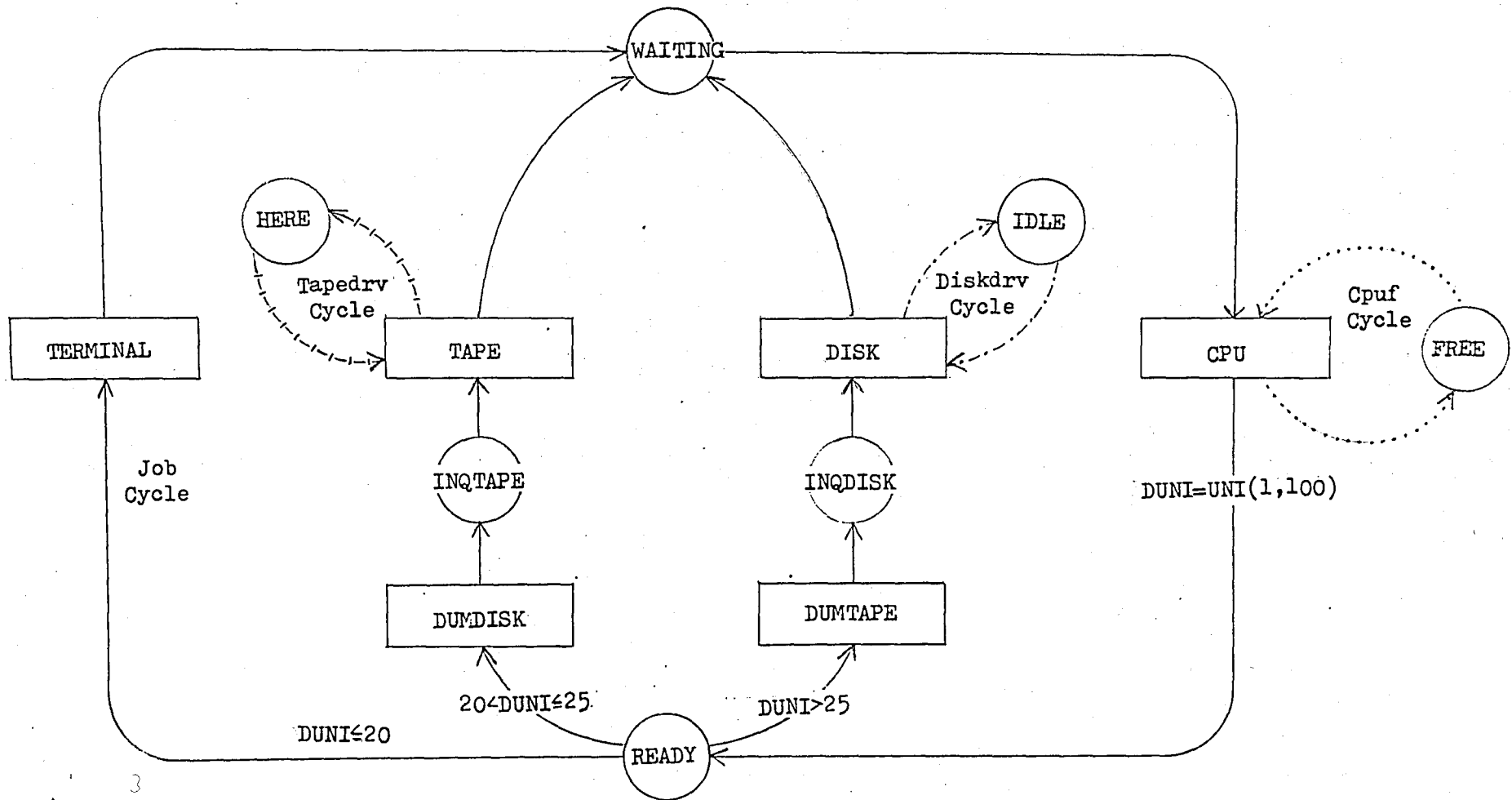


Figure 5.2. Activity cycle diagram of the time-shared computer system

finished and the job turns back to its terminal. After the activations of either DISK or TAPE, the JOB joins the queue WAITING for CPU again.

CPUF, DISKDRV, TAPEDRV would go back to their idle states FREE, IDLE, and HERE after each activation of CPU, DISK and TAPE, respectively.

After explaining the dynamics of the model it would be very easy to understand the input for ACSIM to estimate the parameter sensitivities. The input format is presented in Figure 5.3.

```

COMPUTER:                                15
CPU      JOB WAITING? CPUF FREE?
        AFTER UNI(1,2): JOB READY DUNI=UNI(1,100), CPUF FREE;
DUMDISK  JOB READY DUNI>25?
        AFTER 0: JOB INQDISK;
DUMTAPE  JOB READY DUNI>20?
        AFTER 0: JOB INQTAPE;
TAPE     JOB INQTAPE? TAPEDRV HERE?
        AFTER XPO(14): JOB WAITING, TAPEDRV HERE;
DISK     JOB INQDISK? DISKDRV IDLE?
        AFTER UNI(1,5): JOB WAITING, DISKDRV IDLE;
TERMINAL JOB READY?
        AFTER XPO(75): JOB WAITING;
BEGIN    12 READY,1 FREE,1 IDLE,1 HERE,
END      TAPE 6;
PERTURBN DISK 1-,CPU 1-,TERMINAL 1-,TAPE 1-,
FIN.

```

Figure 5.3. ACSIM input for the estimation of parameter sensitivities in the time-shared computer system

### 5.3. Experimental Results

The time-shared computer system was simulated using ACSIM. The simulation ended at the end of sixth activation of tape drive. Total simulation time was found to be 435 seconds. The sensitivity of this simulation time with respect to the durations of CPU, TERMINAL, TAPE, and DISK were estimated while the system was being simulated.

The perturbations introduced in all the activities were 0.01 second. It was observed that the sensitivity coefficient with respect to the request time from disk was really significant. Then four more simulations were performed by actually decreasing the durations of all the activations for each activity. The results of these brute force runs were exactly the same with the ones estimated by use of ACSIM.

Later the amount of perturbations were increased to observe the system behavior to larger perturbations. With perturbations of 0.02 second, deterministic similarity between the nominal and the perturbed path was still preserved, and ACSIM supplied the exact results. But when the perturbations were increased to 0.05 second, this much decrease in DISK service time caused some entities to change their paths and perturbation propagation procedure resulted in not the exact but still a good result for the sensitivity with respect to DISK durations. However this is not an unexpected result since 0.05 seconds perturbation in the duration of CPU, should not be considered as a small perturbation considering that the mean of the distribution for DISK durations is 3 seconds. The other sensitivity coefficients were still exactly the same when the perturbations were increased to 0.1 second.

It should be noted that since a linear estimation procedure is used, the estimates of sensitivity coefficients obtained by use of ACSIM

would be the same whatever the size of perturbation is. However ACSIM would give exact results with that of brute force analysis in case of small enough perturbations. To show that ACSIM would still give good results in case of finite perturbations, four more simulation runs were performed applying 1 second perturbation in all the activities. The estimates of ACSIM for TERMINAL and TAPE were exactly the same with that of brute force runs, where the estimates for CPU and DISK were not the same but sufficient to observe the level of significance.

All of the experimental results are presented in Figure 5.4. The number of activations of each activity is also included. This would help to analyze the percentage of the times where the local gains are realized as the system gain. For example, DISK started 126 times and sensitivity of the simulation time to the service duration of DISK was found to be 105. This means almost 83.3 per cent of the time the local gains produced in DISK were realized by the system.

ACTIVITY	NUMBER OF STARTS	SENSITIVITY COEFFICIENT				
		ACSIM	BRUTE FORCE ANALYSIS			
			0.01,0.02 s.	0.05 sec.	0.1 sec.	1.0 sec.
TERMINAL	49	2	2	2	2	2
TAPE	6	1	1	1	1	1
CPU	170	6	6	6	6	9
DISK	126	105	105	95	97	59

Figure 5.4. Comparative results of Sample Path Analysis on the time-shared computer system.

The average CPU time it takes ACSIM to estimate all the sensitivity coefficients while simulating the system was observed to be 26 seconds. Each simulation run in which the durations of the activities were actually decreased by the amount of perturbation took 25 seconds of CPU time in average. This means all the sensitivity coefficients were estimated in 26 seconds by ACSIM, and in  $25+25+25+25+25 = 125$  seconds by brute force analysis, the first run for the observation of the nominal path and the remaining four to calculate the sensitivity coefficients with respect to each activity. It should be noted that ACSIM brings a significant amount of decrease in computer time.

The ACSIM output showing the estimated coefficients is given below.

```

COMPUTER
SIMULATION ENDED AT TAPE 6
SIMULATION ENDED AT CLOCK = 435
CPU HAS STARTED 170 TIMES
DUMDISK HAS STARTED 127 TIMES
DUMTAPE HAS STARTED 6 TIMES
TAPE HAS STARTED 6 TIMES
DISK HAS STARTED 126 TIMES
TERMINAL HAS STARTED 49 TIMES

SENSITIVITY OF CLOCK WRT PERTURBATIONS:
ACTIVITY PERTURBATION SENSITIVITY
CPU -1 6
DUMDISK 0
DUMTAPE 0
TAPE -1 1
DISK -1 105
TERMINAL -1 2

ENTITIES AT 435
1 JOB : 1 TAPE 1 DISK 9 TERMINAL
        1 INQDISK
2 CPUF : NO ACTIVE CPUF
        1 FREE
3 TAPEDRV : 1 TAPE
           NO TAPEDRV IN ANY QUEUE
4 DISKDRV : 1 DISK
           NO DISKDRV IN ANY QUEUE

QUEUES AT 435
1 JOB WAITING 0:
2 CPUF FREE 1: 13
3 JOB READY 0:
4 JOB INQDISK 1: 5
5 JOB INQTAPE 0:
6 TAPEDRV HERE 0:
7 DISKDRV IDLE 0:

```

Figure 5.5. Estimated sensitivity coefficients, ACSIM output of the time-shared computer system

## VI. DISCUSSION

There are many different ways in which the efficiency of computer simulations can be improved. Efficiency can be defined by various measures such as the minimization of the variances of sample means, minimization of the time for individual computer runs, or reduction of the number of computer runs required. This study aims to reduce the number of computer runs required for the optimization of the system output by a considerable amount.

To optimize some performance measure in a simulation study, each parameter is changed and the performance measure sensitivity with respect to each change is computed by actually simulating the system after each change. This gives an estimate of the gradient vector of the performance measure. After examining these coefficients, a new set of parameters is found and checked, in this manner the performance measure is optimized, iteratively. The disadvantage of this method is that for  $M$  parameters in the system each step requires  $M$  new simulations which is very demanding. This thesis brings an efficient alternative, which has been developed recently, to this brute force simulation analysis.

This approach estimating the gradient vector of the system output in a single simulation run while observing the nominal system, gives a computational advantage of  $M$  to 1. It can deal with any queueing network, and has full accuracy in case of small enough perturbations. Even in the case that the perturbations are not small enough to

preserve the deterministic similarity, the gradient vector estimated by this approach gives an idea about the relative effectiveness of the perturbations applied in different activities.

The degree of linearity of a system affects the deterministic similarity, and it changes from one system to another. Although it is not a subject of discussion in this study, it could be addressed by all optimization problems.

In summary, the new approach requires only one single observation history, estimates all the sensitivity coefficients supplied by calculations based on perturbation propagation analysis, and it is simple enough to be implemented on any computer.



## REFERENCES

1. Y. C. Ho, M. A. Eyler, T. T. Chien, "A Gradient Technique for General Buffer Storage Design in a Serial Production Line", International Journal on Production Research, Vol. 17, No. 6, pp. 557-580, 1979.
2. Y. C. Ho, M. A. Eyler, "Analysis of Large Scale Discrete Event Dynamical Systems, Proc. IEEE Int. Conf. on Circuits and Computers, 1980.
3. A. T. Clementson, CAPS/ECSL Reference Manual, Univ. of Birmingham, 1978.
4. Y. C. Ho, M. A. Eyler, T. T. Chien, "A New Approach to Determine Parameter Sensitivities of Transfer Lines, Management Science, to appear 1983.
5. Y. C. Ho, X. Cao, "Perturbation Analysis and Optimization of Queueing Networks, Journal of Optimization Theory and Applications, to appear 1983.
6. Y. C. Ho, X. Cao, C. Cassandras, "Infinitesimal and Finite Perturbation Analysis for Queueing Networks, AUTOMATICA, to appear 1983.
7. Y. C. Ho, C. Cassandras, "A New Approach to the Analysis of Discrete Event Dynamic Systems, AUTOMATICA, to appear 1983.
8. R. Suri, "Implementation of Sensitivity Calculations on a Monte Carlo Experiment, Journal of Optimization Theory and Applications, to appear 1983.
9. R. Suri, X. Cao, "The Phantom Customer and Marked Customer Methods for Optimization of Closed Queueing Networks with Blocking and General Service Times."
10. Y. C. Ho, C. Cassandras, R. Suri, "Stochastic Similarity and Statistical Linearity."
11. Y. C. Ho, "SPEEDS: A New Technique for the Analysis and Optimization of Queueing Network ", Technical Report, No. 675, Harvard University, 1983.

12. P. J. Kiviat, Simulation Languages, Appendix C in Computer Simulation Experiments with Models of Economic Systems by T. H. Naylor, John Wiley and Sons, INC., 1971.
13. M. A. Eylar, "ACSIM: A Simulation Program Based on Activity Cycles", Boğaziçi University, 1982.
14. G. K. Hutchinson, "Activity Cycles: A Basis for Manufacturing Systems and Control", ASME Winter Annual Meeting, Dynamic Systems Control Division, 1979.
15. N. Wirth, Systematic Programming: An Introduction, Prentice-Hall, 1973.
16. P. Grogono, Programming in PASCAL, Addison-Wesley Pub. Company, 1980.
17. A. M. Law, W. D. Kelton, Simulation Modeling and Analysis", McGraw-Hill Book Company, 1982.
18. E. G. Coffman, M. Hofri, "A Class of FIFO Queues Arising in Computer Systems", Operations Research, Vol. 26, No. 5, 1978.

APPENDIX A

DYNAMICS OF ACSIM

Main Program:

```
BEGIN
    initialize;
    REPEAT
        move entities;
        scan activities;
        IF none of the activities could start
            THEN update time;
    UNTIL the end of simulation;
    report
END.
```

Initialize:

```
create the given number of entities, activities, and queues;
set system clock;
FOR each entity
    BEGIN set internal clock;
        FOR each activity
            set local gain
    END.
```

Move entities:

```
FOR each active entity
    IF internal clock equals system clock
        THEN move it into its respective queue.
```

Scan activities:

FOR each activity

    BEGIN check conditions;

        IF all the conditions are satisfied

            THEN BEGIN

                include the amount of perturbation into the local  
                gain of the critical entity;

                perform actions

            END

    END.

Update time:

advance the system clock to the smallest internal clock of the  
active entities.

Report:

print the system clock;

FOR each activity

    print the number of times it started;

FOR each activity

    print the amount of perturbation and the sensitivity coefficient;

FOR each queue

    print the number of entities in the queue.

Check conditions:

IF the number of entities in queue is less than requirement

THEN condition is not satisfied

ELSE IF no attribute is specified

THEN BEGIN

condition is satisfied;

specify the entities to be activated;

check if the critical entity is one of them

END

ELSE BEGIN

search for an entity with desired attribute;

IF found

THEN BEGIN

condition is satisfied;

check if it is the critical entity

END

ELSE condition is not satisfied

END.

Perform actions:

FOR each entity to be activated

BEGIN take entity from its queue;

specify its next state;

IF an attribute is specified THEN set its value;

update the internal clock;

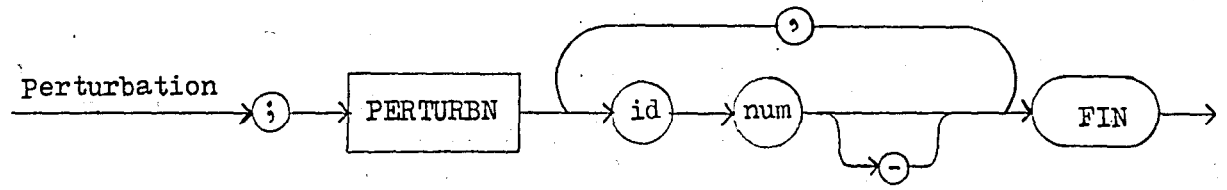
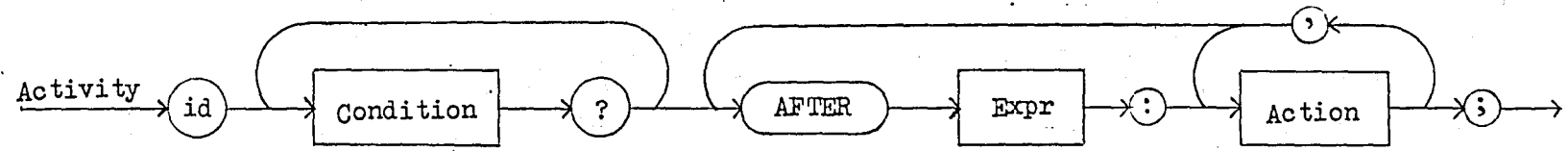
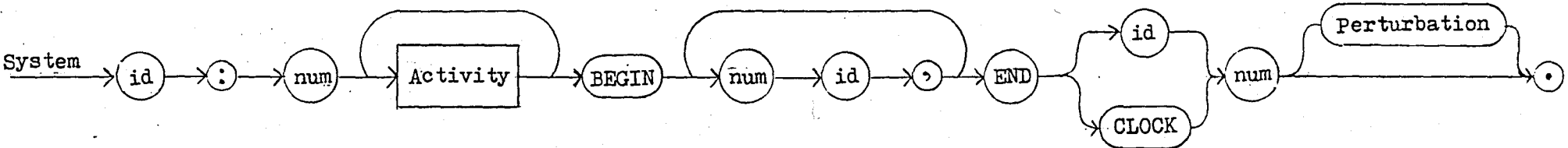
FOR each activity

update the local gain

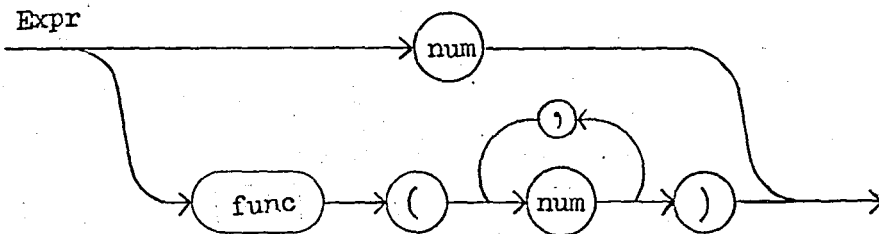
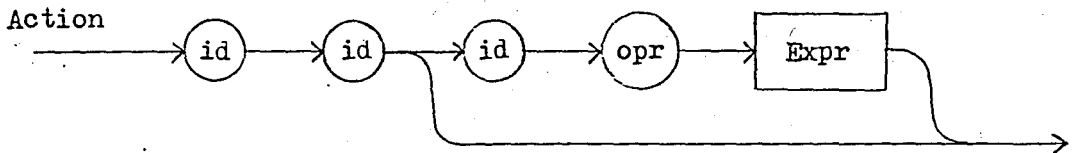
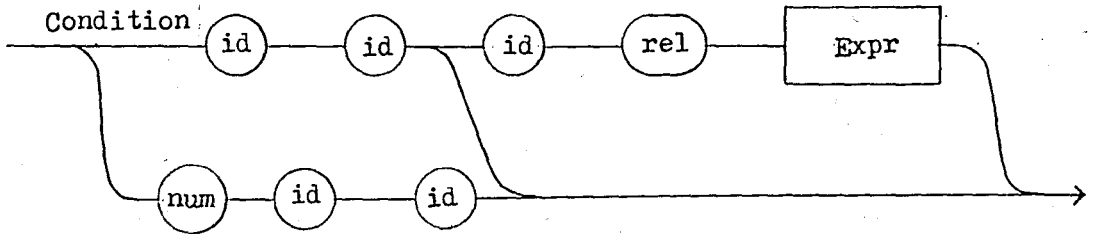
END.

APPENDIX B

SYNTAX DIAGRAM FOR ACSIM







Key words: AFTER, BEGIN, END, CLOCK, PERTURBN, FIN

Terminal Symbols:

id = identifier (sequence of letters and digits)

num = number (sequence of digits)

rel  $\equiv$   $\left\{ \begin{array}{l} > \\ = \text{ (equals)} \\ < \end{array} \right.$

opr  $\equiv$   $\left\{ \begin{array}{l} + \\ = \text{ (assign)} \\ - \end{array} \right.$

func  $\equiv$   $\left\{ \begin{array}{l} XPO \\ UNI \\ NOR \end{array} \right.$

APPENDIX C

ACSIM PROGRAM LISTING

```

RAM ACSIM;

NST AMAX=20;          WAFTER='AFTER'      ;
SMAX=90;            WCLOCK='CLOCK'        ;
CMAX=15;           WBEGIN='BEGIN'        ;
QMAX=40;           WEND='END'            ;
EMAX=200;         WXPO='XPO'            ;
WL=8;             WUNI='UNI'            ;
SL=60;           WNOR='NOR'            ;
PNT='';          WPERTURBN='PERTURBN' ;
BLANK='';        COMMA=',';            SEMCOL=';'      ;
EQL='';         PLUS='+';            MINUS='-';      COLON=':';
                                     QMARK='?';
                                     LSS='<';

(P) ATYP=0; AMAX;
STYP=0; SMAX;
CTYP=0; CMAX;
QTYP=0; QMAX;
ETYP=0; EMAX;
WORD=PACKED ARRAY[1..WL] OF CHAR;
FUNCTN=(CON,XPO,UNI,NOR);
EXPRESSION=RECORD CASE FNC:FUNCTN OF
    CON,XPO: (N:INTEGER);
    UNI,NOR: (P,Q:INTEGER) END;
ACTIVITY =RECORD LASTCON, LASTACT:STYP; LASTENT:ETYP;
    SEED,COUNT,PRTV:INTEGER;
    GOON:BOOLEAN; NAME:WORD END;
STATEMENT =RECORD POS:QTYP; REQ:ETYP;
    OPR:CHAR; XPR:EXPRESSION END;
ENTITY =RECORD LOC:QTYP; TIME, ATRB:INTEGER;
    GAIN:ARRAY[1..AMAX] OF INTEGER;
    CASE ACTIVE:BOOLEAN OF
        TRUE:(ACTNUM:ATYP);
        FALSE:(SUC,PRE:ETYP) END;
QUEUE =RECORD HEAD,TAIL,NUM:ETYP; CLASS:CTYP END;

AR NA:ATYP;          CLOCK,ENDV,INITIALSEED,ERCOUNT:INTEGER;
NS:STYP;           ENDC:ATYP;
NC:CTYP;          SENS,MOVESCAN,ENDSIM:BOOLEAN;
NQ:QTYP;
NE:ETYP;
SYSNAME:PACKED ARRAY[1..SL] OF CHAR;
ACT:ARRAY[1..AMAX] OF ACTIVITY;
STA:ARRAY[1..SMAX] OF STATEMENT;
QUE:ARRAY[1..QMAX] OF QUEUE;
ENT:ARRAY[1..EMAX] OF ENTITY;
ACTN:ARRAY[1..CMAX,1..AMAX] OF ETYP;
CNAME,ANAME:ARRAY[1..CMAX] OF WORD;
QNAME: ARRAY[1..QMAX] OF WORD;

UNCTION RND(VAR S:INTEGER):REAL;
BEGIN S:=(25173*S+13849) MOD 65536; RND:=S/65536 END;

ROCEDURE PUTWORD(W:WORD);
VAR I:1..WL; C:ARRAY[1..WL] OF CHAR;
BEGIN UNPACK(W,C,1);
FOR I:=1 TO WL DO IF C[I]<>BLANK THEN WRITE(C[I])
END;

ROCEDURE DUMPCLS;
VAR Q:QTYP; A:ATYP; C:CTYP; NOENT:BOOLEAN;
BEGIN WRITELN; WRITELN('ENTITIES AT ',CLOCK,':');
FOR C:=1 TO NC DO
BEGIN WRITE(C:2,CNAME[C]:9,COLON,BLANK);
NOENT:=TRUE;
FOR A:=1 TO NA DO
IF ACTNEC[A]>0 THEN
BEGIN WRITE(ACTNEC[A]:2,ACTCA[A].NAME:9,BLANK:);
NOENT:=FALSE
END;
IF NOENT THEN WRITE('NO ACTIVE ',CNAME[C]);
WRITELN; WRITE(BLANK:13);
NOENT:=TRUE;
FOR Q:=1 TO NQ DO WITH QUE[C] DO
IF (C=CLASS) AND (NUM>0) THEN
BEGIN WRITE(NUM:2,QNAME[C]:9,BLANK:3);
NOENT:=FALSE
END;
IF NOENT THEN WRITE('NO ',CNAME[C],', IN ANY QUEUE');
WRITELN
END
END:--(*DUMPCLS*)

```

```

PROCEDURE DUMPQUE;
VAR Q:QTY; F:ETYP;
BEGIN Writeln('QUEUES AT ',CLOCK,':');
FOR Q:=1 TO NQ DO WITH QUE[Q] DO
  BEGIN
    WRITE(Q:2,BLANK,CNAME[CLASS],BLANK,QNAME[Q],NUM:4,CLOCK);
    E:=HEAD;
    WHILE E>0 DO BEGIN WRITE(BLANK,E); E:=ENT[Q].SUC END;
    Writeln
  END
END; (* DUMPQUE *)

PROCEDURE DUMPENT;
VAR E:ETYP;
BEGIN Writeln('ENTITIES AT ',CLOCK,':');
FOR E:=1 TO NE DO WITH ENT[E] DO
  Writeln(E:2,SUC:4,PRE:4,LOC:4,TIME:8,ATRB:8)
END; (* DUMPENT *)

PROCEDURE DUMPSTA;
VAR A:ATYP; S:STYP; C:CTYP; Q:QTY;

PROCEDURE PUTEXP(EX:EXPRESSION);
BEGIN WITH EX DO
  CASE FNC OF
    CON:WRITE(N);
    XPO:WRITE('XPO(',N,',')');
    UNI:WRITE('UNI(',P,COMMA,Q,',')');
    NOR:WRITE('NOR(',P,COMMA,Q,',')');
  END
END; (*PUTEXP*)

BEGIN (*DUMPSTA*)
  S:=0;
  FOR A:=1 TO NA DO WITH ACT[A] DO
    BEGIN Writeln(NAME);
      REPEAT S:=S+1; WRITE(BLANK:5);
        WITH STAC[S] DO
          IF POS>0
            THEN BEGIN
              C:=QUE[POS].CLASS;
              IF REQ>1 THEN WRITE(REQ,BLANK);
              PUTWORD(CNAME[C]); WRITE(BLANK);
              PUTWORD(QNAME[POS]);
              IF OPR<>BLANK
                THEN BEGIN PUTWORD(ANAME[C]);
                  WRITE(OPR);PUTEXP(XPR)
                END;
              IF S=LASTACT
                THEN Writeln(SEMCOL)
                ELSE IF S>LASTCON THEN Writeln(COMMA)
                ELSE Writeln(QMARK)
            END;
          ELSE BEGIN
            WRITE(WAFTER);PUTEXP(XPR);Writeln(COLON)
          END
        UNTIL S=LASTACT
      END;
    WRITE(WBEGIN);
    FOR Q:=1 TO NQ DO WITH QUE[Q] DO
      IF NUM>0 THEN WRITE(NUM,BLANK,QNAME[Q],COMMA); Writeln;
      WRITE(WEND);
      IF ENDC=0 THEN WRITE(WCLOCK)
        ELSE WRITE(ACT[ENDC].NAME);
      Writeln(ENDDV,PNT); Writeln; Writeln
    END; (* DUMPSTA *)

PROCEDURE INITIALIZE;
VAR W:WORD; CH:CHAR; V,I:INTEGER; NUMERAL, LAST:BOOLEAN;
    E:ETYP; Q:QTY;

PROCEDURE ERROR (N:INTEGER);
BEGIN Writeln('ERROR ',N,BLANK,W,BLANK,CH);
  ERCOUNT:=ERCOUNT+1
END;

```

```

PROCEDURE GETWORD;
(* READS NEXT IDENTIFIER INTO W AND THE FOLLOWING SYMBOL INTO CH;
(* IF IDENTIFIER IS NUMERAL THEN V CONTAINS ITS VALUE, OTHERWISE 0
VAR I:0..WL; C:ARRAY[1..WL] OF CHAR;
BEGIN
  WHILE NOT(CH IN ['A'..'Z','0'..'9']) DO READ(CH);
  V:=0; I:=0; NUMERAL:=TRUE;
  REPEAT
    IF I<WL THEN
      BEGIN I:=I+1; C[I]:=CH;
            IF NUMERAL AND (CH IN ['0'..'9'])
              THEN V:=10*V+ORD(CH)-ORD('0')
              ELSE NUMERAL:=FALSE
            END;
      READ(CH)
    UNTIL NOT(CH IN ['A'..'Z','0'..'9']);
    WHILE I<WL DO BEGIN I:=I+1; C[I]:=BLANK END;
    PACK(C,1,W);
    WHILE CH=BLANK DO READ(CH)
  END; (* GETWORD *)

```

```

PROCEDURE GETQUE(VAR Q:QTYP);
VAR C:CTYP;
BEGIN
  CNAME[NC+1]:=W;
  C:=0; REPEAT C:=C+1 UNTIL CNAME[C]=W;
  GETWORD; QNAME[NQ+1]:=W;
  IF C>NC
    THEN BEGIN NC:=C; Q:=NQ+1 END
    ELSE BEGIN Q:=0; REPEAT Q:=Q+1 UNTIL QNAME[Q]=W END;
  IF Q>NQ
    THEN BEGIN NQ:=Q; WITH QUE[Q] DO
      BEGIN HEAD:=0; TAIL:=0; NUM:=0; CLASS:=C END
    END
  END; (* GETQUE *)

```

```

PROCEDURE GETEXP(VAR EX:EXPRESSION);
BEGIN
  GETWORD;
  WITH EX DO
    IF NUMERAL THEN
      BEGIN FNC:=CON; N:=V END
    ELSE IF W=WXPO THEN
      BEGIN FNC:=XPO; GETWORD; N:=V END
    ELSE IF W=WUNI THEN
      BEGIN FNC:=UNI; GETWORD; P:=V; GETWORD; Q:=V END
    ELSE IF W=WNOR THEN
      BEGIN FNC:=NOR; GETWORD; P:=V; GETWORD; Q:=V END
    ELSE ERROR(7)
  END; (* GETEXP *)

```

```

PROCEDURE CONDITION;
BEGIN WITH STALNSJ DO
  IF NUMERAL
    THEN BEGIN REQ:=V; GETWORD; GETQUE(POS); OPR:=BLANK END
    ELSE BEGIN
      REQ:=1; GETQUE(POS);
      IF CH=QMARK
        THEN OPR:=BLANK
        ELSE BEGIN (*ATTRIBUTE OPERATION EXPRESSION*)
          GETWORD; ANAME[QUE[POS].CLASS]:=W;
          OPR:=CH; GETEXP(XPR)
        END
      END;
  WHILE CH<>QMARK DO READ(CH)
END; (* CONDITION *)

```

```

PROCEDURE ACTION;
BEGIN WITH STALNSJ DO
  IF W=WAFTR
    THEN BEGIN (* AFTER CLAUSE *)
      REQ:=0; POS:=0; OPR:=BLANK; GETEXP(XPR);
      WHILE CH<>COLON DO READ(CH)
    END
  ELSE BEGIN (* PROPER ACTION *)
      REQ:=0; GETQUE(POS);
      IF CH IN [COMMA,SEMCOL]
        THEN OPR:=BLANK
        ELSE BEGIN (*ATTRIBUTE OPERATION EXPRESSION*)
          GETWORD; ANAME[QUE[POS].CLASS]:=W;
          OPR:=CH; GETEXP(XPR);
          WHILE NOT(CH IN [COMMA,SEMCOL]) DO READ(CH)
        END
      END
  END;
END

```

```

3BEGIN
  (* INITIALIZE SCALARS *)
  NA:=0; NS:=0; NC:=0; NO:=0; NE:=0; ERCOUNT:=0;
  CLOCK:=0; ENDSIM:=FALSE; MOVESCAN:=TRUE;
  (* READ SYSTEM NAME AND INITIAL SEED *)
  READEN(CH); REPEAT READ(CH) UNTIL CH<>BLANK;
  FOR I:=1 TO SL DO
    IF CH=COLON
      THEN SYSNAME[I]:=BLANK
      ELSE BEGIN SYSNAME[I]:=CH; READ(CH) END;
  WHILE CH<>COLON DO READ(CH);
  GETWORD; IF NUMERAL THEN BEGIN INITIALSEED:=V; GETWORD END
  ELSE INITIALSEED:=0;
  PAGE; WRITELN(SYSNAME, COLON, INITIALSEED:10);
  (* ACTIVITIES AND STATEMENTS *)
  REPEAT NA:=NA+1; WITH ACT[NA] DO
    BEGIN
      NAME:=W; GETWORD;
      REPEAT NS:=NS+1; CONDITION; GETWORD
      UNTIL W=WAFTER;
      LASTCON:=NS;
      REPEAT NS:=NS+1; ACTION; LAST:=(CH=SEMCOL); GETWORD
      UNTIL LAST;
      LASTACT:=NS; COUNT:=0;
      SEED:=TRUNC(RND(INITIALSEED)*65336)
    END
  UNTIL W=WBEGIN;
  (* ENTITIES LOCATED *)
  GETWORD;
  REPEAT IF NOT NUMERAL THEN ERROR(9);
  I:=V; GETWORD;
  QNAME[NO+1]:=W; Q:=0; REPEAT Q:=Q+1 UNTIL QNAME[Q]=W;
  IF Q>NQ
    THEN ERROR(3)
    ELSE REPEAT I:=I+1; NE:=NE+1; WITH ENT[NE] DO
      BEGIN ACTIVE:=TRUE; LOC:=Q; TIME:=0; ATRB:=0 END
    UNTIL I=0;
  GETWORD;
  UNTIL W=WEND;
  (* END CONDITION *)
  GETWORD; ENDC:=0;
  IF W<>WCLOCK THEN
    BEGIN FOR I:=1 TO NA DO
      IF W=ACT[I].NAME THEN ENDC:=I;
      IF ENDC=0 THEN ERROR(4)
    END;
  GETWORD; IF NOT NUMERAL THEN ERROR(9); ENDV:=V; SENS:=FALSE;
  IF CH=SEMCOL THEN
    BEGIN GETWORD; IF W<>WPERTURBN THEN ERROR(3) ELSE SENS:=TRUE;
    FOR I:=1 TO NA DO
      BEGIN ACT[I].PRTV:=0;
      FOR E:=1 TO NE DO
        ENT[E].GAIN[I]:=0;
      END;
    REPEAT GETWORD;
    FOR I:=1 TO NA DO WITH ACT[I] DO
      IF W=NAME THEN
        BEGIN GETWORD; IF NOT NUMERAL THEN ERROR(9);
        IF CH=MINUS THEN PRTV:=(I-1)*V
        ELSE PRTV:=V
        END
    UNTIL CH=PNT;
  END;
  IF 'A' IN OPTIONS THEN
    BEGIN WRITELN; WRITE(' CLOCK ACTIVITY DUR ENT ');
    FOR I:=1 TO NA DO WITH ACT[I] DO
      IF PRTV<>0 THEN WRITE(NAME, PRTV:2); WRITELN;
    END
  END;
  (* INITIALIZE *)

```

```
PROCEDURE MOVEENTITIES;
```

```
VAR E:ETYP; A:ATYP; M:BOOLEAN;
```

```
BEGIN M:=FALSE;
```

```
FOR E:=1 TO NE DO WITH ENT[E] DO
```

```
IF ACTIVE AND (TIME=CLOCK) THEN WITH QUE[LOC] DO
```

```
BEGIN IF HEAD=0 THEN HEAD:=E
```

```
ELSE ENT[TAIL].SUC:=E;
```

```
ACTIVE:=FALSE; NUM:=NUM+1; M:=TRUE;
```

```
SUC:=0; PRE:=TAIL; TAIL:=E
```

```
END;
```

```
IF M THEN FOR A:=1 TO NA DO ACT[A].GOON:=TRUE
```

```
END; (*MOVEENTITIES*)
```

```
PROCEDURE SCANACTIVITIES;
```

```
VAR AA,A:ATYP; S;STYP; CRITICE:ETYP; DUR:INTEGER; M:BOOLEAN;
```

```
MOVE:ARRAY[1..CMA] OF RECORD E,N:ETYP; Q:QTYP END;
```

```
FUNCTION VALUEOF(EX:EXPRESSION):INTEGER;
```

```
VAR X:REAL; J:1..12;
```

```
BEGIN WITH EX DO WITH ACT[A] DO
```

```
CASE FNC OF
```

```
CON: VALUEOF:=N;
```

```
XPO: VALUEOF:=ROUND(N*LN(RND(SEED)));
```

```
UNI: VALUEOF:=TRUNC(P+(Q/P+1)*RND(SEED));
```

```
NOR: BEGIN X:=0; FOR J:=1 TO 12 DO X:=X+RND(SEED);
```

```
VALUEOF:=ROUND(P+Q*(X/6))
```

```
END
```

```
END
```

```
END; (* VALUEOF *)
```

```
FUNCTION CONDITION:BOOLEAN;
```

```
VAR V:INTEGER; CON:BOOLEAN;
```

```
BEGIN WITH ST[CS] DO WITH QUE[POS] DO WITH MOVE[CLASS] DO
```

```
IF NUMREQ THEN CONDITION:=FALSE
```

```
ELSE IF OPR=BLANK
```

```
THEN BEGIN
```

```
CONDITION:=TRUE;
```

```
IF SENS THEN
```

```
BEGIN E:=HEAD; N:=REQ;
```

```
WHILE N>1 DO
```

```
BEGIN E:=ENT[E].SUC; N:=N-1; END;
```

```
IF ENT[E].TIME=CLOCK THEN CRITICE:=E
```

```
END;
```

```
E:=HEAD; N:=REQ; Q:=POS
```

```
END
```

```
ELSE BEGIN
```

```
E:=HEAD; V:=VALUEOF(XPR);
```

```
REPEAT WITH ENT[E] DO
```

```
BEGIN CASE OPR OF
```

```
GTR: CON:=(ATRB>V);
```

```
EQL: CON:=(ATRB=V);
```

```
LSS: CON:=(ATRB<V)
```

```
END;
```

```
IF CON THEN BEGIN N:=1; Q:=POS;
```

```
IF SENS AND (TIME=CLOCK
```

```
THEN CRITICE:=E EN
```

```
ELSE E:=SUC
```

```
END
```

```
UNTIL CON OR (E=0);
```

```
CONDITION:=CON
```

```
END
```

```
END; (*CONDITION*)
```

```

PROCEDURE ACTION;
VAR V:INTEGER; AN:ATYP;
BEGIN WITH STACSJ DO
  IF POS=0
  THEN DUR:=DUR+VALUEOF(XPR)
  ELSE BEGIN (* PROPER ACTION *)
    IF OPR<>BLANK THEN V:=VALUEOF(XPR);
    WITH MOVE[QUEECPOS].CLASSJ DO WITH QUEECQJ DO
      REPEAT WITH ENTCEJ DO
        BEGIN (* TAKE ENTITY E FROM QUEUE Q *)
          IF E=HEAD THEN HEAD:=SUC
            ELSE ENT[PREJ].SUC:=SUC;
          IF E=TAIL THEN TAIL:=PRE
            ELSE ENT[SUCJ].PRE:=PRE;
          NUM:=NUM-1;
          (* ACTIVATE ENTITY E IN NEW POS *)
          LOC:=POS; TIME:=CLOCK+DUR;
          CASE OPR OF
            BLANK :
            PLUS : ATRB:=ATR+B;
            MINUS : ATRB:=ATR-B;
            EQL : ATRB:=V
          END;
          ACT[AJ].LASTENT:=E;
          IF SENS AND (E<>CRITICE) THEN
            FOR AN:=1 TO NA DO
              GAIN[ANJ]:=ENT[CRITICEJ].GAIN[ANJ];
            E:=SUC; ACTIVE:=TRUE; ACTNUM:=A
          END;
          N:=N-1
        UNTIL N=0
      END
    END; (* ACTION *)
PROCEDURE GETACTN;
VAR C:CTYP; A:ATYP; E:ETYP;
BEGIN FOR C:=1 TO NC DO
  FOR A:=1 TO NA DO
    BEGIN ACTN[C,AJ]:=0;
      FOR E:=1 TO NE DO WITH ENTCEJ DO
        IF (C=QUEECLOCJ).CLASS) AND
          ACTIVE AND (ACTNUM=A)
          THEN ACTN[C,AJ]:=ACTN[C,AJ]+1
        END
      END; (*GETACTN*)
BEGIN (* SCANACTIVITIES *)
  FOR A:=1 TO NA DO WITH ACT[AAJ] DO IF GOON THEN
    BEGIN DUR:=0; M:=TRUE;
      IF A=1 THEN S:=0 ELSE S:=ACT[AAJ].LASTACT;
      WHILE M AND (S<LASTCON) DO
        BEGIN S:=S+1; M:=M AND CONDITION END;
        IF M THEN BEGIN
          IF SENS THEN WITH ENT[CRITICEJ] DO
            GAIN[AAJ]:=GAIN[AAJ]-PRIV;
            REPEAT S:=S+1; ACTION UNTIL S=LASTACT;
            COUNT:=COUNT+1;
            IF (ENDC=A) AND (ENDV=COUNT) THEN BEGIN
              Writeln(*SIMULATION ENDED AT*,NAME:9,ENDV:4);
              ENDV:=ENT[CLASTENTJ].TIME; ENDC:=0 END;
            IF 'A' IN OPTIONS THEN BEGIN
              WRITE(CLOCK:6,BLANK,NAME,DUR:5,LASTENT:4);
              IF SENS THEN FOR AA:=1 TO NA DO
                IF ACT[AAJ].PRIV<>0 THEN
                  WITH ENT[CLASTENTJ] DO
                    WRITE(CLOCK,GAIN[AAJ]:9,BLANK);
                  Writeln END
            END
          ELSE GOON:=FALSE
        END;
      GETACTN;
      A:=1; WHILE NOT ACT[AAJ].GOON DO A:=A+1;
      IF A>NA THEN MOVESCAN:=FALSE
    END; (*SCANACTIVITIES*)

```



```

CEDURE UPDATETIME;
VAR TMIN:INTEGER; E:ETYP;
BEGIN
  TMIN:=MAXINT; MOVESCAN:=TRUE;
  FOR E:=1 TO NE DO WITH ENT[E] DO
    IF ACTIVE AND (TIME<TMIN) THEN TMIN:=TIME;
  IF TMIN<MAXINT THEN CLOCK:=TMIN;
  IF (ENDC=0) AND (ENDV<=TMIN) THEN ENDSIM:=TRUE
ND; (*UPDATETIME*)

```

```

CEDURE REPORT;
VAR A:ATYP; E:ETYP;
BEGIN
  WRITELN('SIMULATION ENDED AT CLOCK = ',CLOCK);
  FOR A:=1 TO NA DO WITH ACT[A] DO
    WRITELN(NAME,' HAS STARTED ',COUNT,' TIMES');WRITELN;
  IF SENS THEN
    BEGIN E:=0;
      REPEAT E:=E+1 UNTIL ENT[E].ACTIVE AND (ENT[E].TIME=CLOCK);
      WRITELN('SENSITIVITY OF CLOCK WRT PERTURBATIONS:');
      WRITELN;WRITELN(' ACTIVITY PERTURBATION SENSITIVITY');
      FOR A:=1 TO NA DO WITH ACT[A] DO WITH ENT[E] DO
        IF PRTV<>0
          THEN WRITELN(NAME:11,PRTV:14,
            ROUND((-1)*GAIN[A]/PRTV):11)
          ELSE WRITELN(NAME:11,PRTV:14)
        ;
      ;
    END
  ;
ND; (*REPORT*)

```

```

CEDURE STOP;
BEGIN WRITELN(ERCOUNT,' ERRORS IN ',SYSNAME);HALT END;

```

```

IN INITIALIZE:MOVEENTITIES;
IF 'S' IN OPTIONS THEN DUMPSTA;
IF ERCOUNT>0 THEN STOP;
REPEAT MOVEENTITIES;
  IF 'Q' IN OPTIONS THEN DUMPQUE;
  IF 'E' IN OPTIONS THEN DUMPENT;
  SCANACTIVITIES;
  IF NOT MOVESCAN THEN UPDATETIME
UNTIL ENDSIM;
REPORT; DUMPCLS; DUMPQUE
(* ACSIM *)

```