Evaluation Net Simulation System Reference Manual*

Gary J. Nutt

Department of Computer Science
University of Colorado
Boulder, Colorado 80302

TR#CU-CS-042-74 April 1974

ABSTRACT

This report is a reference manual describing the capabilities, limitations, and usage of the University of Colorado Evaluation Net Simulation System. This software package implements a simulation programming language which is composed of a control flow graph model integrated with the text of the program. The implementation is composed of four subsystems: a graphic net editor, a compiler, a link editor, and an interpreter.

*This work was supported by NSF Grant GJ-660.

Table of Contents

List of Tables

List of Figures

I. INTRODUCTION

The class of evaluation nets has been previously formulated to aid in an orderly approach to simulation model building.[1,2] In this document, it is assumed that the reader has already familiarized himself with the nets and is interested in the corresponding implementation of an evaluation net interpreter.

Briefly, the graph structure of the net and the various procedures define a machine interpretable model. The use of a graph structure as a partial definition is a visual aid to the model designer, and should be an integral part of the implementation. The model designer can then carry on an interactive session with a graphics package to derive the control structure of the model. Subsequent sessions are used to refine the graph model and to define the various procedures for the net. Once an approximation to the design is complete, the user can submit the entire net to an interpreter. The interpreter should then execute the model, given an appropriate set of data. Monitoring the interpretive execution could be done on-line with a suitable graphics console, or it could be an interpreter function with no user intervention to the interpreter once it begins to exercise the net.

The implementation described in this report is for the University of Colorado CDC 6400 system running under the KRONOS operating system. The only special peripheral device required is an interactive graphics console, and in this case, the device we used is a Tektronix 4010 remote display station. This unit is a cathode ray tube device with images stored on the surface of the screen, and thus it is reasonable to employ the device as a remote graphics console with low speed (300 band) communication lines. A refresh scope could provide a wider

1. Nutt, G. J., "The Formulation and Application of Evaluation Nets", University of Washington Computer Science Group, Ph.D. dissertation, Technical Report No. 72-07-02, (1972).
2. Nutt, G. J., "Evaluation Nets for Computer System Analysis", AFIPS Proceedings of the FJCC, Vol. 41, (1972), pp 279-285.

variety of displays if it included suitable local storage and computing power (e.g. a DEC GT-40 with disk auxilliary storage).

The software is primarily composed of FORTRAN IV routines, with the remaining portions of code being written in the CDC 6000 assembly language, COMPASS. An attempt has been made to keep the FORTRAN code reasonably machine independent, those areas relying on word size, etc. being commented as such in the listing. The ANSI standard was also used in writing the FORTRAN programs. One goal of this implementation was to make the programs portable, with a minimum of effort, to either another medium-to-large scale system or a minicomputer system. (It was initially admitted that a minicomputer implementation might be very slow.) The portability constraint was the primary reason for choosing FORTRAN over various other languages available at this installation.

The CDC 6400 - KRONOS implementation of the Evaluation Net Simulation System (ENSS) relies heavily on the permanent file system of KRONOS.[3] All intermediate files between ENSS subsystem steps are saved on KRONOS files. The KRONOS text editor may be used on some of the E-net files.

ENSS is composed of four distinct subsystems (See Figure I.1): ENETDIS, the graphics subsystem for interactive net construction and editing; COMPLR, the text compiler which produces interpretive code to simulate the net; LINK, a link editor which combines two or more nets that have been previously compiled; and INTERP, an interpreter to execute the net. In the following four sections each of the individual subsystems will be discussed with respect to their capabilities, limitations, and usage. Detailed subsystem documentation is not included in this manual.

The following graduate students have participated in various parts of the ENSS-Version 1 implementation: M. Bailey, W. Ellis, and J. Read.

---

3. CDC, "KRONOS 2.0 Reference Manual", publication No. 59150600.

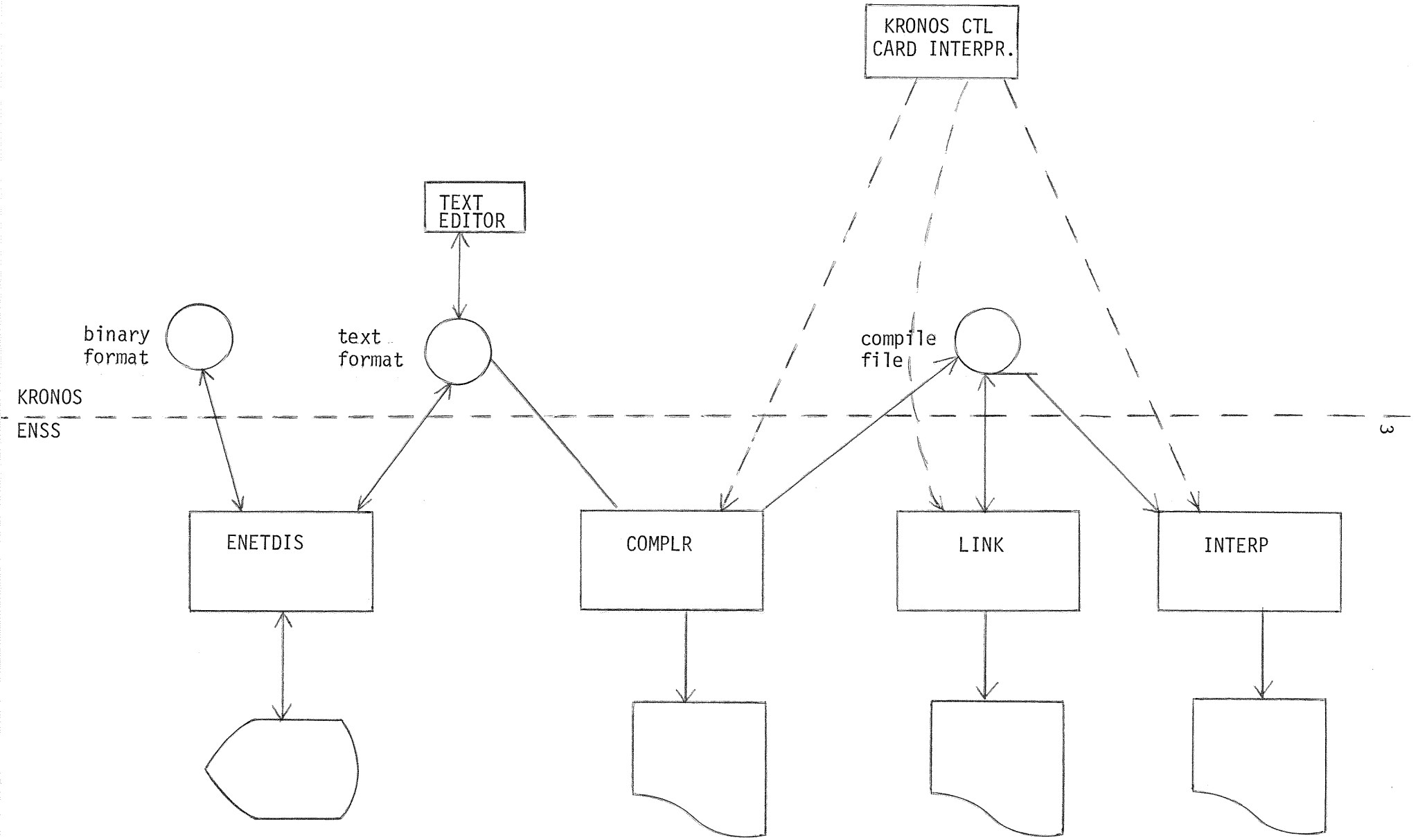

KRONOS CTL
CARD INTERPR.
TEXT
EDITOR
binary
format
text
format
compile
file
KRONOS
ENSS
ENETDIS
COMPLR
LINK
INTERP

Figure I.1

II. THE DISPLAY AND EDIT SUBSYSTEM

A. Capabilities and Limitations

The display and edit subsystem, ENETDIS, is an interactive subsystem designed for use with the Tektronix 4010 display station, (or similar storage scope graphics devices). The purpose of ENETDIS is to allow a user to interactively create and edit the graph structure of an E-net, using the display screen for graphic output and the keyboard for net schema descriptions. The basic design of this system assumes that the user will provide the identification of a set of associated locations with each introduction of a transition; furthermore, the user ordinarily provides the cartesian coordinates of the geometric center of the transition when addressing a transition-location set. ENETDIS does have the capability of constructing a graph structure in the absence of the aforementioned cartesian coordinates, although the transitions are placed on the screen by algorithmically determined cartesian coordinates (rather than user-provided placement).

The subsystem essentially operates in two modes, approximately corresponding to interactive and batch mode. In the interactive mode, ENETDIS input is a set of commands and data which direct the generation of net components, or cause the subsystem to alter currently defined components of the E-net being operated upon. To generate components for the net, the interactive user provides a schema type, transition and location labels, and a cartesian coordinate for the center of the transition. The transition-location set is entered into the description of the current net. Interactive editing of the current net allows the user to change the coordinates (screen position) of any transition-location set as well as the ability to change or rename the transitions or locations.

The "batch mode" of ENETDIS is executed from the graphics console, but causes the subsystem to display an entire E-net from a permanent file containing a text description suitable as input to the compiler subsystem. This format of input does not include screen position coordinates, hence an algorithm is used to compute transition-location set coordinates; the result is that the display often needs to be interactively edited in order to produce a more attractive display.

ENETDIS interprets commands that allow the user to fetch and save KRONOS permanent files in one of two formats called binary format or text format. The text format is the usual (CDC 6000 display code) character string representation described above and is appropriate when the user either wishes to display compiler-input text in the batch mode, or to save an E-net in a format in which the compiler can use. The overhead involved with this file format is caused by the necessity for ENETDIS to convert its internal graph representation either to or from the external textual representation. In a series of console sessions to design the graph structure of a model, it is necessary that intermediate net descriptions be saved; rather than employing the representation transformations to convert between internal and external representations, the user can save an E-net in binary format, (which can be used as subsequent input to ENETDIS).

The level of sophistication incorporated in a graphics display can grow rapidly with a few added constraints. ENETDIS is not a highly sophisticated software tool, but is intended to be flexible enough to aid in E-net graph structure design. The primary limitations imposed by this simplistic approach are machine and installation dependencies in the form of display driver routines.

The line positioning algorithm could stand some improvement, as it allows lines to pass through transitions and locations. Occasionally, lines also coincide. Another significant liability of ENETDIS is its inability to display large E-nets; a windowing process needs to be incorporated to allow for arbitrarily large (in space) displays.

B. Using the Display Subsystem

In the following discussion, at least a superficial understanding of the CDC 6400 - KRONOS system is assumed. Prerequisite knowledge can easily be obtained from the KRONOS Time-Sharing Reference Manual.

ENETDIS is executed under the batch subsystem of KRONOS, but must be called under the TELEX subsystem with the INPUT file corresponding to the display terminal used to initiate the conversation. The following memory requirements (specified as octal numbers) are required:

33100 words to compile

37600 words to load

23200 words to execute.

This execute field length provides for about 100 locations and about 25 transitions, which is likely to be sufficient with the display space limitation discussed earlier.

After logging into the TELEX time sharing system, the following dialogue should ensue, (system output is underlined, and Ⓡ is a carriage return):

SYSTEM: GET,ENET/UN=X652Ⓡ

READY

UNFMTⓇ

READY

CALL,ENETⓇ

The last control command shown is a call to a KRONOS procedure file composed of the following set of control cards (executed in the batch subsystem):

```
GET,TNET/UN=X652.
ATTACH,T4CUG,T4010/UN=LIBRARY.
RFL(40000)
LOAD,TNET,T4CUG,T4010.
ENETDIS.
```

TNET is a permanent file containing the object code of ENETDIS, and T4CUG and T4010 are library files containing (machine and installation dependent) Tektronix 4010 display routines.

Upon executing the "ENETDIS." command in the TNET procedure file, control is passed to the display and edit subsystem, ENETDIS. The subsystem responds by erasing the screen and writing the request

COMMAND

The user must now respond with any of one of the following commands, followed by a carriage return, (illustrated in Tables II.1 - II.5):

Transition-Location Creation Commands

Each of these commands cause a new transition-location set to be introduced into the display data base. Each command causes ENETDIS to request an x-y pair ($0 \leq x \leq 1000$, $0 \leq y \leq 700$) to determine the screen position for the geometric center of the transition by printing the message

POS.

The user must respond with an x-coordinate value, blank, y-coordinate value, and carriage return. ENETDIS then requests a list of labels for the transition-location set by printing the message

LABELS.

The labels must then be entered in the following order: transition label, input locations from top-to-bottom and output locations from top-to-bottom. Labels should consist of 1 to 3 alphanumeric character. Elements of the list are delimited by blanks, and the list is terminated by a carriage return. A run-time FORTRAN diagnostic results if an inappropriate number of labels are entered by the user.

α (R) Construct the display structure for an α-transition and associated locations; α is F, J, T, X, or Y.

Editing Commands

This set of commands provides the capability for changing transition labels, positions, and connections and for changing the name of a location. (Location positions on the screen are algorithmically determined by the screen positions of associated transitions.) The current screen image is not altered by the following commands, although the display data base does reflect the editing.

CHANGE (R) Allows the user to change any of the attributes (other than schema type) of a transition. System response to the command is

TRANS NAME

The user must then enter zero (0) to escape the CHANGE command or a transition name. If the name is not found in the data bank, "NOT FOUND,RETRY" will be printed and the system will wait for another transition name. When a declared transition name is entered, ENETDIS next requests new position and label values by the message

POS,LABELS

The user must respond with a list of x-y coordinates and appropriate labels (see creation commands), delimited with blanks and terminated with a carriage return. If an attribute should not be altered, enter "NC" in the appropriate position in the list. The display structure is modified, although the screen image of the net is not altered unless the DISPLAY command is invoked.

DELETE (R) Allows the user to remove a transition from the active display data bank. The response to this command is

TRANS NAME

The user must then respond with zero (0) to escape the DELETE command or a transition name. If the name is not found in the data bank,

NOT FØUND,RETRY

will be printed and the system will wait for another transition name. When a declared transition name is entered, the corresponding transition is removed from the active display data bank.

LØCHANGE (R) This command causes a location name to be changed. System response to the command is

OLD NAME

The user must then enter zero (0) to escape the LØCHANGE command or a location name. If the name is not found in the declared location list,

NOT FOUND,RETRY

is printed and the system requests another location name. Otherwise, system response is

NEW NAME

to which the user should enter the new name and a carriage return.

REPLACE (R) A transition can be replaced with a transition of another type at the same screen position using this command. System response is

TRANS NAME

The user may escape the REPLACE command by entering zero (0), or he may enter a transition name. If the name is not found in the declared transition list,

NOT FOUND,RETRY

will be printed and the system will wait for another transition name. When a previously declared transition name is entered, the corresponding transition is DELETEd from the transition list. ENETDIS then requests

NEW TYPE

expecting one of "F", "J", "T", "X", or "Y" to be entered by the user. The old transition screen positions is used but the system requests new labels by

LABELS

The user should then enter a new label list (i.e. transition name, input location names, output location names) delimited by blanks and terminated by a carriage return.

Display Directives

This class of ENETDIS commands controls the amount of information displayed on the screen with respect to the current E-net defined in the display data base. The system incorporates a display switch which allows the user to enable/disable the drawing process. The default value of the switch is "off", i.e. transition-location set creation does not cause an image to appear on the screen unless the DISPLAY command is involved.

DISPLAY (R) — The current screen image (including commands) is erased, and redrawn from the current display data base. The value of the display switch is not affected.

DRAWOFF (R) — Sets the value of the display switch to "off".

DRAWON (R) — Sets the value of the display switch to "on". Editing commands remain display-disabled regardless of the value of the display switch.

FLUSH (R) — Clears the display data base, thus enabling a new E-net to be created or edited.

GRID (R) — Displays the cartesian coordinates for the screen. The coordinate labels are divided by 100.

Permanent File Commands

The permanent file commands are included to enable the user to get and save E-nets on files in either binary or text formats. The system responds to these requests by

PF NAME,FORMAT

to which the user should respond with a legitmate KRONOS permanent file name

and "B" for binary format or "T" for text format. An incorrect format response will result in a program abortion. A zero (0) file name is used to escape the file command. TAPE1 is used as the local file for the GET and SAVE operations. Since the files are saved as private files, all GETS on a file must be done under the same user number as the SAVE.

GET (R) — Set the display data base to contain the E-net on the given permanent file. This command destroys the current display data base.

SAVE (R) — Save the current display data base on a permanent file. (The command actually replaces the named file).

PURGE (R) — Purge the named permanent file from the system. This command is equivalent to the KRONOS purge command.

Miscellaneous Commands

END (R) — Terminates ENETDIS and returns control to the KRONOS operating systems. The display program can be immediately reloaded and executed by entering "ENETDIS."

HELP (R) — Erases the screen and provides a synopsis of all commands and execution directions.

The following set of plates illustrates the use of some of the above commands.

Result of the HELP Command

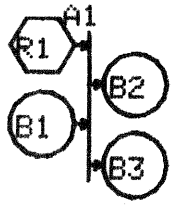
Table II.1

```
COMMAND ? HELP
COMMAND-
          CHANGE - CHANGE ALL BUT TYPE OF TRANSISTION (NC FOR NO CHANGE
          DELETE - DELETE TRANSISTION FROM NET
          DISPLAY - ERASE THE SCREEN AND REDRAW THE CURRENT NET
          DRAWOFF - TURN OFF THE DRAW SWITCH **DEFAULT VALUE**
          DRAWON - TURN ON THE DRAW SWITCH
          END - TERMINATE ENETDIS
          FLUSH - CLEAR PRESENT NET AND RESTART INPUT
          GET - RETRIEVE A PREVIOUSLY SAVED NET
          GRID - MARK THE GRID COORDINATES ON THE SCREEN
          HELP - PRINT THIS LIST
          LOCHANGE - CHANGE THE NAME OF A NODE IN NET
          PURGE - PURGE A PREVIOUSLY SAVED NET
          REPLACE - REPLACE A TRANSISTION WITH A NEW ONE AT SAME PLACE
          SAVE - SAVE A NET AS A PERMANENT FILE

TO RUN ENETDIS-
          SYSTEM-GET,ENET/UN=X652
          UNFMT
          CALL,ENET

TO RERUN THE PROGRAM, ENTER ENETDIS AND CR
```

Entering a Transition - Location Set

Table II.2

```
COMMAND ? DRAWON
COMMAND ? X
POS ? 300 300
LABELS? A1 R1 B1 B2 B3
COMMAND ?
```

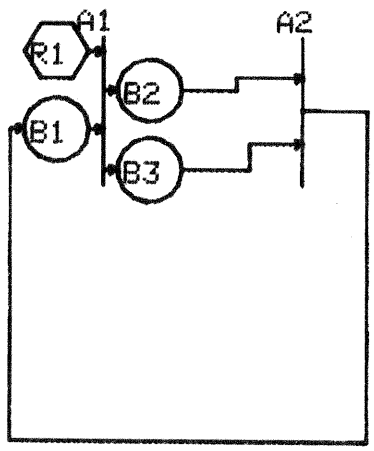


The Addition of Transition A2

Table II.3

(Note that the change command
does not affect the display)

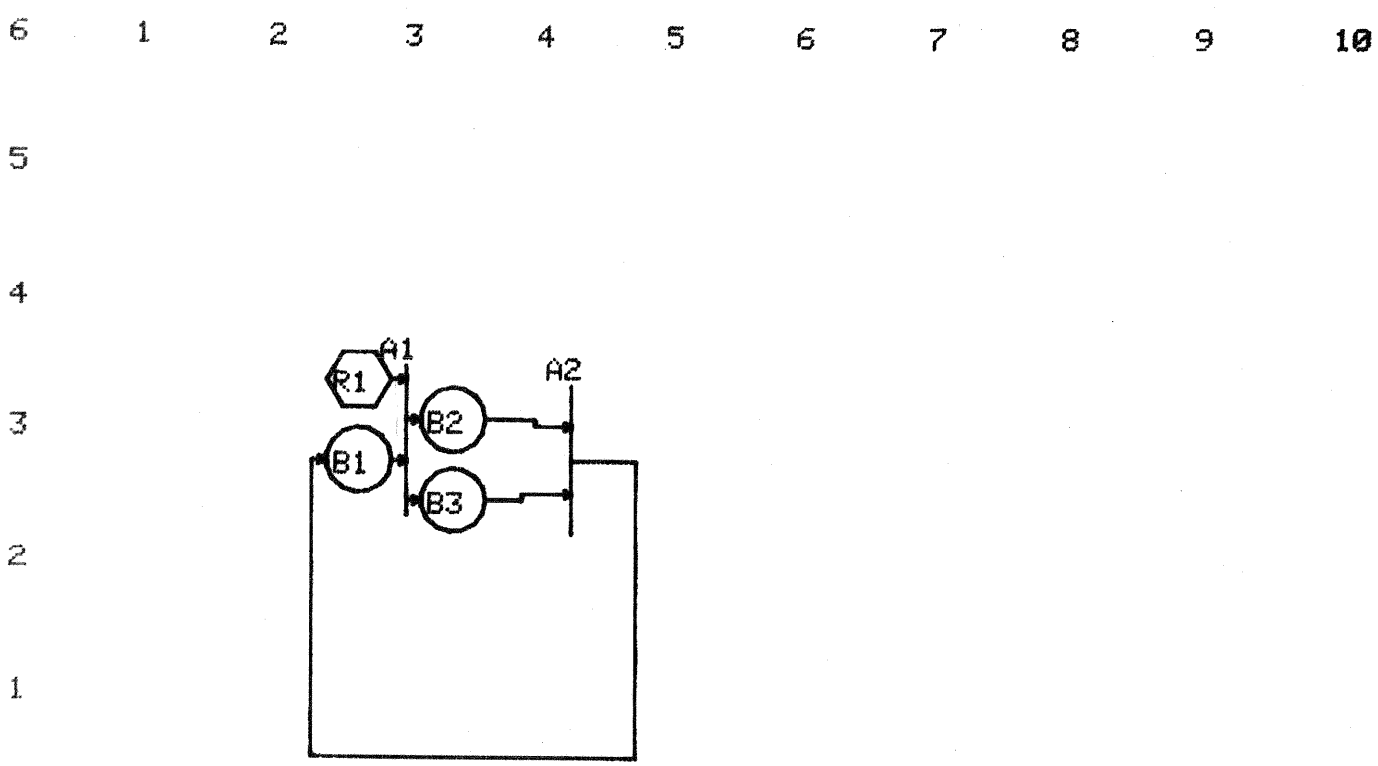
```
COMMAND ? J                    COMMAND ?
POS ? 450 300
LABELS? A2 B2 B3 B1
COMMAND ? CHANGE
TRANS NAME? A2
POS,LABELS? 425 285 NC NC NC NC
```



The Result of the CHANGE Command

Table II.4

```
COMMAND ? GRID
COMMAND ? SAVE
PF NAME,FORMAT? NET1 T
COMMAND ?
```



Text File Produced By SAVEing the Previous Graph

Table II.5

```
OLD,NET1
READY.
LNH

TYPE TRANS
     A1:   X(R1,B1,B2,B3);
     A2:   J(B2,B3,B1);
END.
READY.
```

III. THE COMPILER SUBSYSTEM

A. Capabilities and Limitations

The compiler subsystem, COMPLR, translates an E-net formal description into an abstract machine language which is ultimately executed by the interpreter abstract machine. The abstract machine program includes data structures to represent the set of locations, the set of transitions, the set of environment variables, and a body of two-address abstract machine instructions representing transition procedures, time expressions, and resolution procedures. When the compiler completes the translation process, the abstract machine program is written to a KRONOS file, where the ENSS user may dispose of the file (TAPE7) as he sees fit, (e.g. the file may be immediately executed by the interpreter, or it may be saved as a permanent file for later use). The compiler output is a complete data structure for interpretation purposes, but may also be treated as subnet and be linked to another subnet to form a composite abstract machine language program. The details of the linking process are discussed in Section IV.

The compiler translates an input string <evaluation net> as defined by the BNF syntax given in Appendix B. The compiler design has been influenced by classic table-driven compiler design. The major modules are a lexical analyzer, syntax analyzer, code generator, and output routine.

The line printer output from COMPLR includes a listing of the input stream along with error indicators for constructs that violate the syntax of the source language (see Table III.1). The existence of syntax errors can confuse the syntax analyzer to the point that subsequently compiled expressions will be interpreted as errors, even though they may be correct. This is not a serious liability, since the faulty compilation takes place only after a true syntax error has been encountered.

The current version of the compiler will handle a maximum of 160 locations, constants, and environment variables, or about 130 transition, (each transition description requires a variable number of entries, depending on the transition type; the transition description array elements are dynamically allocated by transition type). The maximum number of abstract machine instructions that can be generated per compilation is 256, (see Appendix C for a description of the abstract machine and language).

The "macro net capability" discussed elsewhere is currently not implemented in ENSS.[4] Subnets must be used to invoke repeated net patterns. It is also required that environment variables be limited to non-array values, i.e. an environment variable may not be declared as an array. The formal syntax accepted by the compiler is given in Appendix B. A summary of the differences between the ENSS implementation and the E-net formulation is given in Appendix A.

B. Using the E-net Compiler

The compiler subsystem resides on a KRONOS permanent file named COMPLR, and must be executed in the batch subsystem, either from a terminal or via the usual batch stream. The following octal memory requirements apply:

34700 words to compile

34400 words to load

24600 words to execute

This run-time field length allows for approximately 130 transitions, using expressions and procedures that compile to less than or equal 256

---

4. Noe, J. D. and Nutt, G. J., "Macro E-Nets for Representation of Parallel Systems," IEEE Transactions on Computers, Vol. C-22, No. 8 (Aug, 1973), pp. 718-727.

abstract machine instructions, (see Appendix C). The text file containing the net description must be on TAPE5 (which is not equivalent to the INPUT file). The load module is written to TAPE7 and a listing of the net is printed on the OUTPUT file device. Thus, the following control cards would compile the net found on KRONOS permanent file TXTNET (in text format), and the load module would be saved on the permanent file named LDMOD:

```
.
.
.
GET,COMPLR/UN=X652.
GET,TAPE5=TXTNET.
COMPLR.
SAVE,TAPE7=LDMØD.
```

The order in which various components of the net are described is rigid; any order of TYPE declarations other than that given here will result in compiler errors. A simple example (described later) is given in Table III.2. All input data are free field, with delimiters.

1. TYPE ␣ LOC: <d.list>;

   Mandatory location declaration list. The <d.list> is composed of individual location declarations for each location that appears in the net. Each declaration is delimited by commas, and the declaration list is terminated by a semicolon. A token with one or more attributes (i.e., an attribute token) is declared by stating the name followed by the number of attributes enclosed by parentheses. Simple tokens (with no attributes) are declared with no parenthesis notation.

2. TYPE ␣ PERIPH: ␣ INPUT=<list> ␣ OUTPUT=<list> ␣ RESØL=<list>;

   Mandatory declaration of input, output, and resolution peripheral locations. Input and output locations must have been previously declared in the TYPE ␣ LØC: section. A <list> is composed of location labels separated by commas and terminated with a blank. Any field

in the declaration may be omitted by deleting the key phrase and list. Each location declared to be an INPUT peripheral location will have a "generator net" concatenated to the input edge of the location This generator produces a constant saturation load of simple tokens to the corresponding INPUT location. Each OUTPUT peripheral location has an "absorber net" appended to its output location which removes a token from the corresponding location immediately after the token takes up residence. The absorber net contains attribute locations that maintain a running sum of corresponding attribute values on the output location and an additional attribute to count the number of tokens that consitute the attribute sums. Additional transitions and locations introduced to the net by the absorber/generator nets appear in interpreter output under the labels of ZZZZi, for $10 \leq i \leq 99$.

3. TYPE ␣ ENVIRØ: <list>;

An optional declaration of a <list> of simple environment variables (no arrays of environment variables are permitted). These variables may be referenced in any procedure.

In the following declarations, resolution procedures, time expressions, and transition procedures will use six "built-in" functions: EMPTY (loc), FULL (loc), RANDOM (seed), LOG (x), EXP (x), and EXTF (i,x). EMPTY and FULL evaluate to true or false, depending upon whether the argument location is empty or full, (e.g. EMPTY (B) = true if location B is empty, otherwise it evaluates to the value false). RANDOM (seed) samples a pseudo random number sequence with the given seed. LOG (x) evaluates to the natural logarithm of the argument X and EXP(x) evaluates to e^x. EXTF (i, x) is a function call to a user-provided FORTRAN function

to be loaded with the interpreter. Normally, i would be used to denote any one of a number of external functions to be evaluated, and x is the actual argument.

4. TYPE ␣ RESØL: R1: <rproc>; R2: <rproc>; ... Rn <rproc>;

Resolution procedures must be defined for each peripheral resolution location Ri declared in the TYPE ␣ PERIPH: declaration. Each <rproc> is a resolution procedure (formally defined in Appendix B) which provides conditions under which the resolution location can become set, reset, or undefined.

5. TYPE ␣ TRANS: A1: <schema>, <time expr>, <tr proc>;

A2: <schema>, <time expr>, <tr proc>;

⋮

Am: <schema>, <time expr>, <tr proc>;

END

Mandatory transition declaration section. The formal syntax for this implementation differs somewhat from the original syntax; differences are noted in Appendix A, and the syntax is given in Appendix B. The <time expr> or <tr proc> may be null, implying zero firing time and default transition procedures respectively. Otherwise, <time expr> is an arithemtic expression, evaluated before the transition fines, which defines the firing time. The <tr proc> defines a procedure to be executed at transition firing termination. The procedure should be written with the understanding that the tokens have already been moved. The <schema> defines the transition type and locations.

A summary of errors that the compiler can detect is given in Table III.1.

Suppose that it is desirable to create an E-net model of the following (trivial) system: there exists a finite queue with only three queue locations. The queue has a poisson arrival pattern with an aritrary mean interarrival time, i.e., the mean value is a simulation parameter. The queue is served by two identical processors, with the service time being uniformly distributed, again with an arbitrary mean value. Figure III.1 is a graph of the E-net (produced by ENETDIS), where locations B3, B4, and B5 represent the (first-come-first-served) queue. Location B6 represents the condition that server number 1 is busy, and a token on B7 represents the condition that server number two is busy. The text of the net might be given as shown in Table III.2.

Table III.1
Error Diagnostics

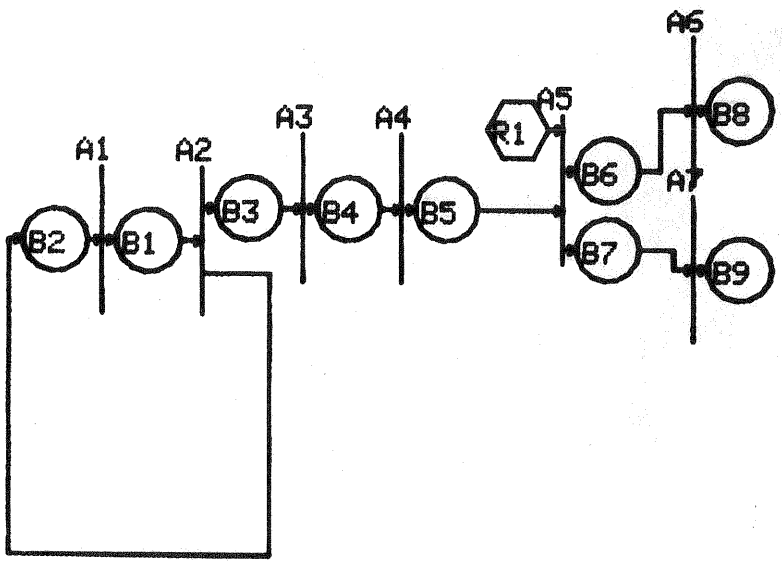
1. Left part of an expression must be a location or environment variable.
2. Store operator missing.
3. Left paren missing.
4. Location subscripts must be constants.
5. Right paren missing.
6. Colon missing.
7. Comma missing.
8. Relational operator missing.
9. Location name not a legal identifier.
10. Undeclared location or environment variable.
11. Expression evaluation stack overflow.
12. Incomplete conditional statement (THEN missing).
13. Semicolon or IF missing.
14. Semicolon or ELSE missing.
15. Semicolon missing.
16. Transition type violation.
17. Undeclared resolution location.
18. Unused.
19. Ill-formed Resolution location or environment variable.
21. Unused.
22. Resolution procedure missing.
23. Attribute reference to simple location.
24. Ill-formed vector-copy command.

Table III.2

```
TYPE LOC: B1(2),B2(2),B3(2),B4(2),B5(2),B6(2),B7(2),B8(2),B9(2);
TYPE PERIPH: OUTPUT=B8,B9 RESOL=R1;
TYPE ENVIRO: MNAR;
TYPE RESOL:
  R1: IF EMPTY(B7) THEN R1:=1 IF EMPTY(B6) THEN R1:=Ø;
TYPE TRANS:
  A1: T(B2,B1),,B1(1):=-MNAR*LOG(RANDOM(Ø));
  A2: F(B1,B3,B2),B1(1),B3(2):=B2(2)*RANDOM(Ø);
 A3: T(B3,B4);
 A4: T(B4,B5);
A5: X(R1,B5,B6,B7);
 A6: T(B6,B8),B6(2);
 A7: T(B7,B9),B7(2);
END
```

A Double Server Queue

Figure III.1



IV. THE LINK EDITOR

A. Capabilities and Limitations

The compiler subsystem produces a file (called the load module) containing all data structures in a format suitable for input to the interpreter. This set of data structures is relatively complex, and contains many cross-references between different parts of the data structure. (These cross-references can be thought of as being absolute addresses.) The link editor has been provided to combine a set of up to five distinct load modules, producing a single load module in the format suitable for input to the interpreter. Thus, the link editor must combine the various data structures, resolving new absolute addresses, and deleting absorber/generator nets from peripheral locations as defined by the net composition.

The load modules to be linked together are passed to the link editor on TAPE1,TAPE2,...TAPE5, and the combined-net load module is on TAPE98 when the link editor terminates. Directives for the linking process are entered via the usual INPUT file. The OUTPUT file will contain a listing of the directives, and a load map if requested by the MAP directive discussed in the usage section. Regardless of the output option chosen, edit warnings indicating location dimension specification conflicts will be written to the OUTPUT file.

The link editor is implemented to handle a resulting net with a maximum size as determined by the current version of the interpreter program. The size limitation arises due to the lack of a "paged memory" for the net description.

B. Usage

The link editor subsystem is saved on a permanent file named LINK. This file may be executed from a terminal or the card reader in the KRONOS batch subsystem. The following octal memory requirements apply:

34500 words to compile

51400 words to load

43100 words to execute

The link editor maintains a central memory image of the resulting combined load module; the load module is thus constrained to 320 locations, constants, and environment variables. The abstract machine code for all procedures must not exceed 1536 instructions (see Appendix C). The load module for the first subnet must appear on TAPE1, the load module for the second net must appear on TAPE2, etc. No more than five subnets may be combined on any one run of LINK. TAPE98 contains the combined load module when LINK terminates. Suppose that one wished to combine two subnet load modules on files named NETBI1 and NETBI3; the resulting net is to be saved on a file named BIGNET. The following sequence of KRONOS control cards will accomplish this task:

```
.
.
.
GET(TAPE1=NETBI3,TAPE2=NETBI1)
GET(LINK/UN=X652)
LINK.
SAVE(TAPE98=BIGNET)
.
.
.
```

The following link editor directives are passed to the program on the TAPE99=INPUT file. The order of directives given below is rigid, and must not be violated:

1. LINK (output net name)

   Mandatory directive which specifies the name of the net in load module format on TAPE98 when the program terminates.

2. MAP (α)

   Optional load map content indicator. If the MAP directive is omitted, no load map will be produced.

α=P Partial load map. A summary of the nets to be combined is given,

i.e., names, number of involved peripheral locations, the appended character for unique transition/location names, and table sizes. A similar summary is provided for the resulting net composition.

∝=F Full load map. The net summaries discussed in the partial load map option are included with additional detail, e.g., additional subfield lengths in the location table are provided. A table of subnet interconnections summarizes location name, status, dimension, and orientation. This option also dumps the transition table and location table to the OUTPUT file. Details of these tables are discussed in the Appendix C.

3. SUBNETS (net_1=B_1, net_2=B_2,...,net_n=B_n)

Mandatory subnet declaration list, where $1<n\leq 5$. The net whose name is given by net_i is expected to be passed to the program on $TAPE_i$. B_i is an appendage character that prefixes each location name in the load module on TAPEi. For example, if B3 is produced on TAPE4 by the compiler, and NETNM=X is the fourth net name in the declaration list, any other directive must refer to the location as XB3.

4. $Bloc_i$ TØ $Bloc_j$

Mandatory directive to the link editor requesting that the output peripheral location $Bloc_i$ is to become the same location as the input peripheral location $Bloc_j$.

5. END

Mandatory directive to terminate the input string.

Table IV.1 is a composition of the example net in section III and a simple absorber net. The output option is full.

```
LINKAGE EDITOR
      FULL MAP:


LINK (COMBIN)
MAP(F)
SUBNETS (NETBI=A,ABSBI=B)
  AB8 TO BB1
END



EXTENT OF LINKED OUTPUT E-NET:

      TRANS TABLE SIZE       140

      LOCATION TABLE:
           (LOCATION) SIZE    18
           (EV/CONST) SIZE    13
           (OVERALL)  SIZE    31

      CODE TRIPLES NUMBER     159

      USAGE LISTS SIZE          0

      INVOLVED PERIPHERALS      2

EDIT WARNING:
     DIMENSION CONFLICT BETWEEN
       OUTPUT PERIPHERAL AB8    (  2)
       INPUT PERIPHERAL  BB1    (  3)
     DIMENSION   3 RETAINED FOR BB1


** NEW SUB-NET FILE COMBIN  **

      REFERENCED SUB-NETS:
         NETBI  ABSBI

      LINKED PERIPHERALS PER SUB-NET:
             1          1

      APPENDED CHARACTER PER SUB-NET:
             A          B

      EXTENT OF EACH SUB-NET:

                  TRANS   MID    LOC     CODE     USAGE
         SUB-NET  TABLE   LOC   TABLE   TRIPLES   LISTS

          NETBI     109    14     19       97        1
          ABSBI      73    10     18       64        1



LINKAGE STRUCTURE:
   SUBNETS PERIPH  STATUS  TO SUB-NET  PERIPH  DIMENSION
   NETBI   AB8     OUTPUT    ABSBI     BB1          2
   ABSBI   BB1     INPUT     NETBI     AB8          3
```

LINKED NET:

TRANSITION TABLE:

INDEX	TYPE	NAME	TRPR	FTIME	OCC/EN				
				A-LOC	A-TR	C-LOC	C-TR	B-LOC	B-TR	RESOL
1	1	AA1	12	0000000012	000000					
				2	10	1	10			
10	3	AA2	22	0000000024	000000					
				1	1	3	21	2	1	
21	1	AA3	31	0000000035	000000					
				3	10	4	30			
30	1	AA4	36	0000000042	000000					
				4	21	5	39			
39	4	AA5	41	0000470047	000000					
				5	30	6	51	7	60	14
51	1	AA6	55	0000000065	000000					
				6	39	10	69			
60	1	AA7	60	0000000072	000000					
				7	39	9	101			
69	2	BA1	68	0000000102	000000					
				11	92	12	80	10	51	
80	4	BA2	74	0017520110	000000					
				12	69	13	92	14	121	72
92	1	BA3	88	0000000126	000000					
				13	80	11	69			
101	2	AZZZ16	93	0000000133	000000					
				9	60	15	112	16	112	
112	1	AZZZ17	105	0000000147	000000					
				15	101	16	101			
121	2	BZZZ16	110	0000000154	000000					
				14	80	17	132	18	132	
132	1	BZZZ17	124	0000000172	000000					
				17	121	18	121			

LOCATION TABLE:

INDEX	STOR PTR	NAME	DIM/USE
1	9	AB1	00000002
2	7	AB2	00000002
3	18	AB3	00000002
4	29	AB4	00000002
5	38	AB5	00000002
6	47	AB6	00000002
7	49	AB7	00000002
8	59	AB8	00000002
9	68	AB9	01450002
10	11	BB1	00000003
11	7	BB2	00000003
12	9	BB3	00000003
13	20	BB4	00000003
14	22	BB5	01710003
15	89	AZZZ12	00000003

16	100	AZZZ13	00000003
17	53	BZZZ12	00000004
18	64	BZZZ13	00000004
19	16	AR1	00000001
20	0	AMNAR	00000000
21	2	A2	77777776
22	1	A1	77777776
23	0	A0	77777776
24	13	BR1	00000001
25	0	BTEST1	00000000
26	0	BTEST2	00000000
27	2	B2	77777776
28	1	B1	77777776
29	5	B5	77777776
30	0	B0	77777776
31	10	B10	77777776

USAGE LIST VECTOR:

INDEX	PACKED WORD
1	0000000000000000

V. The Interpreter Subsystem

A. Capabilities and Limitations

The E-net interpreter program, INTERP, implements the abstract machine discussed in Appendix C. The first level of the abstract machine handles token flow through the network which has been previously compiled, (and possibly link edited with other subnets), by COMPLR. The tokens represent the data structures which may have their values referenced and altered through transition procedures, resolution procedures, and integer-valued time expressions. Resolution procedures and time expressions can only reference the data structures when determining the token flow through a net. The second level of the abstract machine implements data structure access and modification.

INTERP reads a net in the internal format from the file named TAPE1; the usual situation would be that TAPE1 is the same file as TAPE7 produced by the compiler. The user must provide interpreter specifications on the INPUT file. INTERP then simulates E-net activity using discrete simulation techniques to handle transition firing, and interpreting data structure manipulation procedures. Each transition specification dictates an expression of the amount of time that a particular firing should remain in the active phase. The token flow through the net determines the time at which the transition should become active, i.e., the enabling time is determined by the token flow. The active phase is completed by the termination procedure for the transition, in which tokens are moved and the transition procedure is evaluated. Thus, transition simulation is natural using an event approach; the first event of a firing corresponds to the enabling of the transition firing sequence in the net.

The user has the capability of obtaining a full trace of the transition firing sequence, or he may suppress the trace and only receive statistics at simulation termination. The net can be exercised repeatedly using different initial markings; environment variable initial values are also controlled by the input specifications. The initial simulation clock time is an input parameter, as well as the simulation termination time. All arithmetic (except for clock manipulation) is floating point.

The current version of the interpreter provides summary information indicating total and average dwell time for each location, including the system-provided absorber/generator locations (labeled ZZZ's). The location throughput count is also provided. The absorber locations contain tokens with attributes that automatically sum the values of the corresponding output peripheral location and the respective attribute average values. Finally, the list of transitions that are active at simulation termination is printed on the OUTPUT file.

Improvements to the output of INTERP might include a set of histograms specified by the USER. It might also be possible to include a turnaround time and throughput rate specification for user-provided paths through an E-net.

A user-provided FORTRAN function subprogram may be loaded with INTERP to provide for extended capability in each particular simulation. The details for using the subprogram in E-net procedures was given in Section III.B.

A useful programming improvement to the current version of INTERP would be to incorporate overlays into the code; the initialization of the program requires a significant amount of space.

B. Usage

The interpreter subsystem resides on the KRONOS permanent file named INTERP, and must be executed in the batch subsystem, either from a terminal or via the usual batch stream. The following octal memory requirements apply:

34500 words to compile

47600 words to load

30200 words to execute

Note that these figures use a small (dummy) version of EXTF. Version 1 of INTERP provides sufficient memory to handle a maximum of 320 locations, constants, and environment variables; depending on the number of each type of transition allowed; this is equivalent to approximately 250 transitions. The procedures must compile to less than or equal to 1536 instructions. The net in compiled format is read from TAPE1; no listing (nor load map) is provided. Any output will be written to the TAPE6=ØUTPUT file. The following example is a set of control cards to load an E-net on KRONOS permanent file TAPE7 and to interpret the net with EXTF source code on the KRONOS permanent file named EXTCØDE.

```
.
.
.
RFL (40000)
GET(EXTCØDE)
RUN(I=EXTCØDE)
GET(INTERP/UN=X652)
COPY(LGØ,INTERP)
GET(TAPE1=TAPE7)
RFL(46000)
INTERP.
.
.
.
```

The following description of interpreter specifications is passed to the program via the TAPE5=INPUT file. The numerical order of specifiers is important, and must not be violated; the INTERPRET and STOP specifiers are mandatory, and all others are optional, (with default values as shown).

1. INTERPRET (net name)

   Required. Specifies the name of the evaluation net load module found on TAPE1.

2. ØUTPUT (∝)

   Optional, the default causes INTERP to print only the list of currently active transitions at simulation termination.

∝=P The output includes the default option plus summary statistics for each location. Total location dwell time, token throughput for each location, and the average residency time of each token on the location are provided. If the location is non-empty at termination time, the attributes of the resident token are also given. If the resident token is on an absorber location, average attribute values are also printed. The termination value of each environment variable is also printed.

∝=F All ∝=P output is given along with a full trace of the transition firing sequence, (i.e. the simulation time for each enabled phase and termination phase is given).

3. MARK

   Optional. Specifies the marking of the net.

   Location Bi(j) is marked with the following text entered after the MARK card.

   Bi:1=<value>,2=<value>,...,j=<value>;

   Location marking is terminated with a new card with END.

4. ENVIRØ

   Optional. Used to specify initial values for environment variables. To mark a variable named VAR, include the following after the ENVIRØ entry:

   VAR=<value>;

   This list is terminated by END.

5. START=<integer time>

   Optional. Default value is zero. Specifies the initial time assigned to the simulation clock.

6. STØP=<integer time>

   Required. Specifies the time at which the simulation terminates.

Table V.1 is a list of initial specification errors that INTERP can detect. Run time errors are self descriptive, and indicate errors such as token storage array overflow, empty event list (i.e. no transition can fire), etc.

Table V.2 is a sample listing for simulation of the net given in section III. The start time is 100, and the stop time is 400, and the output option is F (full).

Table V.1

Initial Specification Errors

1. INVALID DATA CARD--CALLED FROM DATCARD
2. <Name> LOCATION IS NOT IN THE SUB-NET (DATCARD)
3. <msg> IS IN AN INVALID NUMBERIC FIELD ON A DATA CARD (DATCARD)
4. INTERPRET INVALID INTERPRET (lfn) CARD (INTCARD)
5. INVALID MARK CARD (MARCARD)
6. <name> LOCATION IS NOT IN THE SUB-NET (MARCARD)
7. EXCESSIVE NUMBER OF INITIAL MARKINGS (MARCARD)
8. <name> mark specification is bad (MARCARD)
9. <msg> IS PART OF AN INVALID NUMBERIC FIELD ON MARK CARD (MARCARD)
10. <name> LOCATION HAS AN ATTRIBUTE THAT EXCEEDS THE DIMENSION (MARCARD)
11. ATTEMPT TO READ PAST END-OF-FILE FATAL ERROR (MEXT)
12. INVALID OUTPUT CARD (ØUTCARD)
13. INVALID START CARD (SSCARDS)
14. BAD NUMERIC FIELD IN START CARD (SSCARDS)
15. STOP TIME OMITTED (SSCARDS)
16. INVALID OUTPUT CARD (SSCARDS)
17. BAD NUMERIC FIELD IN START CARD (SSCARDS)

```
!INTERPRETER SPECIFICATIONS:


? INTERPRET(2SERVER)
 INTERPRET(2SERVER)
? OUTPUT(F)
 OUTPUT(F)
? MARK
 MARK
? B2: 1=0, 2=200;
 B2: 1=0, 2=200;
? END
 END
? ENVR←IRO
 ENVIRO
? MNAR=75
 MNAR=75
? END
 END
? START=100
 START=100
? STOP=400


STOP=400
```

```
Ø          SCHEDULE A1          FC=1   AT          100
Ø       ACTIVATE A1       FC=1   AT        100
Ø          SCHEDULE A2          FC=5   AT          301
Ø       ACTIVATE A2       FC=5   AT        301
Ø          SCHEDULE A3          FC=1   AT          301
Ø          SCHEDULE A1          FC=1   AT          301
Ø       ACTIVATE A1       FC=1   AT        301
Ø       ACTIVATE A3       FC=1   AT        301
Ø          SCHEDULE A2          FC=5   AT          308
Ø          SCHEDULE A4          FC=1   AT          301
Ø       ACTIVATE A4       FC=1   AT        301
Ø          SCHEDULE A5          FC=1   AT          301
Ø       ACTIVATE A5       FC=1   AT        301
Ø          SCHEDULE A6          FC=1   AT          380
Ø       ACTIVATE A2       FC=5   AT        308
Ø          SCHEDULE A3          FC=1   AT          308
Ø          SCHEDULE A1          FC=1   AT          308
Ø       ACTIVATE A1       FC=1   AT        308
Ø       ACTIVATE A3       FC=1   AT        308
Ø          SCHEDULE A2          FC=5   AT          323
Ø          SCHEDULE A4          FC=1   AT          308
Ø       ACTIVATE A4       FC=1   AT        308
Ø          SCHEDULE A5          FC=2   AT          308
Ø       ACTIVATE A5       FC=2   AT        308
Ø          SCHEDULE A7          FC=1   AT          343
Ø       ACTIVATE A2       FC=5   AT        323
Ø          SCHEDULE A3          FC=1   AT          323
Ø          SCHEDULE A1          FC=1   AT          323
Ø       ACTIVATE A1       FC=1   AT        323
Ø       ACTIVATE A3       FC=1   AT        323
Ø          SCHEDULE A2          FC=5   AT          351
Ø          SCHEDULE A4          FC=1   AT          323
Ø       ACTIVATE A4       FC=1   AT        323
Ø       ACTIVATE A7       FC=1   AT        343
Ø          SCHEDULE A5          FC=2   AT          343
Ø          SCHEDULE ZZZZ16      FC=4   AT          343
Ø       ACTIVATE ZZZZ16   FC=4   AT        343
Ø          SCHEDULE ZZZZ17      FC=1   AT          343
Ø       ACTIVATE ZZZZ17   FC=1   AT        343
Ø       ACTIVATE A5       FC=2   AT        343
Ø          SCHEDULE A7          FC=1   AT          518
Ø       ACTIVATE A2       FC=5   AT        351
Ø          SCHEDULE A3          FC=1   AT          351
Ø          SCHEDULE A1          FC=1   AT          351
Ø       ACTIVATE A1       FC=1   AT        351
Ø       ACTIVATE A3       FC=1   AT        351
Ø          SCHEDULE A2          FC=5   AT          352
Ø          SCHEDULE A4          FC=1   AT          351
Ø       ACTIVATE A4       FC=1   AT        351
Ø       ACTIVATE A2       FC=5   AT        352
Ø          SCHEDULE A3          FC=1   AT          352
Ø          SCHEDULE A1          FC=1   AT          352
Ø       ACTIVATE A1       FC=1   AT        352
Ø       ACTIVATE A3       FC=1   AT        352
Ø          SCHEDULE A2          FC=5   AT          413
Ø       ACTIVATE A6       FC=1   AT        380
Ø          SCHEDULE A5          FC=1   AT          380
Ø          SCHEDULE ZZZZ14      FC=4   AT          380
Ø       ACTIVATE ZZZZ14   FC=4   AT        380
Ø          SCHEDULE ZZZZ15      FC=1   AT          380
Ø       ACTIVATE ZZZZ15   FC=1   AT        380
Ø       ACTIVATE A5       FC=1   AT        380
Ø          SCHEDULE A4          FC=1   AT          380
Ø          SCHEDULE A6          FC=1   AT          467
Ø       ACTIVATE A4       FC=1   AT        380
```

```
1E-NET SIMULATION CONCLUSION AT          400
 START=      100    UP TIME=      300

LOCATION SUMMARY
      NAME  TSA PTR  NO      VALUE          TIME   THRUPUT     AVG VALUE
      B1            1                       504         6       84.0000
                        1      61.4146
                        2     200.0000
      B2            0                       100         5       20.0000
      B3            0                         0         5        0.0000
      B4            0                        28         5        5.6000
      B5           37                        49         5        9.8000
                        1       0.0000
                        2     103.3058
      B6           25                        79         2       39.5000
                        1       0.0000
                        2      87.7532
      B7           31                        35         2       17.5000
                        1       0.0000
                        2     175.2827
      B8            0                         0         1        0.0000
      B9            0                         0         1        0.0000
      ZZZZ10        0                         0         1        0.0000
      ZZZZ11        7                       280         1      280.0000
                        1       0.0000
                        (       0.0000)
                        2      79.7349
                        (      79.7349)
                        3       1.0000
                        (       1.0000)
      ZZZZ12        0                         0         1        0.0000
      ZZZZ13       13                       243         1      243.0000
                        1       0.0000
                        (       0.0000)
                        2      35.6016
                        (      35.6016)
                        3       1.0000
                        (       1.0000)
      R1         2300          0.0000
      MNAR       2301         75.0000
      2          2302          2.0000
      1          2303          1.0000
      0          2304          0.0000


 EVENT SCHEDULE:

      TIME  TRANS      FIRE

        413  A2       A TO C,D

        467  A6        A TO C

        518  A7        A TO C
STOP
```

Appendix A

Implementation versus Design

The most significant deviation of the implementation of evaluation nets from their design is in the syntax specification of the net. The implementation syntax is fully defined in Appendix B. This modification was necessary to make the use of ENSS easier, not the implementation. The original procedure declarations were similar to LISP conditional expressions, while the implementation employs conditional expressions similar to those found in ALGOL-60 programs. The spirit of the procedure declaration is preserved.

A serious deviation from the design is that resolution procedures are evaluated from right-to-left in the implementation, whereas the design of resolution procedure evaluation used a left-to-right process. For example, suppose

$\Psi(r_i)=r_i$: IF true THEN r_i:=0 IF true THEN r_i:=1;

evaluates r_i to 1 in the implementation and r_i to 0 in the design.

The design of evaluation nets indicated that during transition firing termination, the net programmer could assume that a copy of the token on the input location also existed simultaneously on the output location; the input location actually destroyed its token copy at the end of the transition procedure call. In the ENSS implementation, two copies do not simultaneously exist; the token is moved to the output location before the transition procedure is executed. The significance of this deviation is better understood by considering a transition procedure for a T transition, T(Bi,Bj):

```
Bj(3):=f(Bi(k))
```

will always set Bj(3) to zero in the implementation, whereas the design implies that the above expression is equivalent to:

```
Bj(3):=f(Bj(k)).
```

Environment variables must be simple variables in the ENSS implementation; arrays of environment variables cannot be employed by the net programmer.

Appendix B

Evaluation Net Syntax

Assume the net is given by $((L,P,R,A),M_0,(\xi,\Psi)$

<evaluation net>::=<location declaration section><Environment variable declaration section><Resolution procedure declaration section><transition declaration section>

<location declaration section>::=TYPE ␣ LØC:<declaration list>;<peripheral location declaration section>

<declaration list>::=<location variable>,<declaration list>|<location variable>

<peripheral location declaration section>::= TYPE ␣ PERIPH:<input location list><output location list><resolution location list>;

<input location list>::=INPUT=<simple location variable list>␣|null

<output location list>::=ØUTPUT=<simple location variable list> ␣|null

<resolution location list>::= RESØL=<simple rloc variable list>|null

<simple location variable list>::=<location name>,<simple location variable list>|<location name>

<simple rloc variable list>::=<resolution location name>,<simple rloc variable list>|<resolution location name>

<environment variable declaration section>::=TYPE ␣ ENVIRØ:<environment variable list>;|null

<environment variable list>::=<environment variable name>,<environment variable list>|<environment variable name>

<resolution procedure declaration section>::=

TYPE ␣ RESØL:<resolution procedure declaration list>

<resolution procedure declaration list>::=<resolution procedure declaration>

<resolution procedure declaration list>

|<resolution procedure declaration>

<resolution procedure declaration>::=

<resolution location name>:<conditional list>;

|<resolution location name>:<expression list>;

<transition declaration section>::=TYPE ␣ TRANS:

<transition declaration list>END

<transition declaration list>::=<transition declaration>

<transition declaration list>

|<transition declaration>

<transition declaration>::=<transition name>:<transition body>;

<transition body>::=<transition head><transition tail>|<transition head>

<transition tail>::=,<transition procedure>|null

<transition procedure>::=<expression list><conditional list><else clause>

|<conditional list><else clause>|<expression list>|null

<transition head>::=<T transition>|<and transition>|<or transition>

<T transition>::= T(<location name>,<location name>)<simple time expression>

<and transition>::= F<and schema>|J<and schema>

<and schema>::=(<location list>)<simple time expression>

<location list>::=<location name>,<location name>,<location name>

<or transition>::= X<or schema>|Y<or schema>

<or schema>::=(<resolution location name>,<location list>)<complex time expression>

<complex time expression>::=,(<right part>,<right part>)|null

<simple time expression>::=,<right part>|null

<conditional list>::=<conditional expression><conditional list>

|<conditional expression>

<conditional expression>::= IF<predicate>THEN<expression list>

<conditional tail>::=<expression list>ELSE<expression list>

|<expression list>

|<expression list><conditional expression>

<predicate>::=<boolean factor>|<predicate>v<boolean factor>

<boolean factor>::=<relation>|<boolean factor>∧<relation>

<relation>::=<right part><relational operator><right part>

|<simple predicate>

<relational operator>::=>|≥|=|≤|<|≠

<simple predicate>::= EMPTY (<location name>)|FULL(<location name>)

<expression list>::=<expression>,<expression list>|<expression>

<expression ::=<left part><right part>|<vector copy>

<vector copy>::=<location name>:=<location name>

<left part>::=<simple variable>:=

<right part>::=<term>|<add operator><term>|

<right part><add operator><right part>

<term>::=<factor>|<term><mult operator><factor>

<factor>::=<primary>|<factor>↑<primary>

<primary>::=<unsigned number>|<simple variable>|<function call>

<mult operator>::=*|/

<add operator>::= +|-

<location variable>::=<location name>(<unsigned integer>)

<location name> :=<identifer>εL

<resolution location name>::=<identifier>εR

<environment variable name>::=<identifer>εξ

<transition name>::=<identifer>εA

<simple variable>::=<location variable>|<environment variable name>

<identifer>::=<alphafirst><alpha><alpha><alpha><alpha>

<alphafirst>:: A|B|C|...|X|Y|Z

<alpha>::=<alpha first>|null

<function call>::=RANDOM(<right part>)|LOG(<right part>)
|EXP(<right part>)|EXTF(<right part>,<right part>)

<number>::=<unsigned number>|-<unsigned number>

<unsigned number>::=<unsigned integer>|<decimal fraction>
|<unsigned integer><decimal fraction>

<decimal fraction>::= .<unsigned integer>

<unsigned integer>::=<digit>|<unsigned integer><digit>

<digit>::= 0|1|2|...|9

Appendix C
ENSS Abstract Machine

ENSS employs an abstract machine to simulate the operation of an evaluation net. The abstract machine can be thought of as containing two levels, one to handle token movement through an evaluation net, and the other to evaluate time expressions and resolution and transition procedures.

The token machine directly simulates the net as defined by the transition schema and the initial marking. The compiler produces a location table and a transition table which totally describe the physical structure of the net and the identification of tokens with their values. The tables are not fully initialized until a marking for the net is declared; the interpreter handles this task and then immediately begins simulating token activity.

The expression evaluation abstract machine again uses the transition and location tables, as well as a token storage array. Figure C.1 shows the logical relationship among the various abstract machine components. Instruction words are composed of an operation code and a maximum of two operands. The "control unit" and the "arithmetic-logic unit" are combined into a single set of subroutines which fetch an instruction word from the JTRIP array, masks out the three fields, and executes the instruction on data obtained from the token storage array via the LOC array. The indirect addressing through the LOC array is useful for handling token movement and dynamic dimensioning. The token storage array is a linked list of five-word elements; whenever a token moves to a new location with a larger or smaller dimension, token array storage blocks are allocated or deallocated, respectively. JTRAN contains all information about the particular transition being simulated; thus it is used in resolution procedure evaluation for X or Y transitions and in time expression evaluation and transition procedure evaluation for all transitions. The machine evaluates expressions using a stack (called the REG array). The

result of an evaluation leaves the final result in REG(1), also called the accumulator, ACC. Operand types are assumed to be floating point in all operations.

Table C.1 is a dump of the load module produced by the compiler, given the net declaration of Table III.2.

The following list of instructions can be executed by the abstract machine:

(1)	Return	No operands. Indicates the completion of a procedure.
(2)	Set Logic	No operands. Sets the Logic Register to zero (false).
(3)	Branch on Zero, <address>	Branch to the <address> in the JTRIP array if the ACC is zero (false); else set the Logic Register non-zero (true) and continue executing instructions sequentially.
(4)	Unary Minus, <i>	ACC←-C(LOC<i>)*
(5)	Store, <i>,<j>	LOC<i>←C(LOC<j>) (If i<0 then LOC<i>≡ REG(-i))
(6)	Add, <i>,<j>	ACC←C(LOC<i>)+C(LOC<j>)
(7)	Subtract, <i>,<j>	ACC←C(LOC<i>)-C(LOC<j>)
(8)	Multiply, <i>,<j>	ACC←C(LOC<i>)*C(LOC<j>)
(9)	Divide, <i>,<j>	ACC←C(LOC<i>)/C(LOC<j>) (floating point quotient)
(10)	Exponentiate, <i>,<j>	ACC←C(LOC<i>)↑C(LOC<j>)
(11)	Logical AND, <i>,<j>	ACC←C(LOC<i>)∧C(LOC<y>)

*The expression "C(LOC i)" is a shorthand notation meaning the contents of the token attribute pointed to by LOC(i,1), where i is a location index.

(12)	Logical OR,<i>,<j>	ACC←C(LOC<i>)∨C(LOC<j>)
(13)	Less, <i>,<j>	ACC←if C(LOC<i>)<C(LOC<j>) then TRUE else FALSE
(14)	Less Equal, <i>,<j>	ACC←if C(LOC<i>)≤C(LOC<j>) then TRUE else FALSE
(15)	Equal,<i>,<j>	ACC←if C(LOC<i>)=C(LOC<j>) then TRUE else FALSE
(16)	Greater Equal,<i>,<j>	ACC←if C(LOC<i>)≥C(LOC<j>) then TRUE else FALSE
(17)	Greater, <i>,<j>	ACC←if C(LOC<i>)>C(LOC<j>) then TRUE else FALSE
(18)	Not Equal, <i>,<j>	ACC←if C(LOC<i>) ≠ C(LOC<j>) then TRUE else FALSE
(19)	Logic Check, <address>	if Logic Register ≠ 0 then goto <address>
(20)	Load, <i>	ACC←C(LOC<i>)
(21)	Create, <i>,<j,n>	Create a new token for LOC<i>; Copy n attributes from LOC<j> into LOC<i>.
(22)	Allocate, <m>,<i>	Assign m blocks of token storage to LOC<i>.
(23)	Deallocate, <m>,<i>	Deallocate m blocks of token storage from LOC<i>.
(24)	Destroy, <i>,<m>	Destroy (i.e. release token storage) the token on LOC<i>which contains m blocks.
(25)	Y-transition, <rloc>	LOC<rloc>, a resolution location, and ACC are set to zero if the the token is to be moved from the 0-input to the output location, and one if the token is to be moved from the 1-input to the output location.

(26)	No Operation	Null.
(27)	Move,<i>,<j>	Move the token from LOC<i> to LOC<j>.
(28)	Jump, <address>	Unconditionally branch to JTRIP(address) for the next instruction.
(29)	Load Clock	Loads ACC with the current value of the ENSS clock.
(30)	Vector Copy,<i>,<j>	Copies all attributes of the token on LOC<i> to LOC<j>.
(31)	Full Predicate,<i>	If LOC<i>contains a token, loads ACC with a non-zero value, otherwise ACC←0.
(32)	Empty Predicate, <i>	If LOC<i>does not contain a token, loads ACC with a non-zero value, otherwise ACC←0.
(33)	Random, ∅,<j>	The second operand is a constant located at LOC<j> which is used as the seed for a call to a pseudo random number generator. The result is left in ACC.
(34)	Natural Log, ∅,<j>	$ACC \leftarrow LOG_n(C(LOC<j>))$
(35)	Exponential Function, ∅,<j>	$ACC \leftarrow e^{C(LOC<j>)}$
(36)	External Function,<i>,<j>	Calls a user-provided FORTRAN function subprogram named EXTF with the following sequence: ACC←EXTF(FIX(C(LOC<i>)), C(LOC<j>))

```
1SUB-NET FILE TAPE 1

     (NTRAN,  MID, NLOC,NTRIP, NUSE, JFTR, JFLC, JFTP)
        109    14    19    97     1    69    10    63
ØLOCATION TABLE:

                  INDEX  NAME   (TR PTR)/DIM

 00000000000000000011    B1       0000000002
 00000000000000000007    B2       0000000002
 00000000000000000022    B3       0000000002
 00000000000000000035    B4       0000000002
 00000000000000000046    B5       0000000002
 00000000000000000057    B6       0000000002
 00000000000000000061    B7       0000000002
 00000000000000000073    B8       0001050002
 00000000000000000104    B9       0001310002
 00000000000000000105    ZZZZ10   0000000003
 00000000000000000120    ZZZZ11   0000000003
 00000000000000000131    ZZZZ12   0000000003
 00000000000000000144    ZZZZ13   0000000003
 00000000000000000220    R1       0000000001
 00000000000000000000    MNAR     0000000000
 00000000000000000002    2        7777777776
 00000000000000000001    1        7777777776
 00000000000000000000    0        7777777776
ØTRANS TABLE:

  INDEX  TYPE  NAME  TRPR      FTIME     OCC/EN
                      A-LOC  A-TR  C-LOC  C-TR  B-LOC  B-TR  RESOL

    1     1   A1      12   0000000012  000000
                    2      10        1     10
   10     3   A2      22   0000000024  000000
                    1      1         3     21      2     1
   21     1   A3      31   0000000035  000000
                    3      10        4     30
   30     1   A4      36   0000000042  000000
                    4      21        5     39
   39     4   A5      41   0000470047  000000
                    5      30        6     51      7     60     14
   51     1   A6      55   0000000065  000000
                    6      39        8     69
   60     1   A7      60   0000000072  000000
                    7      39        9     89
   69     2   ZZZZ14  65   0000000077  000000
                    8      51       10     80      11    80
   80     1   ZZZZ15  77   0000000113  000000
                   10      69       11     69
   89     2   ZZZZ16  82   0000000120  000000
                    9      60       12    100      13   100
  100     1   ZZZZ17  94   0000000134  000000
                   12      89       13     89
```

Table 3.1(3)

ØINSTRUCTION TABLE:

INDEX	OP	OPERAND1	OPERAND2
1	5	000000016	000000020
2	2	000000000	777777776
3	32	000000007	000000000
4	3	000000006	777777776
5	5	000000016	000000021
6	32	000000006	000000000
7	3	000000011	777777776
8	5	000000016	000000022
9	1	000000001	000000000
10	20	000000022	777777776
11	1	000000012	777777776
12	27	000000002	000000001
13	26	000000001	777777776
14	33	000000000	000000022
15	34	000000016	777777776
16	8	000000017	777777776
17	4	777777776	000000000
18	5	004010001	777777776
19	1	000000014	000000002
20	20	004010001	777777776
21	1	000000000	777777776
22	27	000000001	000000002
23	21	000000003	000010001
24	26	000000027	777777776
25	33	000000020	000000022
26	8	004020002	777777776
27	5	004020003	777777776
28	1	000000026	000000001
29	20	000000022	777777776
30	1	000000035	777777776
31	27	000000003	000000004
32	26	000000001	777777776
33	1	000000037	000000003
34	20	000000022	777777776
35	1	000000042	777777776
36	27	000000004	000000005
37	26	000000001	777777776
38	1	000000044	000000004
39	20	000000022	777777776
40	1	000000047	777777776
41	20	000000016	777777776
42	3	000000056	777777776
43	27	000000005	000000007
44	26	000000001	777777776
45	28	000000060	777777776
46	27	000000005	000000006
47	26	000000054	777777776
48	20	000000016	777777776
49	3	000000064	777777776
50	26	000000000	777777776

Table C.1(c)

```
 51    1 000000062 000000005
 52    1 000000056 000000005
 53   20 004020006 777777776
 54    1 000000000 777777776
 55   27 000000006 000000010
 56   26 000000001 777777776
 57    1 000000067 000000006
 58   20 004020007 777777776
 59    1 000000000 777777776
 60   27 000000007 000000011
 61   26 000000001 777777776
 62    1 000000074 000000007
 63   20 000000022 777777776
 64    1 000000077 777777776
 65   27 000000013 000000012
 66   26 000000001 777777776
 67    6 004010013 004010010
 68    5 004010012 777777776
 69    6 004020013 004020010
 70    5 004020012 777777776
 71    6 000000021 004030013
 72    5 004030012 777777776
 73   24 000000010 000000001
 74    1 000000010 000000013
 75   20 000000022 777777776
 76    1 000000113 777777776
 77   27 000000012 000000013
 78   26 000000001 777777776
 79    1 000000115 000000012
 80   20 000000022 777777776
 81    1 000000120 777777776
 82   27 000000015 000000014
 83   26 000000001 777777776
 84    6 004010015 004010011
 85    5 004010014 777777776
 86    6 004020015 004020011
 87    5 004020014 777777776
 88    6 000000021 004030015
 89    5 004030014 777777776
 90   24 000000011 000000001
 91    1 000000011 000000015
 92   20 000000022 777777776
 93    1 000000134 777777776
 94   27 000000014 000000015
 95   26 000000001 777777776
 96    1 000000136 000000014
0USAGE LISTS*

    1   00000000000000000000
```

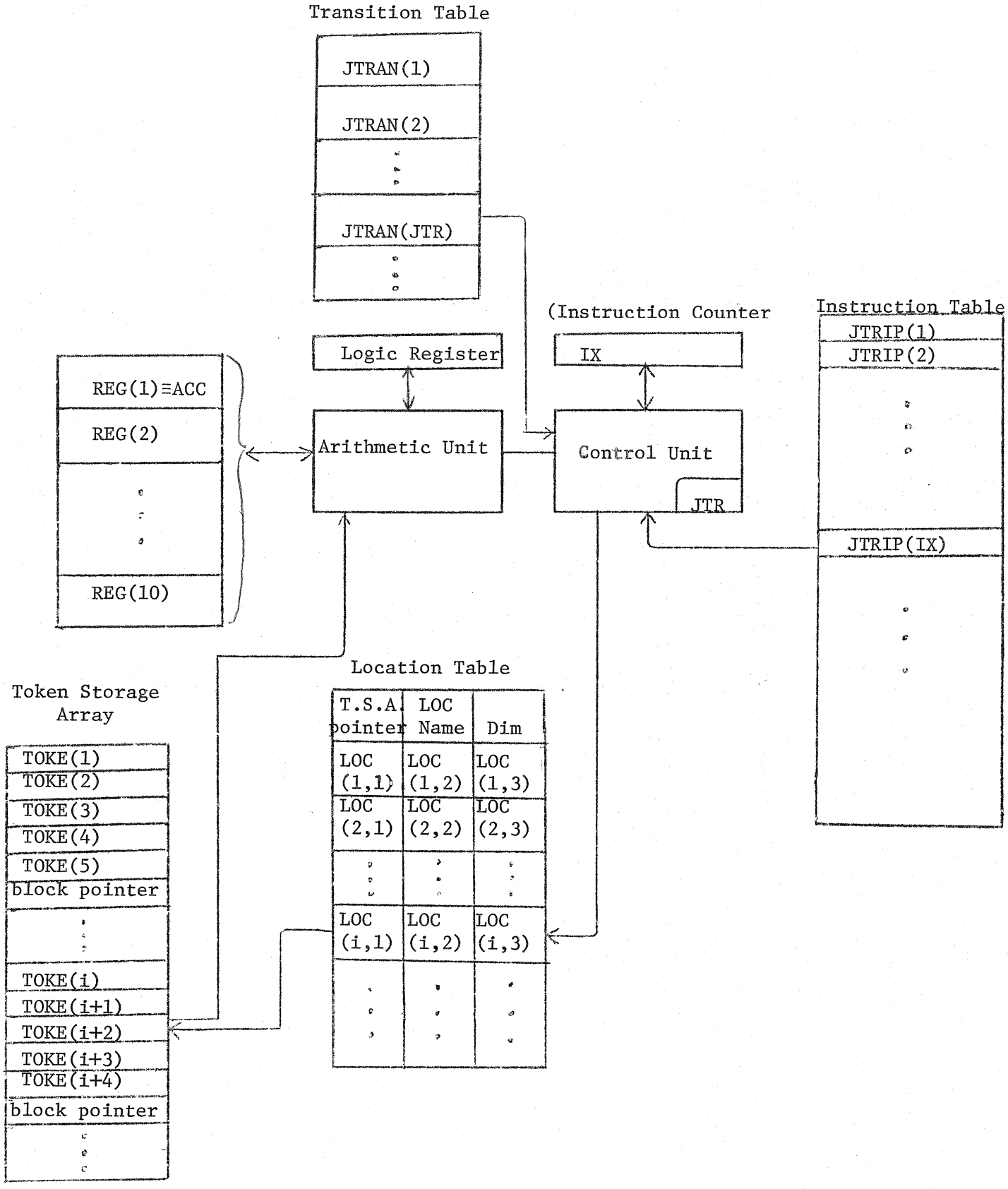

Transition Table
JTRAN(1)
JTRAN(2)
JTRAN(JTR)
(Instruction Counter
Instruction Table
JTRIP(1)
JTRIP(2)
JTRIP(IX)
Logic Register
IX
REG(1)≡ACC
REG(2)
REG(10)
Arithmetic Unit
Control Unit
JTR
Location Table
T.S.A. pointer
LOC Name
Dim
LOC (1,1)
LOC (1,2)
LOC (1,3)
LOC (2,1)
LOC (2,2)
LOC (2,3)
LOC (i,1)
LOC (i,2)
LOC (i,3)
Token Storage Array
TOKE(1)
TOKE(2)
TOKE(3)
TOKE(4)
TOKE(5)
block pointer
TOKE(i)
TOKE(i+1)
TOKE(i+2)
TOKE(i+3)
TOKE(i+4)
block pointer

Figure C.1