Wiley 0471763101

Simulation Using GPSS - Instruction Manual

Model: **0471763101** | Brand: **Wiley**

Introduction

This manual provides a comprehensive guide to understanding and utilizing the General Purpose Simulation System (GPSS). It is designed for students, researchers, and professionals interested in discrete-event simulation and its practical applications. The book covers fundamental concepts, advanced modeling techniques, and practical examples to facilitate learning and application of GPSS.

GPSS is a powerful simulation language used for modeling complex systems, allowing users to analyze system behavior, identify bottlenecks, and optimize processes before physical implementation. This manual serves as your primary resource for mastering GPSS.

Getting Started

How to Use This Manual

To effectively use this manual, it is recommended to read chapters sequentially, especially for those new to simulation or GPSS. Each chapter builds upon previous concepts, introducing new features and modeling constructs progressively. Practical examples are provided throughout to illustrate theoretical concepts.

Familiarity with basic programming logic and system analysis concepts will be beneficial, though not strictly required as the book aims to be self-contained for GPSS fundamentals.

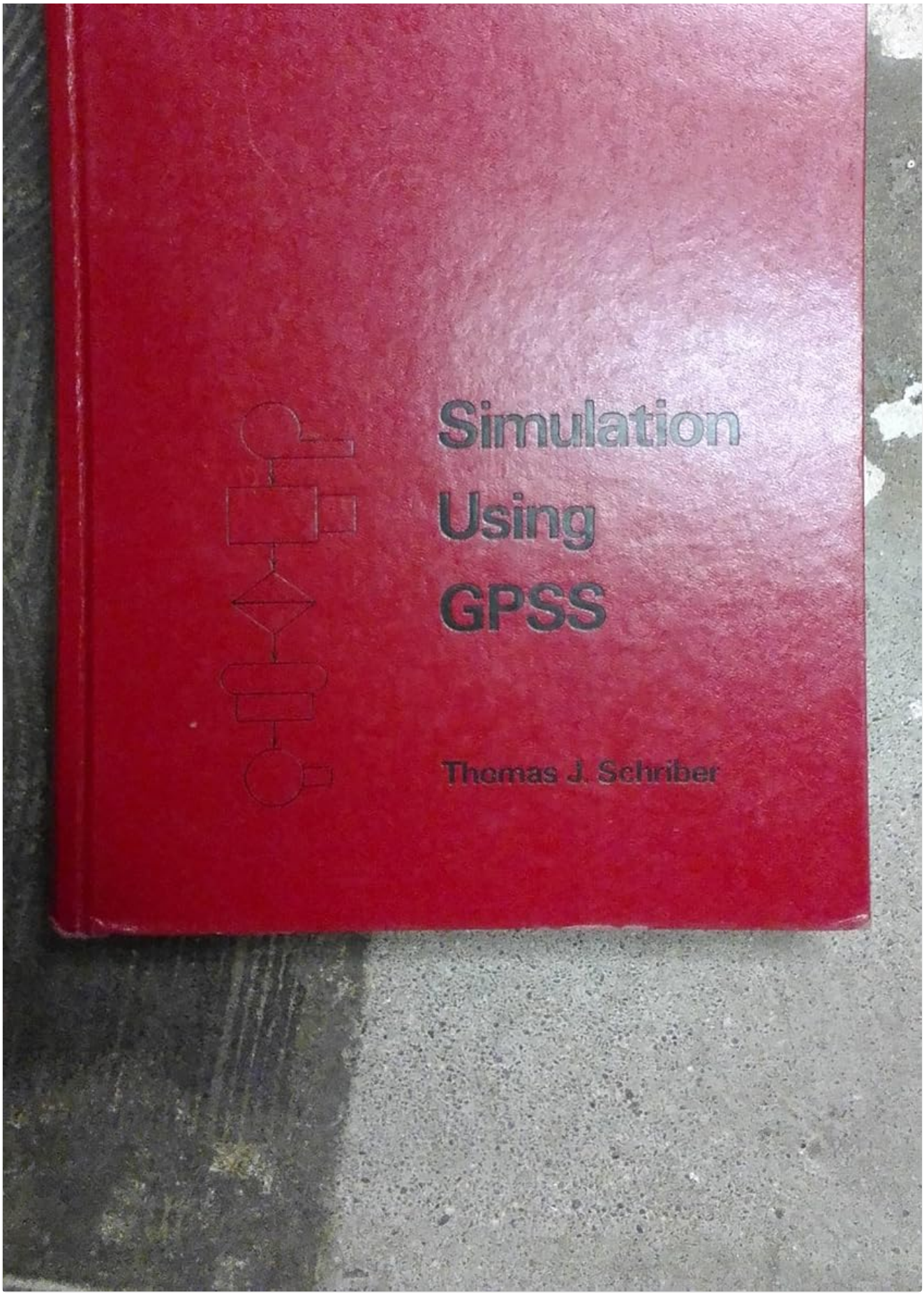


Figure 1: The front cover of the "Simulation Using GPSS" textbook, featuring the title, author Thomas J. Schriber, and a stylized flowchart graphic on a red background.

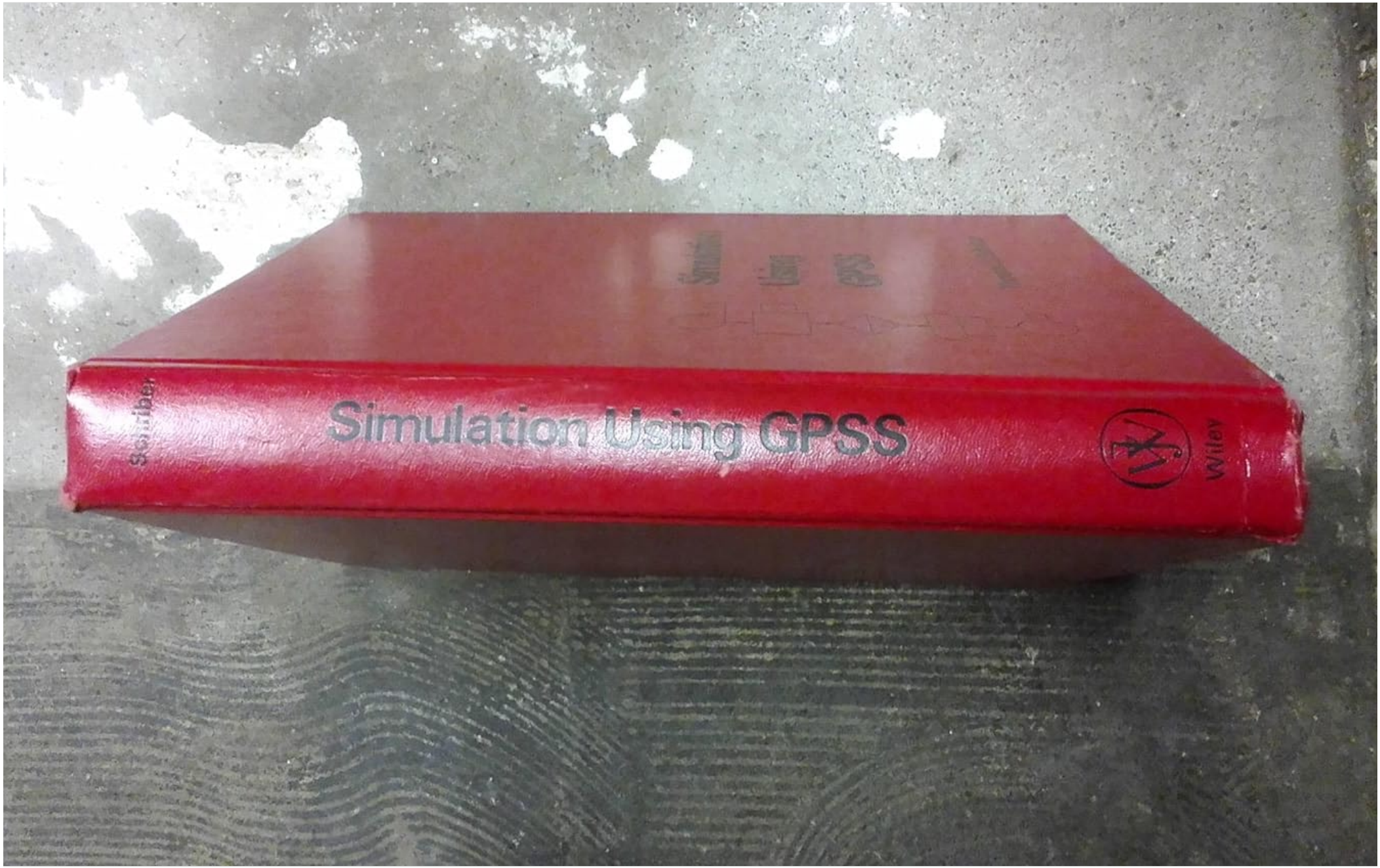

Simulation Using GPSS
Schriber
Wiley

Figure 2: The spine of the book, displaying "Simulation Using GPSS" and the publisher's logo (Wiley) at the bottom.

Figure 3: The plain red back cover of the book, showing minimal details.

Core Concepts and Application

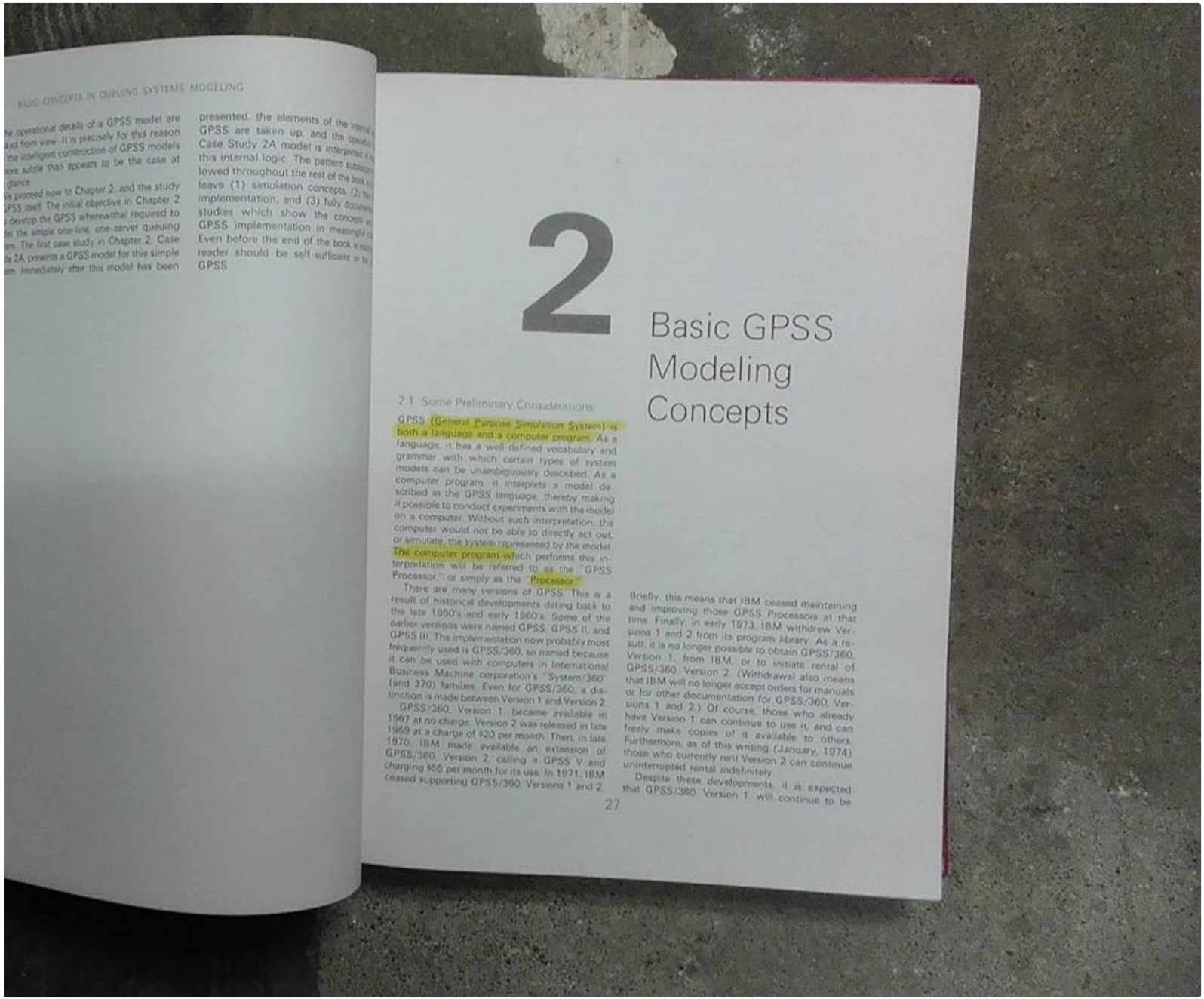
Understanding GPSS Elements

GPSS models are constructed using a set of predefined blocks, entities, and facilities. Transactions flow through these blocks, simulating the movement of items or information within a system. Key elements include:

- **Transactions:** Dynamic entities that move through the model, representing customers, parts, or data.
- **Blocks:** Static entities that perform operations on transactions, such as generating, seizing, delaying, or terminating.
- **Facilities:** Single-capacity servers that can be seized by one transaction at a time.
- **Storages:** Multi-capacity servers that can be entered by multiple transactions simultaneously, up to their capacity.
- **Queues:** Used to hold transactions waiting for a facility or storage.

Building Simulation Models

The process of building a GPSS model involves defining the system's components, their interactions, and the flow of transactions. This manual guides you through creating models from simple queuing systems to more complex manufacturing or service operations. Emphasis is placed on logical flow and accurate representation of real-world processes.



BASIC CONCEPTS IN QUEUING SYSTEMS MODELING

he operational details of a GPSS model are
ked from view. It is precisely for this reason
the intelligent construction of GPSS models
ore subtle than appears to be the case at
glance.
'e proceed now to Chapter 2, and the study
PSS itself. The initial objective in Chapter 2
develop the GPSS wherewithal required to
el the simple one-line, one-server queuing
ms. The first case study in Chapter 2, Case
dy 2A, presents a GPSS model for this simple
em. Immediately after this model has been

presented, the elements of the internal
GPSS are taken up, and the operation
Case Study 2A model is interpreted in
this internal logic. The pattern subsequ
lowed throughout the rest of the book
leaves (1) simulation concepts, (2) the
implementation, and (3) fully documen
studies which show the concepts and
GPSS implementation in meaningful
Even before the end of the book is reac
reader should be self-sufficient in
GPSS.

2 Basic GPSS Modeling Concepts

2.1 Some Preliminary Considerations

GPSS (General Purpose Simulation System) is both a language and a computer program. As a language, it has a well-defined vocabulary and grammar with which certain types of system models can be unambiguously described. As a computer program, it interprets a model described in the GPSS language, thereby making it possible to conduct experiments with the model on a computer. Without such interpretation, the computer would not be able to directly act out, or simulate, the system represented by the model. The computer program which performs this interpretation will be referred to as the "GPSS Processor," or simply as the "Processor."

There are many versions of GPSS. This is a result of historical developments dating back to the late 1950's and early 1960's. Some of the earlier versions were named GPSS, GPSS II, and GPSS III. The implementation now probably most frequently used is GPSS/360, so named because it can be used with computers in International Business Machine corporation's "System/360" (and 370) families. Even for GPSS/360, a distinction is made between Version 1 and Version 2.

GPSS/360, Version 1, became available in 1967 at no charge. Version 2 was released in late 1969 at a charge of $20 per month. Then, in late 1970, IBM made available an extension of GPSS/360, Version 2, calling it GPSS V and charging $55 per month for its use. In 1971, IBM ceased supporting GPSS/360, Versions 1 and 2. Briefly, this means that IBM ceased maintaining and improving those GPSS Processors at that time. Finally, in early 1973, IBM withdrew Versions 1 and 2 from its program library. As a result, it is no longer possible to obtain GPSS/360, Version 1, from IBM, or to initiate rental of GPSS/360, Version 2. (Withdrawal also means that IBM will no longer accept orders for manuals or for other documentation for GPSS/360, Versions 1 and 2.) Of course, those who already have Version 1 can continue to use it, and can freely make copies of it available to others. Furthermore, as of this writing (January, 1974) those who currently rent Version 2 can continue uninterrupted rental indefinitely.

Despite these developments, it is expected that GPSS/360, Version 1, will continue to be

Figure 4: An open page from the book, displaying the title for "Chapter 2: Basic GPSS Modeling Concepts," indicating the foundational content covered early in the manual.

(6) Program Output

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	1.192	101	23	22.7	27.594	35.730		
AREA2	5	2.158	77	2	11.7	607.764	583.796		

AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(a)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	1.633	198	66	23.2	29.580	38.532		
AREA2	5	1.400	24	5	17.0	669.035	564.381		

AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(b)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	1.550	296	85	28.7	27.653	38.232		
AREA2	5	1.500	42	8	14.2	537.039	626.777		

AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(c)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	1.596	393	111	28.2	28.604	39.946		
AREA2	5	1.263	53	11	20.7	457.807	577.765		

AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(d)

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
AREA1	3	1.268	493	147	29.8	25.618	37.927		
AREA2	5	1.439	72	11	15.2	491.620	581.525		

AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES

(e)

FIGURE 2F.4 Selected Program Output for Case Study 2F. (a) Queue statistics after one day. (b) Queue statistics after two days. (c) Queue statistics after three days. (d) Queue statistics after four days. (e) Queue statistics after five days.

(7) Discussion

Model Implementation. The model was run for five consecutive 8-hour working days, with aggregate statistics measured from the start of the simulation being printed out at the end of each day. In the Extended Program Listing in Figure 2F.3, note how this effect is accomplished with the timer-Transaction GENERATE Block (Card 28) and the START card (Card 31). Through the A and C Operands on the START card, the Termination Counter and Snap Interval Counter are initialized with values of 5 and 1, respectively. A timer-Transaction enters the model every 480 simulated minutes and immediately terminates, decrementing both of these counters by 1. For the first four timer-Transactions, this causes the Processor to (1) print out the standard model statistics, because the Snap Interval Counter has been decremented to zero, (2) restore the Snap Interval Counter's value to 1, and (3) continue the simulation because the Termination Counter is not yet zero. Finally, when the fifth timer-Transaction enters the model and terminates, the simulation is turned off and the standard printout is produced for the fifth and last time.

Program Output.[31] Referring to Figure 2F.4(e), the maximum number of television sets in the staging areas ahead of the original

[31] Total CPU time for the simulation on an IBM 360/67 computer was 4.1 seconds.

station (AREA1 Queue) and adjustment station (AREA2 Queue) did not exceed 3 and 5, respectively, for the five simulated days. These maximum contents had already been realized by the end of day 1 [Figure 2F.4(a)], suggesting that these statistics reached stable values during the simulation. In contrast, note the variation in the values of AVERAGE TIME/TRANS for the Queue AREA2 throughout the simulation.

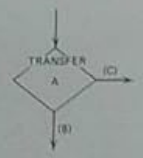
On day 1, 17.2 percent of the television sets going through inspection were routed to the adjustment station [as shown in Figure 2F.4(a), of 101 TOTAL ENTRIES to the AREA1 Queue, 99 went through inspection and 2 are still in the Queue; of the 99 going through inspection, 17 were sent to the AREA2 Queue]. During the 5 days, Figure 2F.4(e) information reveals that 14.2 percent of the inspections resulted in sets being sent on for adjustment.

2.46 Conditional Transfer of Transactions to One of Two Blocks

When the TRANSFER Block is used in statistical transfer mode, a Transaction's choice of next Block is independent of conditions currently in effect there. If the next Block chosen denies entry, the Transaction simply waits until permission to enter is granted. In an alternative mode, the TRANSFER Block can be used to send a Transaction to *whichever one of two Blocks will first accept it*. When applied this way, the TRANSFER Block is said to be used in BOTH mode. Specifications for this mode are shown in Figure 2.56.

The A Operand is literally the word BOTH. The B and C Operands are the names of two Block Locations in the model. A Transaction entering the TRANSFER Block in BOTH mode immediately attempts to move into the "B Block" (i.e., the Block at the Location whose name is supplied as the B Operand). If entry is denied, it then attempts to move into the "C Block." If that Block also denies entry, the Transaction is left on the Current Events Chain. From a Block-oriented point of view, the Transaction remains in the TRANSFER Block itself, where it contributes to the Current Count at that Block. At each next scan of the Current Events Chain, the Processor again attempts to move the Transaction into the B Block and, if entry is still denied there, into the C Block. Eventually, then, the Transaction moves into whichever one of the two Blocks will first accept it. Note that the B Block is tested first, then (if necessary) the C Block is tested. When both Blocks are simultaneously willing to accept the Transaction, then, it is the B Block which the Transaction enters.

As an example of BOTH mode use, assume that there are only three chairs in the waiting area of a barber shop. Customers arrive at the shop every 14 ± 7 minutes, but are willing to wait only if there is a chair for them to sit in. Otherwise, they leave and do not return later. Letting the Storage SEATS represent the seating capacity in the shop, Figure 2.57 shows a Block Diagram for a model to simulate the shop's operation. Customers entering the model first check to see if a seat is available in the waiting area. If it is, they stay; otherwise, they leave. Note that, in this example, the TRANSFER Block's "C



Operand	Significance	Default Value or Result
A	Literally, the word BOTH	Error
B	A Block Location ("B Block")	Sequential Block
C	Another Block Location ("C Block")	Error

FIGURE 2.56 The TRANSFER Block in BOTH Mode

Figure 5: A page from the book featuring a detailed GPSS flowchart diagram, illustrating the flow of transactions through various blocks in a simulation model.

8. Without using indirect addressing, show an alternative method for accomplishing what takes place when a Transaction moves through the Block sequence shown in Figure 6.7.

9. Without using indirect addressing, show an alternative method for accomplishing what takes place when a Transaction moves through the Block sequence shown in Figure 6.8.

10. As indicated in Chapter 3, the range of values taken on by the eight uniform random-number generators in GPSS is context-dependent, as summarized below.

    (a) When used as Function arguments, the random values are uniformly distributed on the closed interval between 0.000000 and 0.999999.
    (b) When used in any other context, the random values are uniformly distributed on the closed interval of integers between 000 and 999.

    This statement about context-dependency is valid whether the random-number generators are referenced via indirect specification (as RN*j*, where *j* = 1, 2, 3, . . . , 7, or 8), or via indirect addressing (as RN*k*, where *j* = 1, 2, 3, . . . , 98, 99, or 100). Subsequent to IBM's March, 1973, decommitment to the GPSS/360 Processor, it was unfortunately discovered that because of an error in the GPSS/360 code, *indirectly addressed* random-number generators used as *Function arguments* take on integer values between 000 and 999, inclusive. The argument of such a Function is then almost always out of range on the high side, i.e., is almost always 1 or greater. With this error in effect, the Function then almost always returns its *largest* possible value. Devise and run a GPSS model to determine if the indicated error exists in your GPSS Processor. If your Processor is in error, have the appropriate person in your computing center put a fix on it.

11. (a) Rows 1 through 10 in fullword Matrix 3 are to be assigned values returned by the GPSS random-number generators when they are used in the first context described in exercise 10. Values returned by RN*j*, *j* = 1, 2, 3, . . . , 8, are to be stored in columns 1, 2, 3, . . . , 8, respectively, of the Matrix. Show a Block Diagram segment which accomplishes this. Make the segment as compact as possible by using indirect addressing and looping. Run your model, and compare the entries in Matrix 3 with those shown in Table 3.1 in Chapter 3. (Hint: see exercise 7, Section 5.7.)
    (b) Repeat (a), only under the assumption that the random number generators are used in the second context described in exercise 10.

12. In early versions of GPSS, there were no SELECT Blocks. Assuming for the moment that there still are no SELECT Blocks, show a Block Diagram segment which has an effect equivalent to the segment shown in Figure E12. (The Block Diagram segment shown is a repetition of the segment in Figure 4.22.) Do this each of two different ways, as indicated below.
    (a) Do not use the LOOP Block in your segment.
    (b) Use the LOOP Block in your segment.

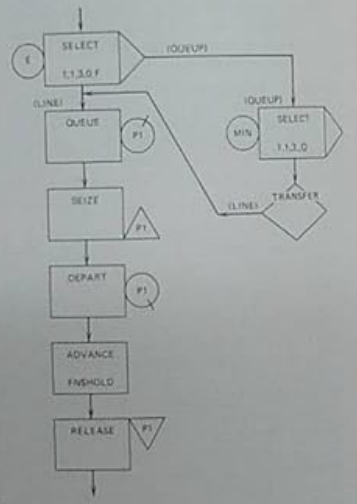


FIGURE E12

6.13 Attribute-Valued Functions

The three types of GPSS Functions which have been introduced so far are discrete, continuous, and list Functions. When these Functions are defined, *constants* are supplied as both the first and second members of the Function-defining ordered pairs of points. This means that each different value that such a Function can return is *directly specified* in the Function definition. (Of course, continuous Functions can also return values from the piecewise linear continuum which the Function definition describes.)

It is also possible to define *attribute-valued* Functions in GPSS. As in the case of discrete, continuous, and list Functions, the first member of each ordered pair of points used to define such a Function is a constant. In contrast, however, the second member of each ordered pair is a GPSS *Standard Numerical Attribute*. This means that such different value the Function can return is *indirectly specified* in the Function definition.

When an attribute-valued Function is defined, E is used as the first B-Operand character in the Function header card. Except for this, and the fact that Standard Numerical Attributes are used in the defining pairs of points, attribute-valued Functions are defined and used the same way as discrete, continuous, and list Functions.

An example of an attribute-valued Function is shown in Figure 6.9. The Function, named

```
GETUT  FUNCTION   P7,E4
9,11/10,FR3/11,FR12/12,FR7
```

FIGURE 6.9 A first example of an attribute-valued Function.

GETUT, has Parameter 7 as its argument. The Function returns as its value the fractional utilization of some Facility. For Parameter 7 values of 9, 10, 11, and 12, the Function's value is the fractional utilization of Facility 5, 3, 12, and 7, respectively.

For out-of-range arguments, the GPSS Processor treats attribute-valued Functions the same way it treats discrete Functions. For the Function GETUT, for example, if the value of P7 is less than 9, FR5 is returned as the Function's value; or, if P7's value exceeds 12, FR7 is returned as the value of the Function.

Figure 6.10 shows a second example of an attribute-valued Function. The Function's argument is random-number generator 5. The Function's name is SWICH. When the Function is called, its value equals the value of some other Function. The other Function is chosen with equal likelihood from the three Functions named JOE, PETE, and JIM.

```
SWICH  FUNCTION   RN5,E3
0,FNSJOE/.667,FNSPETE/1,FNSJIM
```

FIGURE 6.10 A second example of an attribute-valued Function.


```
MIXED  FUNCTION   X$CAROL,E4
21,V$RED/25,XH35/27,10/26,QSQUICK
```

FIGURE 6.11 A third example of an attribute-valued Function.

In Figure 6.11, an attribute-valued Function is shown which has a Savevalue as its argument. If X$CAROL has a value of 21 or less, the Function takes on the value of the Variable named RED; if X$CAROL is 22, 23, 24, or 25, the Function's value is XH35; and so on. This example shows that the second member of the Function-defining ordered pairs need not all be the same type of Standard Numerical Attribute. It also shows that one or more of the second members *can* be constants. In the example, for X$CAROL values of 26, or 27, the Function's value is simply 10.

Now consider attribute-valued Functions for which the first members of the Function-defining ordered pairs form the sequence of integers 1, 2, 3, etc. In such circumstances, attribute-valued Functions can be declared to be of the list type. List, attribute-valued Functions are defined by using M as the first B-Operand character in the Function header card. Nothing is gained *conceptually* by defining attribute-valued Functions to be of the list type, but increased speed of Function evaluation does result. And, as is the case with the list Functions introduced earlier, an error results during execution if the Function's argument is found to be out of range.

Figure 6.12(a) shows an example of an at-

```
POINT  FUNCTION   P$E5
1,V1/2,P2/3,X7/4,Q4/5,-9.9
```

(a)

```
POINT  FUNCTION   P$E5
1,V1/2,P2/3,X7/4,Q4/5,-9.9
```

(b)

FIGURE 6.12 An example of an attribute-valued Function, and its list Function equivalent. (a) The attribute-valued Function. (b) The equivalent list Function.

Figure 6: Another page from the manual, presenting a GPSS flowchart alongside explanatory text, demonstrating a specific simulation scenario and its block representation.

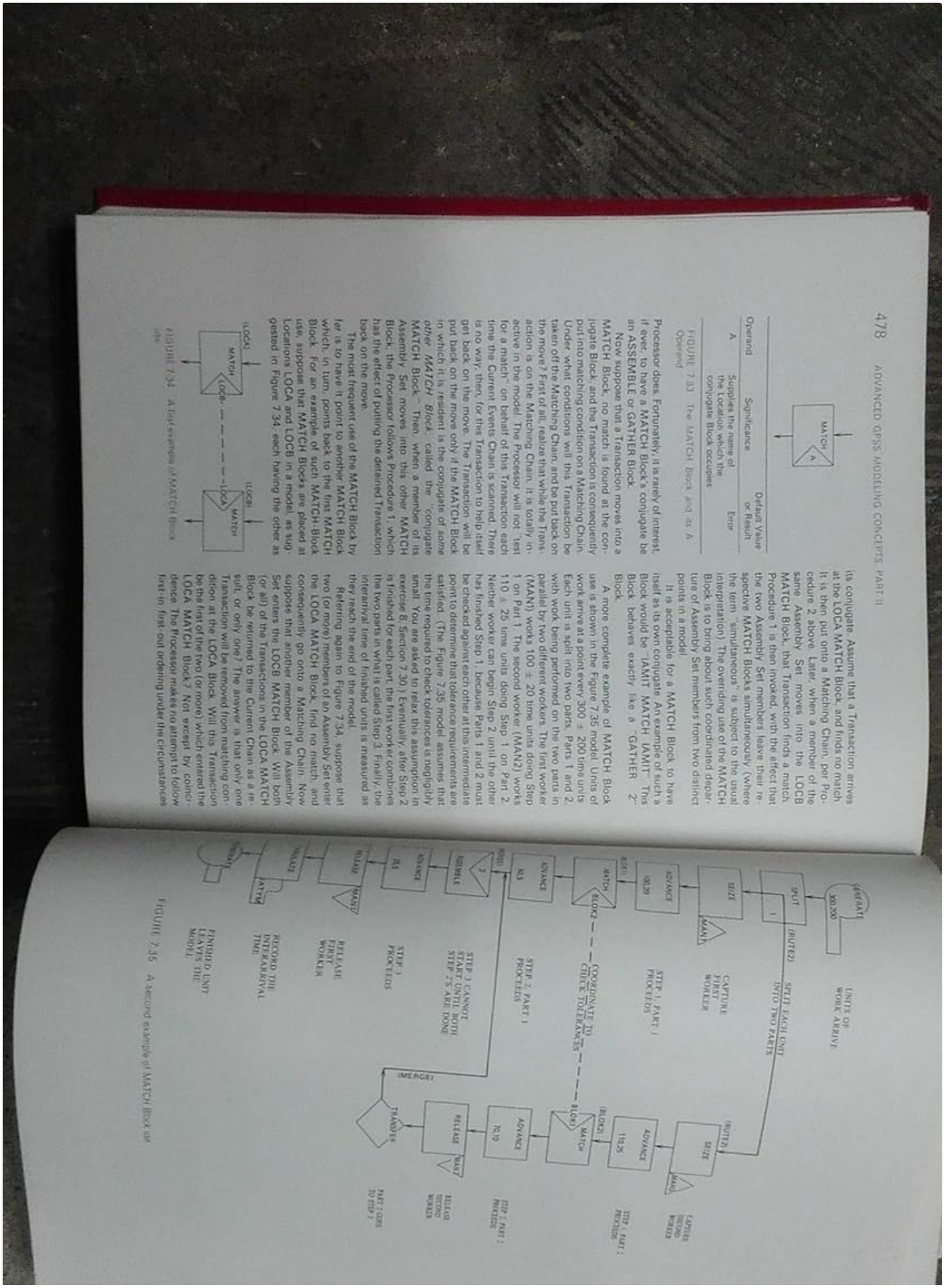


Figure 7: A page displaying a more complex GPSS model diagram, showcasing advanced block interactions and system logic.

Troubleshooting and Debugging

When developing GPSS models, errors can occur due to syntax mistakes, logical flaws, or incorrect parameter assignments. This section provides general guidance on identifying and resolving common issues:

- **Syntax Errors:** Carefully review block parameters and statement formats. GPSS is sensitive to correct syntax.
- **Logical Errors:** Trace transaction flow using debugging tools or by manually stepping through small models. Ensure that transactions are routed as intended and that resources are correctly seized and released.
- **Run-time Errors:** Pay attention to error messages generated by the GPSS simulator. These often point to issues like deadlocks, infinite loops, or resource contention.
- **Output Analysis:** If model results are unexpected, re-evaluate the input data and the assumptions made during model construction. Verify that the model accurately reflects the real-world system.

Refer to the specific GPSS software documentation for detailed debugging features and error code explanations.

Specifications

Attribute	Detail
Publisher	Wiley
Publication Date	August 16, 1974
Edition	1st
Language	English
Print Length	533 pages
ISBN-10	0471763101
ISBN-13	978-0471763109
Item Weight	3.44 pounds

Support and Further Information

For any inquiries regarding the content of this publication, errata, or academic use, please contact the publisher directly. While this book is a foundational text, newer editions or supplementary materials may be available from the publisher or related academic resources.

Publisher: Wiley

For general information about Wiley publications, visit their official website:www.wiley.com