ON THE APPLICATION OF USER CHAINS IN GPSS

THOMAS J. SCHRIBER
GRADUATE SCHOOL OF BUSINESS
THE UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN

ABSTRACT

The GPSS Processor uses a Current Events Chain, Future Events Chain, Interrupt Chains, and Matching Chains to support the logic of a GPSS simulation. These chains, which are an implicit part of the language, are automatically maintained and manipulated by the Processor as a simulation proceeds. At the analyst's option, one or more additional chains of a type known as User Chains can be explicitly incorporated into a GPSS model. These user-defined chains can be introduced for either one of two quite distinct reasons: (1) to decrease the execution time requirements of a given model, and (2) to implement queue disciplines other than first-come, first-served, within Priority Class. Despite their application scope, however, there are several subtleties associated with User Chain use. These subtleties arise principally because the GPSS Processor is inherently sequential in nature. This paper, presented in the spirit of a tutorial, explores User Chain applications and identifies some of the subtleties associated with their use.

1. Introduction

Those to whom this paper is addressed are assumed (a) to be active GPSS model-builders, thoroughly conversant with the operation of the Current and Future Events Chains in the language, but (b) without prior knowledge of the GPSS User Chain entity. The first assumption makes it possible to avoid starting the paper at too elementary a level. The second assumption provides an excuse to include here the fundamental User Chain groundwork needed to support some of the points to be made. Apart from these conveniences, are the assumptions realistic? The evidence suggests that they are. In much GPSS modeling, it is not necessary to apply User Chains (even though their application might be of advantage in decreasing the CPU time required for a simulation). Furthermore, the topic of User Chains is an "advanced" one in GPSS. The self-taught GPSS model-builder, then, can conveniently avoid getting into the User Chain concept. And the person who has "gone through a course" on GPSS may not have been told much, if anything, about User Chains, unless the course was either "long" or "advanced". Finally, there is no definitive treatment of User Chains anywhere in the literature on GPSS. They are introduced, yes, and the mechanics of using the two GPSS "Blocks" associated with them are spelled out, but there are few examples given for them. It is not unusual to see only one or two examples showing how User Chains can be applied when a constrained resource is being simulated with a single GPSS Facility. But this is the simplest, most straightforward application of User Chains. As such, it gives no hint of the subtle "simultaneity of events" complications which are associated with modestly more imaginative User Chain use.

There seems to be a need, then, for a more complete treatment of the GPSS User Chain entity. An attempt is made here to provide that treatment. In total, 7 different Block Diagrams, or Block Diagram segments, are presented and discussed to illustrate User Chain use in various ways [a]. Particular emphasis is given to the "simultaneity of events" problems that can occur in conjunction with User Chains. After the

[a] Portions of this material are taken from the manuscript for a book being written by Thomas J. Schriber (see reference [1]). As part of the manuscript, these portions have been copyrighted by Professor Schriber, and are reproduced here with his permission.

examples presented here have been studied, the GPSS model-builder should be able to apply User Chains creatively, and properly, in whatever contexts might be encountered in practice.

2. The Concept and Utility of User Chains

Whenever a Transaction encounters a blocking condition during a simulation, it is left, by default, on the Current Events Chain by the GPSS Processor. There are two disadvantages associated with this default Processor behavior.

(1) The CPU time required to simulate with the model may be larger than necessary. This is true even though the Processor makes a distinction between "unique", and "non-unique", blocking conditions in a model. The distinction is made because certain CPU time economies can be realized through the "Scan Indicator" concept whenever a blocking condition is unique. Transactions experiencing unique blocking are "scan-inactive" [b]. Even scan-inactive Transactions are processed at least one time, however, at each reading of the simulation clock. It is true that the only CPU time used to process scan-inactive Transactions is that required to test their Scan Indicators. In the long run, however, even this CPU time can be significant. Furthermore, when blocked Transactions are scan-active, the Processor attempts to move them into their next Block each time they are encountered in the scan, even though the logic of a given situation may make it evident (to the analyst, not to the Processor) that a blocking condition is still in effect. It should be clear, then, that if blocked Transactions could be made totally inactive in a model by removing them from the Current Events Chain, execution time economies could result.

(2) The second potential disadvantage concerns queue discipline. The ordering of blocked Transactions on the Current Events Chain is determined solely by their Priority Level, and the chronological sequence in which they were hooked onto that chain. This is why the default queue discipline in GPSS is "first-come, first-served, within Priority Class". If some other queue discipline were to be implemented, these steps would have to be performed.

(a) Instead of leaving waiting Transactions on the Current Events Chain, they would have to be removed from that chain and put "someplace else".

(b) Then, when the time came for one of them to move forward in the model (to capture a now-available server, for example), the Transaction brought from that "someplace else" and put back on the Current Events Chain could be selected by some criterion other than "first-come, first-served, within Priority Class".

In summary, there are two possible benefits to be realized if blocked Transactions can be removed temporarily from the Current Events Chain. The time required to simulate with a model can conceivably be decreased; and arbitrarily-defined queue disciplines can be implemented.

For the reasons cited, an entity known as "User Chains" has been made a part of the GPSS language. User Chains are a "someplace else" where Transactions can be when they are in a model, but are not on the Current Events Chain or one of the other "implicit" chains. Like the Current and Future Events Chains, User Chains have a "front" and a "back". But here the similarity stops. In the case of the Current and Future Events Chains, the GPSS Processor automatically moves Transactions to and from them, and maintains a pre-defined ordering property for Transactions on them. In the case of User Chains, Transactions are hooked onto them only according to logic explicitly provided by the analyst. Furthermore, the analyst can choose from several available alternatives to determine the position a Transaction is to occupy on a User Chain when it is placed there. In like fashion, Transactions are unlinked from User Chains and brought back into active status only according to the analyst's explicitly-provided logic. The analyst can also choose from a series of options in selecting the one or more Transactions which are to be unhooked from a User Chain and put back onto the Current Events Chain.

This overall logic of User Chain use is shown schematically in Figure 1. Blocks in the figure have been labeled A, B, C, D, E, and F. Blocks C, D, and E suggest the basic sequence followed to simulate use of a limited resource, such as that modeled with a Facility or Storage. C, D, and E might be a SEIZE-ADVANCE-RELEASE combination, or an ENTER-ADVANCE-LEAVE sequence. Block A represents the "look-ahead" feature of User-Chain logic, and block B indicates the consequence which follows when the look-ahead reveals that a blocking condition exists. Block F suggests how an active Transaction which has just removed a blocking condition causes a Transaction to be unlinked from a User Chain and brought back to the Current Events Chain, scheduled to make use of the now-available resource.

As might be expected, a pair of complementary GPSS Blocks is used to accomplish the User-Chain logic shown in Figure 1. One of these Blocks corresponds to the "linking logic" shown at A and B in the figure. The other performs the "unlinking logic" shown at F. It is essentially the use of these two Blocks which will be described in this paper.

User Chains have many of the same features as other GPSS entities. There can be many different User Chains in a model. Each chain can be named

---

[b] Familiarity with unique and non-unique blocking conditions and the Scan Indicator is assumed. For an explanation of these concepts, see sections 7.2 and 7.3 in reference [1].

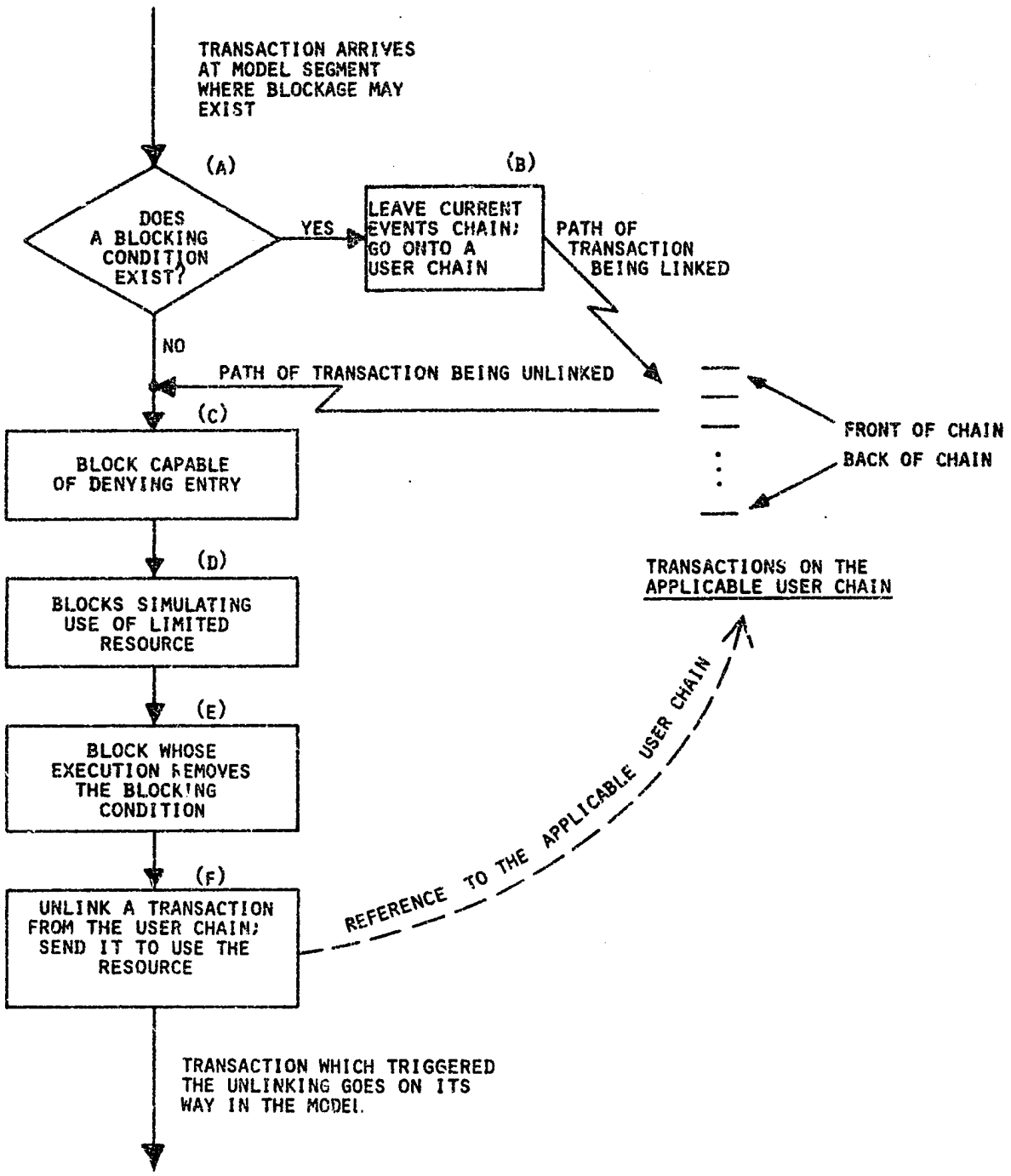


Figure 1 A Schematic Representation of the Logic of User Chain Use

either numerically, or symbolically, according to the usual rules. The number of different User Chains permissible depends on the amount of computer memory available to the Processor. Like Facilities, Storages, Queues, Tables, Blocks, etc., User Chains have a set of Standard Numerical Attributes associated with them. Furthermore, a set of User Chain statistics much like those for Queues appears as part of the standard output produced at the end of a simulation.

Like the Current and Future Events Chains, non-empty User Chains are printed out by the Processor at the end of a simulation only if "1" is used as the D Operand on the START Card. The PRINT Block can also be used to print out User Chains. For this purpose, the Block's A and B Operands indicate the smallest and largest numbers, respectively, of the User Chains which are to be printed out. The Field C mnemonic is CHA. When a Transaction moves into the Block "PRINT 2,5,CHA", then, User Chains 2 through 5 are printed out as a result.

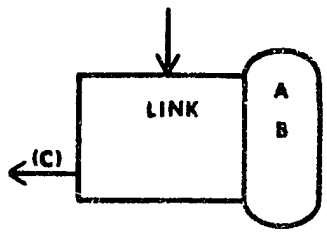
3. Transaction Movement to and from User Chains: The LINK Block and the UNLINK Block

The ability to put a Transaction onto a User Chain is provided with the LINK Block. The LINK Block can be used in either one of two modes: conditional mode, or unconditional mode. A conditional-mode LINK Block plays the roles of blocks A and B in Figure 1; that is, it embodies a certain "look-ahead" feature, as suggested by block A in Figure 1, and it has the capability of either sending a Transaction to capture an

available server, or of putting a Transaction onto a User Chain if there is no available server. In contrast, an unconditional-mode LINK Block has no effective look-ahead capability; it therefore plays only the role of block B in Figure 1. Transactions which enter an unconditional-mode LINK Block are always put onto a User Chain as a consequence.

Because use of the LINK Block in unconditional mode is the easiest to understand, this usage mode will be discussed first. Consider Figure 2, which spells out the specific details associated with the LINK Block. As indicated in that figure, when no C Operand is supplied for the LINK Block, the Block is being used in unconditional-linkage mode. (In fact, if the C Operand were eliminated from the LINK Block in Figure 2, the path leading from the LINK Block would be eliminated, too.) When a Transaction moves into such a LINK Block, it is placed on the User Chain whose name is supplied by the Block's A Operand. The position an incoming Transaction takes on the User Chain is governed by the LINK Block's B Operand. The four-character B Operands FIFO (First-In, First-Out) and LIFO (Last-In, First-Out) cause the Transaction to be placed on the back or front of the chain, respectively. If the B Operand is Pj, where j is some integer from 1 to 100, Transactions are arranged on the User Chain in order of increasing Pj value [c]. Each incoming Transaction is placed ahead of those chain residents which have a higher Pj value, but behind those which have a lower Pj value. In case of ties, the incoming Transaction goes behind other residents having the same Pj value. For example, suppose that Transactions A, B, and C have P3 values of -4, 21, and 32, respectively, and they enter the Block "LINK HOLD,P3". Then, after the linking, Transaction A is at the front of the User Chain HOLD, B is behind it, and C is at the back of the chain. If Transaction D now enters the LINK Block and has a P3 value of 21, it is placed between Transactions B and C on the User Chain.

Linking is conditional when the LINK Block's C Operand is used. A Transaction moving into a conditional-mode LINK Block will either be placed on the User Chain, or will be routed to the "C Block", i.e., the Block in the Location whose name is supplied by the C Operand. In practice, the "C Block" often turns out to be the Block which is sequential to the LINK Block in the model. But even when this is the case, the analyst must use the C Operand on the conditional-mode LINK Block, and must attach the corresponding Location Name to the sequential Block. There is no requirement, however, that the "C Block" be sequential to the LINK Block. This explains why there is a horizontal path leading from the Fig-



Operand	Significance	Default Value or Result
A	Name (numeric or symbolic) of a User Chain	Error
B	Specifies where the Transaction is to be placed on the User Chain; there are three possibilities.	Error
	B Operand: FIFO — Go on the back of the chain	
	B Operand: LIFO — Go on the front of the chain	
	B Operand: Pj — Merge into the chain immediately ahead of the Transaction with the next higher value of Parameter j	
C	Optional Operand; Block Location to which the Transaction moves if it is not linked onto the User Chain	Transaction is linked unconditionally onto the User Chain

Figure 2 The LINK Block and Its A, B, and C Operands

[c] Some caution is required here. When the LINK Block's B Operand is Pj, the "P" simply signals to the Processor that the linking criterion is "ordered according to the value of a Parameter". The number of the Parameter is directly specified, and is j itself. If a given LINK Block has P10 as its B Operand, then, the linking criterion is "ordered according to the value of Parameter 10", not "ordered according to the Parameter whose number can be found in Parameter 10".

ure 2 LINK Block, instead of a vertical path.

Nothing has been said yet about what determines whether a Transaction entering a conditional-mode LINK Block takes the C-Block exit, or is linked onto the referenced User Chain. To choose between these two possibilities, the GPSS Processor tests the setting of the referenced User Chain's Link Indicator. Each User Chain has its own Link Indicator. The indicator is either "on" ("Set"), or "off" ("Reset"). If the Link Indicator is "off" when a Transaction moves into a conditional mode LINK Block, the Processor does two things.

(1) It turns the Link Indicator "on".
(2) It does not link the Transaction onto the User Chain; instead, it routes the Transaction to the "C Block".

On the other hand, if the Link Indicator already is "on" when a Transaction enters a conditional-mode LINK Block, the Processor puts the Transaction onto the User Chain, and leaves the Link Indicator "on".

As indicated earlier, the unconditional-mode LINK Block corresponds precisely to block B in Figure 2. In this unconditional mode, the referenced User Chain's Link Indicator has no useful purpose. In contrast, the conditional-mode LINK Block takes on the roles played by blocks A and B in Figure 1. The referenced User Chain's Link Indicator embodies the "look-ahead" feature, and can be thought of much in the sense of a green-red traffic light. When the Link Indicator is "off", the traffic light is green. When a Transaction enters a conditional-mode LINK Block and finds that the traffic light is green, it interprets this as a "no blockage" signal. The Transaction moves ahead in the model, but before doing so, it switches the traffic light to red (Link Indicator "on") as a signal for later arrivals to the LINK Block. Conversely, if a Transaction arrives at the LINK Block and finds the traffic light is red (Link Indicator "on"), it interprets this to mean that blockage exists, and consequently goes onto the User Chain instead of moving ahead in the model.

The Link Indicator's look-ahead role cannot be fully appreciated until the Block complementary to the LINK Block has been described, and its effect on the Link Indicator's setting has been indicated. It might be mentioned now, however, that use of the Link Indicator for look-ahead purposes is extremely restricted. In fact, it is really useful as a built-in look-ahead device only when the limited resource which might offer blockage is simulated with a Facility. Most of the time, the analyst supplies his own look-ahead logic with a TEST or GATE Block at position A in Figure 1, and sends Transactions into an unconditional-mode LINK Block when the look-ahead reveals that a blocking condition exists.

The Block complementary to the LINK is the UNLINK. It is the UNLINK Block which is used at position F in Figure 1. The purpose of the UNLINK Block, of course, is to remove one or more Transactions from a User Chain and put them back on the Current Events Chain, so that the Processor can subsequently move them forward again in the model. By using appropriate UNLINK Block Operands, the analyst can specify which Transaction(s) on the User Chain qualify for unlinking. There are two broad possibilities here.

(1) Transactions can be removed from the front or from the back of the User Chain. In this case, Transactions "qualify" for unlinking simply by virtue of the position they occupy on the User Chain.

(2) Transactions can be removed from anywhere on the User Chain, providing that their properties satisfy analyst-specified conditions.

Only the possibilities indicated in (1) above will be described in this section. The possibilities indicated in (2) will be taken up in Section 7.

The UNLINK Block is shown with its various Operands in Figure 3. In considering the Block, it is important to distinguish between the Unlinker-Transaction (i.e., the Transaction which moves into the UNLINK Block, thereby initiating the unlinking operation), and the Unlinkee-Transaction (i.e., the Transactions being unlinked). When a Transaction enters the UNLINK Block, the Processor removes from the referenced User Chain the number of Transactions specified via the C Operand (assuming this many are on the User Chain to begin with, and that they satisfy the unlinking conditions). The C Operand, which can be a constant, a Standard Numerical Attribute, or ALL, is termed the "Unlink Count". If the C Operand is ALL, then all qualifying Transactions on the referenced User Chain will be unlinked. The UNLINK Block's B Operand indicates the Location of the Block to which each of the Unlinked Transactions is to be routed. The D and E Operands are used in combination to indicate from which end of the User Chain the Unlinked Transactions are to be taken. When neither Operand is used, Transactions are unlinked from the front of the User Chain. When BACK is used as the D Operand, and the E Operand is not used, Transactions are unlinked from the back of the User Chain.

The UNLINK Block's F Operand is optional. If it is not used, the Unlinker moves unconditionally from the UNLINK Block to the sequential Block. If used, the F Operand supplies the name of the non-sequential Location to which the Unlinker moves next if no Transactions were unlinked in the attempted unlink operation [d].

[d] For the two UNLINK Block D-E combinations in Figure 3, the condition "no Transactions were unlinked can arise if and only if the referenced User Chain is empty prior to the attempted unlinking. For the other UNLINK Block D-E Operand combinations to be discussed in section 7, the "no Transactions were unlinked" condition can occur even when there are Transactions on the User Chain at the time of the attempted unlinking.

Now consider the effect of the UNLINK Block on the referenced User Chain's Link Indicator. When the User Chain is empty at the time a Transaction moves into the UNLINK Block, the Processor switches that User Chain's Link Indicator "off". Using the traffice light analogy, this is equivalent to switching the traffic light from red to green. It is logical for the Unlinker to do this when it has just ceased to cause blockage at a point, and then discovers (because of the empty User Chain) that no other Transaction is currently waiting for the blockage to be removed. Later, when the next Transaction appears at the associated LINK Block, the green traffic light serves as a signal that it need not go on the User Chain. Instead, the Transaction will switch the light to red, then move ahead in the model without delay.

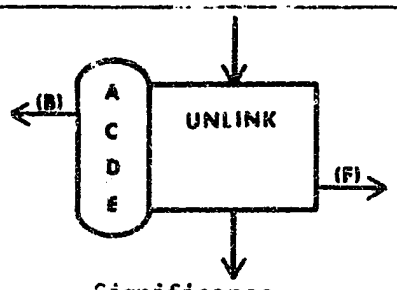
Control of a User Chain's Link Indicator can be summarized this way.

(1) The Link Indicator can be turned "on" (but never turned "off") at the LINK Block.

(2) The Link Indicator can be turned "off" (but never turned "on") at the UNLINK Block.

Consider next a chain-oriented interpretation of the way the UNLINK Block works. When a Transaction enters the UNLINK Block, the Processor removes Transactions from the referenced User Chain, one-by-one, placing each Transaction in turn on the Current Events Chain as the last member in its Priority Class. The Processor works from the front of the User Chain toward the back, unless the D-E Operand combination is "BACK; not used", in which case it works from the back toward the front. Execution of the UNLINK Block causes the Status Change Flag to be turned "on" if at least one Transaction is thereby unlinked [e]. When the UNLINK operation is complete, the Unlinker continues its forward movement in the model. This means that the Unlinked Transactions, if any, have not yet been processed. When the Unlinker finally comes to rest, the Processor tests the Status Change Flag and, if it is "on", turns it "off", and re-starts the scan of the Current Events Chain. This guarantees that, independent of their Priority Level, any Unlinked Transactions will be processed at the current reading of the simulation clock.

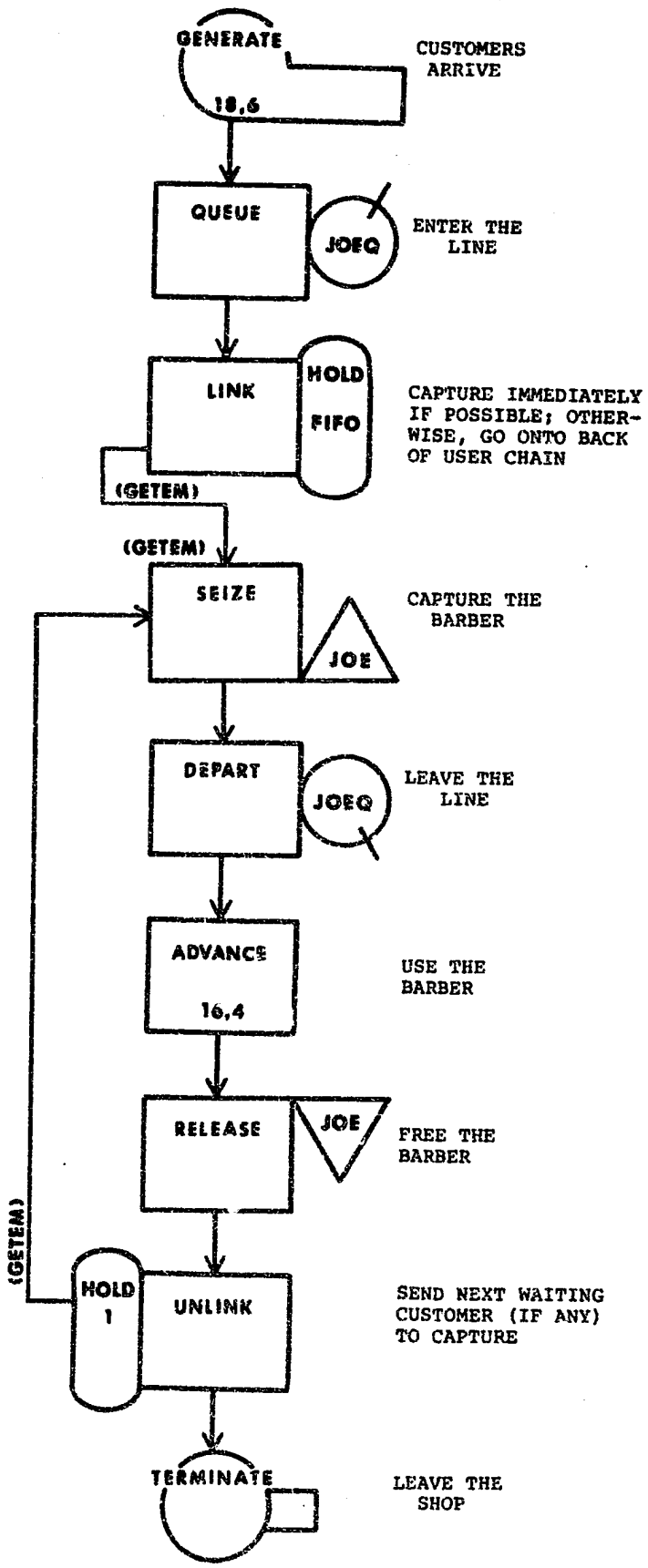
Finally, the relationship between User Chains and Block Counts should be carefully noted. When a Transaction is on a User Chain, it is not "in" any Block in the model. In particular, it is not "in" the LINK Block via which it was put onto the User Chain. Transactions on User Chains do not reflect themselves, then, in any fashion through Current Counts at Blocks. When a Transaction has just been unlinked from a User Chain and brought to the Current Events Chain, it would seem that it is also not yet "in" any Block. Conceptually,



Operand	Significance	Default Value or Result
A	Name (numeric or symbolic) of a User Chain	Error
B	Block Location to which the unlinked Transaction(s) is (are) to be routed	Error
C	The number of Transactions to be unlinked (the Unlink Count); can be a constant, a Standard Numerical Attribute, or ALL	Error
D and E	Specify which end of the User Chain Transactions are to be taken from, per this scheme:	Transactions are unlinked from the front of the User Chain
	D Operand: Not Used; E Operand: Not Used; End Indicated: Front End	
	D Operand: BACK; E Operand: Not Used; End Indicated: Back End	
F	Optional Operand; Block Location to which the Unlinker-Transaction moves next if no Transactions are unlinked	Unlinker-Transaction unconditionally moves to the sequential Block in the model

Figure 3 The UNLINK Block and Its Operands

[e] Familiarity with the concept of the Status Change Flag is assumed. For an explanation, see sections 7.2 and 7.3 in reference [1].



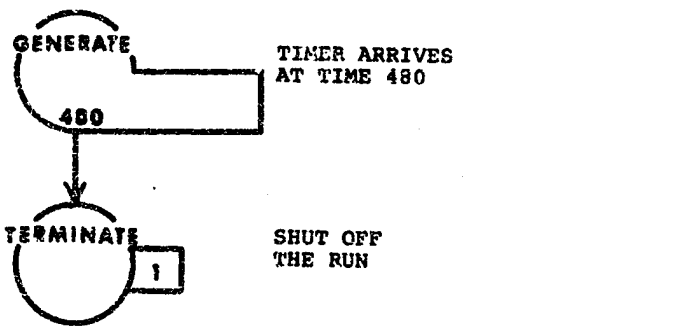


Figure 4 A First Example of User Chain Use

the just-unlinked Transaction has much in common with Transactions which are "on their way" into a model via a GENERATE Block into which they have not yet moved. Nevertheless, from the point of view of Current Counts, the Processor treats unlinked Transactions as though they are in the UNLINK Block whose execution caused them to be brought from the User Chain to the Current Events Chain. This fact is sometimes of importance when Current Block Counts are being interpreted. It also explains why the UNLINK Block's B Operand (i.e., the "Next Block Attempted" for unlinked Transactions) appears in Figure 3 on a path leading from the UNLINK Block. The idea here is to provide a graphic indication of the fact that unlinked Transactions do move from the User Chain via the UNLINK Block into their "Next Block Attempted". In contrast, the other two paths leading from the Figure 3 UNLINK Block apply to the Unlinker-Transaction. One path leads to the sequential Block; the other leads to the non-sequential Block which is implied if the optional F Operand is used.

4. Basic User Chain Use with Facilities and Storages

The basic use of User Chains with Facilities and Storages is illustrated through a series of three examples in this section. First, their use with single Facilities is shown. In this situation, the User Chain Link Indicator is adequate for the required look-ahead logic. Then, their use with Storages is illustrated. Such use requires analyst-supplied look-ahead logic, and this in turn requires caution in terms of a potential simultaneity-of-events problem which can arise. Later, in Sections 5 and 7, additional examples of User Chain use will also be given.

4.1 User Chain Use with a Facility. A Block Diagram for a one-line, one-server queuing system is presented in Figure 4. The particular model shown is for a "one-man barber shop". Interarrival time for customers at the shop is 18±6 minutes; service time is 16±4 minutes. The implicit time unit in the model can consequently be inferred from Figure 4 to be 1 minute. The two-Block timer segment indicates that, when the model is run, the simulation simply shuts off after 480 minutes (i.e., 8 hours) of simulated time. No special provisions are made, then, to provide a realistic closeup feature for the model. Any "customers" in the model at the end of the 8th simulated hour are simply left "as they are". The queue discipline to be practiced in the shop is first-come, first-served.

Notice that a LINK-UNLINK Block pair has been incorporated into the Block Diagram, implying that customer-Transactions who are waiting their turn to get a haircut are kept in this simple model on a User Chain. The LINK Block has been sandwiched

between the QUEUE and SEIZE Blocks; similarly, the UNLINK Block is sandwiched between the RELEASE and TERMINATE Blocks. The effect of the presence of these two Blocks will now be explored.

When a customer arrives at the shop, he first updates waiting line statistics by moving into the QUEUE Block. He then moves into the conditional-mode LINK Block. If the Link Indicator is "on" (traffic light red), the customer-Transaction is linked on the back (FIFO) of the User Chain HOLD, and the Link Indicator remains "on". If the Link Indicator is found to be "off" (traffic light green), however, it is switched "on" and the customer-Transaction proceeds to the Block in the location GETEM, i.e., moves into the SEIZE Block. The DEPART-ADVANCE-RELEASE sequence then follows. After the RELEASE, the customer-Transaction attempts to unlink 1 Transaction from the front of the User Chain HOLD (UNLINK Block D and E Operands both blank), sending it to the Block GETEM to capture the now-available Facility. If the attempted unlinking is unsuccessful because the User Chain is empty, the Link Indicator is switched from "on" to "off" (traffic light green) so that the next arrival, instead of linking, will move directly to SEIZE.

Note that, when the traffic light is red at the LINK Block, arriving Transactions are placed on the back of the User Chain (go to the back of the line). Later, via action initiated by an Unlinker Transaction at the UNLINK Block, they are removed from the front of the User Chain. The resulting queue discipline is first-come, first-served [f].

The pattern followed by the Link Indicator in the Figure 4 model reveals how it serves as a built-in look-ahead device in the context of Facility use. It is initially "off". The first customer-Transaction turns it "on", then captures the server. While the server is being used by this first customer of the day, the Link Indicator remains "on". Suppose the second customer-Transaction arrives while the server is still in use. Finding the Link Indicator "on", the second customer goes onto the User Chain. When the first customer finishes, he unlinks the second customer and sends him to capture the barber. Meantime, because the User Chain referenced from the UNLINK Block was not empty, the Link Indicator remains "on". In fact, it is "on" whenever any customer is using the barber, whether that customer (a) found the indicator "off", and moved directly to capture, or (b) found the indicator "on", and spent time in residence on the User Chain before eventually being sent to capture. The only way to turn the Link Indicator "off" is for a customer to finish with the barber when no other customers are waiting (User Chain empty). Turning the Link Indicator "off" in this circumstance guarantees that when the next customer does arrive, he will proceed to capture the barber immeidately.

The punchcards for the Figure 4 model were prepared, and the model was run for one simulated day. The D Operand on the START Card was used to force a chain printout at the end of the simulation. Figure 5 shows a portion of the output that was thereby produced. Parts (a), (b), and (c) in Figure 5 show the Current, Future, and User Chains, respectively. There is a single resident on the Current Events Chain, Transaction 3; this Transaction is poised to release the Facility. [The NBA ("Next Block Attempted") column in Figure 5(a) shows a value of 7. This is the Location occupied in the model by the RELEASE Block, as "counting it out" in Figure 4 will show.] The two residents on the Future Events Chain are the incipient Transaction arrivals at the two GENERATE Blocks in the Model. (Their NBA entries are 1 and 10, respectively, which are the Locations occupied by the GENERATE Blocks in the model.)

In Figure 5(c), the User Chain is described as "USER CHAIN 1". The symbolic name HOLD has been made equivalent to the number 1 by the Processor, and this numeric equivalent has been used to label the User Chain in the printout. There is one Transaction resident on the User Chain, Transaction 4. Note that the various column labels for the User Chain are identical to those for the Current and Future Events Chains.

The Transaction on the User Chain is the next customer, waiting for the barber. We know this because of the problem context, but the GPSS Processor does not know this. In fact, the "destination" of the Transaction on the User Chain will not be known to the Processor until it is unlinked. At that time, the UNLINK Block's B Ope-

[f] It is sometimes mistakenly concluded that if the B Operand at a LINK Block is FIFO (meaning that incoming Transactions are linked onto the back of the User Chain), it must be specified at the associated UNLINK Block that Transactions are to be removed from the front of the User Chain; or, that if the B Operand at a LINK Block is LIFO (meaning that incoming Transactions are linked onto the front of the User Chain), the associated UNLINK Block must specify that Transactions are to be removed from the back of the pertinent User Chain. This is not the case. The LINK and UNLINK Blocks are entirely independent of each other. It is the analyst's responsibility to see to it that the linking and unlinking criteria interact in such a way that the overall effect "makes sense" in context. For example, the B Operand at a LINK Block can be FIFO, and the associated UNLINK Block can specify that Transactions are to be unlinked from the back of the pertinent User Chain. The resulting queue discipline would be "last-come, first-served".

CURRENT EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4
3	480	6			7	3	453	0	0	0	0
								0	0	0	0
								0	0	0	0

(a) Current Events Chain*

FUTURE EVENTS CHAIN

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4
1	409				1	1	-27	0	0	0	0
								0	0	0	0
								0	0	0	0
5	960				10	5	-1	0	0	0	0
								0	0	0	0
								0	0	0	0

(b) Future Events Chain*

USER CHAIN 1

TRANS	BDT	BLOCK	PR	SF	NBA	SET	MARK-TIME	P1	P2	P3	P4
4	472					4	472	0	0	0	0
								0	0	0	0
								0	0	0	0

(c) User Chain*

USER CHAIN	TOTAL ENTRIES	AVERAGE TIME/TRANS	CURRENT CONTENTS	AVERAGE CONTENTS	MAXIMUM CONTENTS
HOLD	18	4.277	1	.160	1

(d) User Chain Statistics

QUEUE	MAXIMUM CONTENTS	AVERAGE CONTENTS	TOTAL ENTRIES	ZERO ENTRIES	PERCENT ZEROS	AVERAGE TIME/TRANS	$AVERAGE TIME/TRANS	TABLE NUMBER	CURRENT CONTENTS
JOEQ	1	.160	27	12	44.4	2.851	5.133		1

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

(e) Queue Statistics

Figure 5 Selected Output Produced by the Figure 4 Model at the End of the Simulation

*The 7 rightmost columns of information associated with these chains have been eliminated.

rand will be used by the Processor to determine the unlinked Transaction's "Next Block Attempted". Note, then, the entry in the BLOCK column in Figure 5(c) is "blank". The BLOCK column indicates which Block a Transaction is currently "in". But, as explained earlier, when a Transaction is on a User Chain, it is not "in" any Block in the model.

The BDT ("Block Departure Time") column in Figure 5(c) shows a value of 472. Block Departure Time is the time the Transaction is scheduled to try to move into its "Next Block Attempted." As far as its "future movement" is concerned, the BDT entry for User Chain Transactions is meaningless. The BDT value shown in User Chain printout can be interpreted as the time the Transaction was linked onto the User Chain.

The statistics for the User Chain HOLD which appear in the standard output are shown in Figure 5(d). Figure 5(e) shows the statistics for the Queue JOEQ. Comparison of the two sets of staticstics reveals that they are quite similar. The Queue statistics contain somewhat more information than those for the User Chain, indicating how many zero entries there were (ZERO ENTRIES), what percentage the zero entries were of the total (PERCENT ZEROS), and what the average Queue residence time was when zero entries were included (AVERAGE TIME/TRANS).

At first, it might be thought that there are no "zero entries" to User Chains because, "if blockage does not exist, Transactions bypass the chain and move directly forward in the model." User Chains can experience zero entries, however. That is, it is possible for some Transactions to have zero residence time on a User Chain. This will happen, for example, in the Figure 4 model when the following conditions are true.

(1) The Facility is in use.

(2) No Transaction is waiting to capture the Facility.

(3) There is a time-tie between the two events "completion of service", and "arrival of the next customer".

(4) The event-sequence is "arrival", followed by "service completion".

In the scan of the Current Events Chain at the simulated time in question, then, the arriving customer-Transaction is processed first, per (4) above. Finding the User Chain's Link Indicator "on", the Processor puts this Transaction on the User Chain. The releasing Transaction is then processed. After moving through the RELEASE Block, it unlinks the just-arrived Transaction from the User Chain and sends it to capture the Facility. Hence, although the just-arrived Transaction was made a User Chain resident, its residence time on the chain was zero. It contributes then, to the User Chain TOTAL ENTRIES statistic. And, from the Queue's point of view, it contributes to the ZERO ENTRIES statistic.

The phenomenon just described explains why there were 18 TOTAL ENTRIES to the User Chain in Figure 5(d), but only 15 non-zero entries to the Queue in Figure 5(e). Three of the User Chain entries were apparently of the "zero residence time" type. Note that this phenomenon also makes interpretation of the AVERAGE TIME/TRANS statistic for User Chains somewhat subtle. It would be easy to draw the false conclusion for the Figure 4 model that $AVERAGE TIME/TRANS in the Queue should equal the AVERAGE TIME/TRANS statistic for the User Chain. $AVERAGE TIME/TRANS measures the waiting time only of those who had to wait, however; in contrast, the AVERAGE TIME/TRANS value for User Chains can, in general, include Transactions which did not actually have to wait. The 3 "zero residence time" entries to the User Chain explains why AVERAGE TIME/TRANS is only 4.277 time units in Figure 5(d), whereas $AVERAGE TIME/TRANS is 5.1333 time units in Figure 5(e).

User Chain statistics and Queue statistics, although similar, differ from each other, then, in these three major ways.

(1) Zero-entry information is provided for Queues.

(2) The AVERAGE TIME/TRANS User Chain statistic requires careful interpretation.

(3) The distribution of Queue residence time is easily estimated with use of the QTABLE Card, whereas nothing analogous to the QTABLE Card is available for User Chains.

4.2 More About the Link Indicator. Consider use of the LINK-UNLINK Block pair in connection with any segment of a GPSS Block Diagram. Figure 6 illustrates this situation where, for generality, the particular Blocks occupying the Block Diagram segment in question are not shown, but for specificity the LINK-UNLINK Block Operands are shown. Assume that Transactions can gain entry to the segment only by moving through the conditional-mode LINK Block in the figure, and that they can exit the segment only by moving through the UNLINK Block. There can then never be more than one Transaction in the encircled Block Diagram segment at a time.

This last statement can be made as a direct consequence of the properties of the User Chain's Link Indicator. The reasoning goes like this. When the simulation starts, the encircled segment is empty. Furthermore, the Link Indicator is "off". When the first Transaction arrives at the LINK Block, it therefore moves immediately to the Block in the Location MOVIN, thereby entering the segment. (MOVIN is assumed to be the Location of the "first" Block in the encircled segment.) While the first Transaction is in the segment, then, any other arrivals at the LINK Block are put on the User Chain. When the first Transaction eventually leaves the segment via the UNLINK Block, it unhooks exactly 1 Transaction from the User Chain, routing it into the segment.

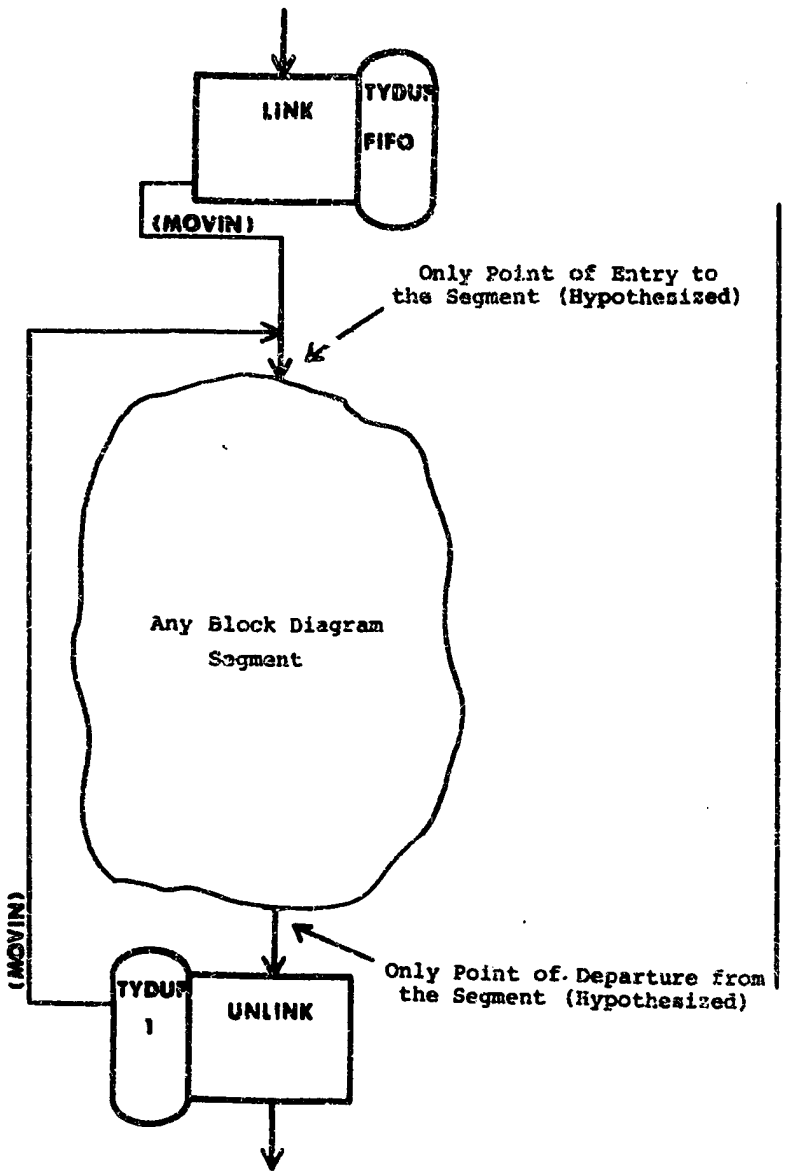


Figure 6 Use of a User Chain with an Arbitrary Block Diagram Segment

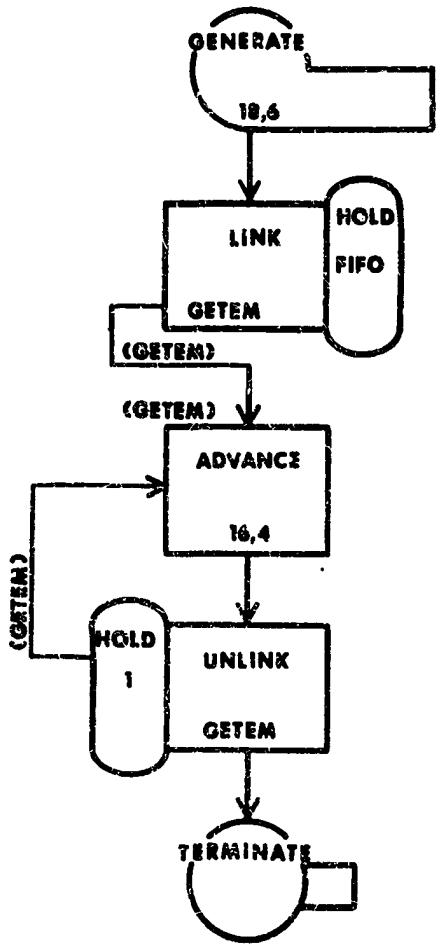


Figure 7 A Model Segment which Simulates a Single Server without Using a SEIZE-RELEASE Block Pair

Hence, the segment-exiting Transaction "replaces itself" in the segment with another Transaction. This "replacement pattern" is in effect as long as there is at least 1 Transaction on the User Chain when the UNLINK Block is executed. If the User Chain is empty when the UNLINK Block is executed, the result is that the Link Indicator gets turned "off". This means that when another Transaction eventually arrives at the LINK Block, it moves immediately into the segment, causing the Link Indicator to be turned back "on" in the process, etc.,etc.

The ideas just expressed really only repeat what was said about the Link Indicator when it was introduced in Section 3. Repeating the ideas in the context of Figure 6, however, leads directly to the two following conclusions.

Conclusion 1. When a User Chain is used in connection with a Facility, the SEIZE-RELEASE Block pair is not really needed, unless the analyst requires the statistics which the Facility entity provides. After all, use of a SEIZE-RELEASE Block pair has two effects.

(1) It guarantees that there will never be more than one Transaction at a time in transit between the pair of Blocks (assuming, of course, that alternative methods of "getting between" the pair of Blocks are not used in the model).

(2) It causes the GPSS Processor to maintain certain statistics about the "use" of the Facility.

But Effect (1) is precisely the effect that the Link Indicator has when a conditional-mode LINK Block is used. Consequently, if Effect (2) is not needed, the SEIZE-RELEASE Block pair can be eliminated. For example, Figure 7 repeats Figure 4, with the SEIZE-RELEASE Block pair eliminated. The QUEUE-DEPART Block pair has also been eliminated, on the hypothesis that the User Chain statistics are sufficient measures of waiting line behavior for the application at hand. The Block sequence "LINK-ADVANCE-UNLINK" in Figure 7 may seem a bit strange at first, but it nonetheless validly simulates a single server under the conditions stated here.

Conclusion 2. When the User Chain's Link Indicator is relied upon to supply "look-ahead logic", it is extremely inflexible. In fact, because a consequence of its use is to let only

"one Transaction at a time" in the model segment between the LINK and UNLINK Blocks, the Link Indicator is really only of value when the constrained resource being simulated between the LINK-UNLINK Blocks is a Facility (or a unit-capacity constraint). For example, suppose the constrained resource is being simulated with a Storage whose capacity is two. This means that up to 2 Transactions at a time should be permitted to be in transit between the LINK-UNLINK Block pair. Because this effect cannot be achieved with the Link Indicator, the analyst must supply his own look-ahead logic to determine whether an arriving Transaction can move into the model segment, or must be put onto the User Chain. The next section goes into further detail about use of a User Chain with the Storage entity.

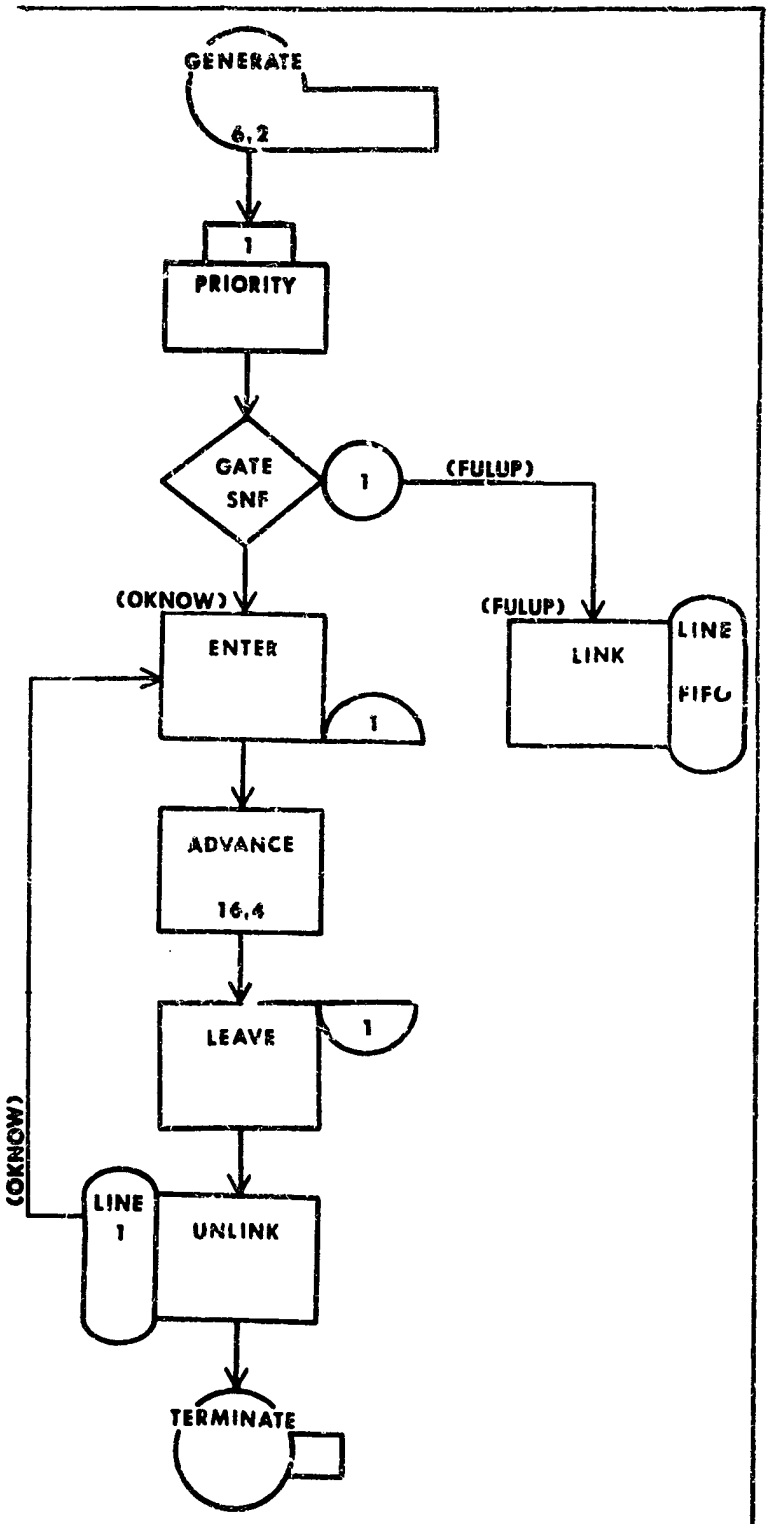


Figure 8 A Second Example of User Chain Use

4.3 User Chain User with a Storage. Suppose that in the Figure 4 barber shop, customer inter-arrival time decreases to 6±2 minutes and, to offset this heavier traffic pattern, two more barbers are hired. Figure 8 shows the Block Diagram for a model of the shop under these circumstances. Discussion of the model will be broken into two parts. First, the "GATE-ENTER-LINK" Block arrangement will be commented upon. Then the reason for placing the PRIORITY Block between the GENERATE and GATE Blocks will be explained.

As indicated under Conclusion 2 in the preceding sub-section, the analyst must supply his own look-ahead logic when a User Chain is used in conjunction with a Storage. The GATE Block in Figure 8 provides this required look-ahead logic. When a customer-Transaction moves into the "GATE SNF 1" Block, a test is conducted to determine whether at least one barber is currently available, i.e., to determine whether the Storage used to simulate the three barbers is not full. If the "Storage Not Full" condition is true, the customer-Transaction moves sequentially through the gate and captures a barber. If the "Storage Not Full" condition is false, the customer-Transaction exits the gate non-sequentially and moves into the LINK Block. No C Operand is provided with the LINK Block, with the result that Transactions entering it are unconditionally placed on the User Chain. Transactions enter the LINK Block, however, only on the condition that the Storage is full. Via use of the GATE Block, then, the "unconditional" linking of Transactions is forced to be conditional after all.

Now consider why the "PRIORITY 1" Block has been placed between the GENERATE and GATE Blocks in the Figure 8 model. The PRIORITY Block has been used to defense against invalid logic which could come about if a certain simultaneity-of-events situation were to arise. Suppose that the following conditions are true at a given point in simulated time.

(1) All 3 barbers are captured.
(2) At least 1 Transaction is waiting on the User Chain.
(3) One of the in-service customers is just leaving.
(4) The next customer is just arriving.
(5) The leaving Transaction is ahead of the arriving Transaction on the Current Events Chain.

When the leaving Transaction is processed, it first moves into the LEAVE Block, thereby changing the condition "Storage Not Full" from false to true. This leaving Transaction then moves into the UNLINK Block, causing the Transaction at the front of the User Chain to be moved to the Current Events Chain. Because of its earlier movement through the PRIORITY Block, the unlinked

Transaction has a Priority Level of 1. It is therefore placed on the Current Events Chain ahead of the arriving customer-Transaction, which has a Priority Level of 0. After the leaving Transaction has terminated, the Processor restarts the CEC scan. It first processes the just-unlinked customer-Transaction, moving it into the ENTER-ADVANCE sequence. Execution of the ENTER Block results in the condition "Storage Not Full" being made false again. When the arriving customer-Transaction is processed later in the scan, it therefore moves non-sequentially from the GATE Block to the LINK Block, and is put on the User Chain. The previously-waiting customer has captured the barber, which is as it should be.

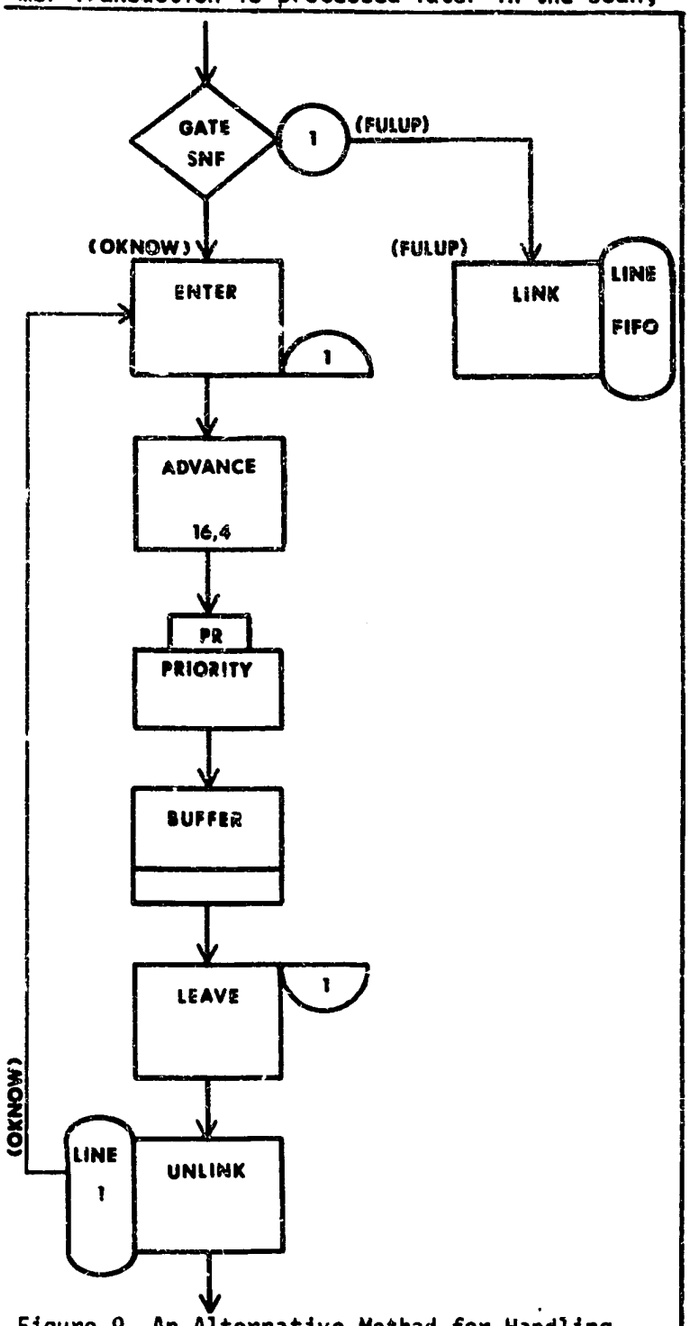


Figure 9 An Alternative Method for Handling the Potential Problem of Simultaneity in the Figure 8 Model

It is easy to see how the logic of the model would be subverted if the PRIORITY Block were removed from the model, and the conditions described above came about. The unlinked Transaction would be put on the current Events Chain behind the just-arriving customer-Transaction. When the arriving Transaction reached the GATE Block, the "Storage Not Full" condition would be true. The "newcomer" would therefore capture the barber. When the just-unlinked Transaction eventually tried to move into the ENTER Block, entry would be denied. This previously-waiting Transaction would have "missed its chance". Furthermore, its subsequent waiting would take place on the Current Events Chain, not on the User Chain.

Whether using the PRIORITY Block in the Figure 8 model is "worth it" can be debated. The Block does guarantee that the logic of the model will always be valid. On the other hand, to include this "additional" Block increases the number of Blocks in the model from 8 to 9. Speaking very roughly, this means that execution time for the model is increased by about 12.5% relative to the no-PRIORITY-Block version of the model [g]. Depending on the size of the implicit time unit and the intensity of the traffic pattern, the conditions leading to the "problem of simultaneity" may come about very infrequently. To save execution time in modeling situations such as this, the analyst might prefer to deliberately exclude the PRIORITY Block, and simply "accept" occasional invalidity in his models. (Note that the potential "problem of simultaneity" did not have to be defensed against when the Link Indicator provided the look-ahead logic in Figure 4. To check your understanding of the ideas presented so far in this paper, you should now try to explain why this is true).

4.4 An Alternative for Handling the Simultaneity Problem. In the Figure 8 model, it was "convenient" to place the PRIORITY Block between the GENERATE and GATE Blocks. But this was largely because the modeling context used as an example was completely self-contained. It is not normally true that a Transaction "approaches" a Storage from a GENERATE Block. More often than not, the "approach" is made from some preceding non-trivial model segment. The question then arises, how is one to handle the problem of simultaneity in this circumstance? The purpose of this section is to suggest an alternative which can be used under more complicated circumstances.

Figure 9 shows the suggested alternative in the

[g] When the PRIORITY Block's A Operand is a constant, a minimum of 114 assembler instruction-executions are performed when a Transaction moves into the Block. (This count applies to GPSS/360, Version 1, Modification Level 3.)

form of a Block Diagram segment corresponding to use of the Figure 8 Storage. It is assumed that all Transactions move into the segment with a common Priority Level, whatever that may be. The "defense" against the simultaneity problem takes the form of the PRIORITY-BUFFER sequence placed between the ADVANCE and LEAVE Blocks.

Consider how the PRIORITY-BUFFER combination works to eliminate the simultaneity problem. Assume that the conditions required for the simultaneity problem are in effect, as follows.

(1) All 3 barbers are captured.
(2) At least 1 Transaction is waiting on the User Chain.
(3) One of the in-service customers is just leaving.
(4) The next customer is just arriving.
(5) The leaving Transaction is ahead of the arriving Transaction on the Current Events Chain.

Now, when the leaving Transaction is processed, it immediately moves into the PRIORITY Block, where its "old" Priority Level is reassigned as its "new" Priority Level. This produces no change in Priority Level, but it does cause the Processor to re-position the Transaction on the Current Events Chain as the last member in its "new" Priority Class. This means that the leaving Transaction is now behind the arriving Transaction (which reverses condition (5) stated above). The leaving Transaction then moves into the BUFFER Block, forcing the Processor to re-start its CEC scan. As the re-initiated scan proceeds, the arriving Transaction is (eventually) encountered, finds the condition "Storage Not Full" is false (the leaving Transaction has not yet executed the LEAVE Block), and therefore transfers non-sequentially to the User Chain. Later in the scan, the Processor resumes the forward movement of the leaving Transaction, moving it through the LEAVE Block to the UNLINK Block. The Transaction at the front of the User Chain is then transferred to the Current Events Chain, and enters the Storage when the scan is re-started (execution of the LEAVE Block caused the Status Change Flag to be turned "on"; execution of the UNLINK Block then caused it to be turned on "again", redundantly).

It should be clear what would happen under the stated conditions if the PRIORITY-BUFFER Blocks were not in the model. The leaving Transaction would be processed first, making the condition "Storage Not Full" true. The unlinked Transaction would be put on the Current Events Chain behind the arriving Transaction, since they are postulated to have the same Priority Level. The arriving Transaction would then enter the Storage, thereby capturing the server who had been intended for the unlinked Transaction. By the time the unlinked Transaction was processed, there would be "no room left" for it in the Storage. In short, the logic of the model would be invalid.

In Figure 9, the PRIORITY and BUFFER Blocks were shown separately, albeit in sequence, to make the preceding explanation a bit more straightforward. Active GPSS users will recall that when the "buffer option" is used with the PRIORITY Block, the effect achieved is precisely identical to that of the two-Block PRIORITY-BUFFER sequence shown in Figure 9. That is, the single Block "PRIORITY PR,BUFFER" could be used to replace the PRIORITY-BUFFER Block-pair. In the remaining examples in this paper, this "buffer option" will be used with the PRIORITY Block when the occasion arises, for the sake of "Block economy".

5. More Examples of User Chain Use

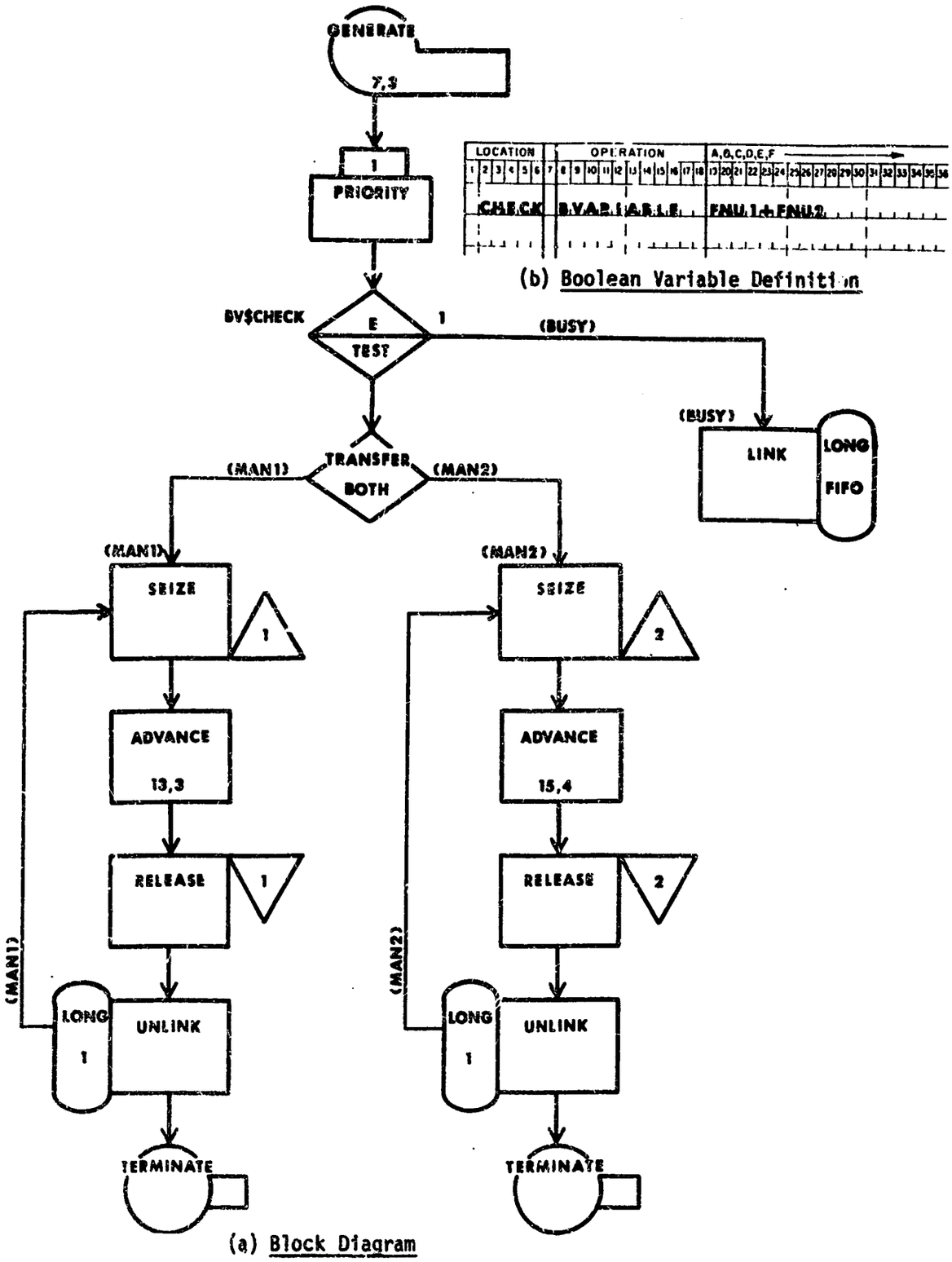
Two more examples of User Chain use with Facilities are given in this section. The first example illustrates application of User Chains with two Facilities operating in parallel. The second example shows how the "shortest imminent operation" queue discipline is simulated with use of Parameter-mode linking at the LINK Block.

5.1 User Chain Use with Two Parallel Facilities.

Suppose that two barbers work in a barber shop. Service time for the first and second of these barbers is 13±3 and 15±4 minutes, respectively. Customers arrive at the shop every 7±3 minutes. Figure 10 shows how a User Chain can be incorporated into a model of the barber shop. When a customer-Transaction arrives at the shop, it moves (through the PRIORITY Block) into the TEST Block shown in Figure 10(a). There, it evaluates the Boolean Variable CHECK, as defined in Figure 10(b), to determine whether either one or both of the barbers are available. If a barber is free, the customer-Transaction proceeds to the TRANSFER Block in BOTH mode, from whence it is routed to a SEIZE Block not currently denying entry. If both barbers are busy, it is routed from the transfer-mode TEST Block to the unconditional-mode LINK Block. There, it is linked onto the back of the User Chain LONG.

Whenever a customer-Transaction releases a barber, it causes another waiting customer to be unlinked from the User Chain and routed directly to the pertinent SEIZE Block. Because of the "PRIORITY 1" Block following the GENERATE Block, the unlinked customer-Transaction is assured of being the one to capture the just-released barber under all circumstances.

This example is one in which the execution time savings resulting from User Chain use can be substantial. In contrast with Figures 4 and 8, where the potential blocking conditions are both unique, the TRANSFER Block in Figure 10 offers non-unique blocking. If the User Chain were not used, each delayed Transaction on the Current Events Chain would attempt to move through the TRANSFER Block at each CEC scan, thereby consuming a telling amount of time. For the model shown, the TRANSFER Block is executed successfully one time by each arriving customer who finds a barber immediately available. No attempt is otherwise made to execute the Block.



(b) Boolean Variable Definition

(a) Block Diagram

Figure 10 A Third Example of User Chain Use

5.2 User Chain Use for "Shortest Imminent Operation" Queue Discipline. Consider this problem. At the Facility MAC, the queue discipline practiced is "shortest imminent operation". This means that the Transaction expected to hold the Facility the shortest length of time is the one permitted to capture it next. In case of ties for shortest imminent operation, the ties are to be resolved on a first-come, first-served basis. When a Transaction does capture the Facility, its actual holding time follows the exponential distribution.

Assuming that a Transaction's expected holding time at the Facility is stored in its second Parameter, Figure 11 shows a Block Diagram segment which implements this queue discipline. (It is assumed that the Function symbolically named XPDIS is the usual 24-point Function used to sample from an exponential distribution with a mean of 1.) Transactions waiting for the Facility are put onto the User Chain QUE, ordered according to their P2 value. This means they are put onto the chain in order of "shortest imminent operation", with expected operation times increasing from the front of the User Chain toward the back. Furthermore, in event of ties, each most-recent arrival is placed behind earlier arrivals which have the same P2 value. In short, linking "in Parameter mode" results in direct implementation of the shortest imminent operation queue discipline (assuming, of course, that Transactions are later unhooked from the front of the User Chain).

In the Figure 11 model segment, just before a Transaction releases the Facility, it moves into a "PRIORITY PR,BUFFER" Block. The Processor therefore re-positions the Transaction on the Current Events Chain as the last member in its Priority Class, and re-starts the scan. This guarantees that, in case of a time-tie between the events "next arrival of a job", and "release of the Facility", the arriving job-Transaction is hooked onto the User Chain before the next Transaction to capture the Facility is captured. (It is assumed that all Transactions which use the Facility have the same Priority Level.) The result is to insure that the arriving job-Transaction is included in the "competition" that takes place to see which waiting job has the shortest imminent operation. If this simultaneity-of-events situation arose and the PRIORITY Block were not included in the Figure 11 model segment, the shortest imminent operation queue discipline could be violated.

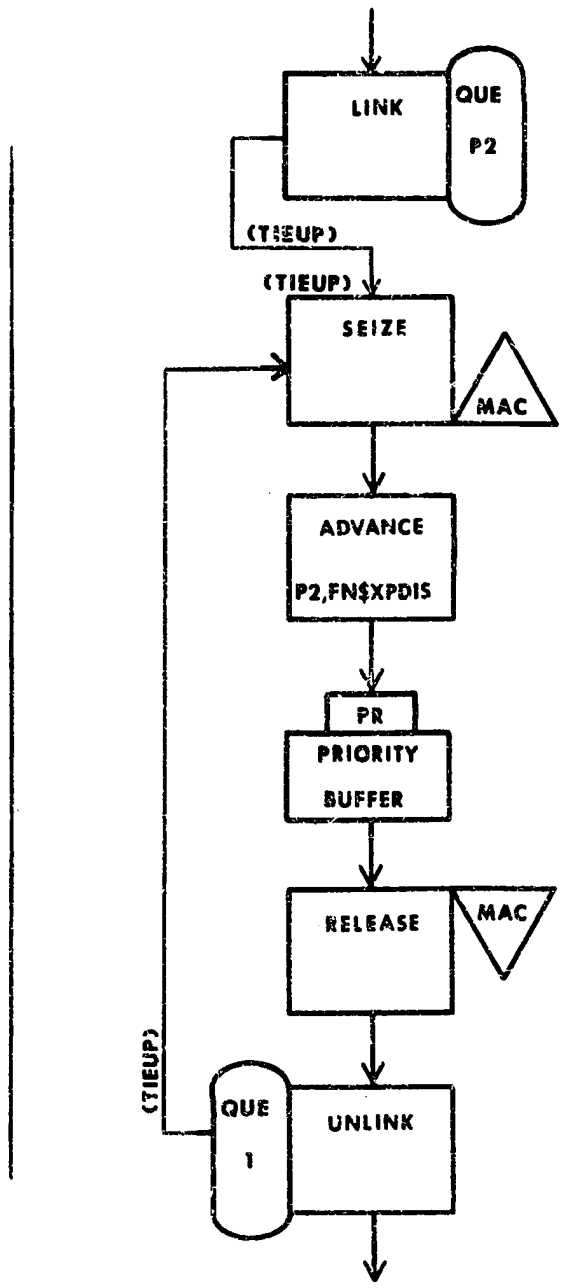


Figure 11 A Fourth Example of User Chain Use

6. User Chain Standard Numerical Attributes

Each User Chain has five Standard Numerical Attributes associated with it. The pre-defined names of these attributes, and the significance of the corresponding values, are shown in Table 1.

When the Processor encounters a CLEAR Card, the values of these Standard Numerical Attributes are set to zero, and any User Chain residents are removed from the model. The effect of the RESET Card is to set the values of CAj, CCj, and CTj to zero. The value of CHj remains the same, and the value of CMj is set to the current value of CHj. Of course, any User Chain residents are left undisturbed during the resetting operation.

7. Conditional Unlinking of Transactions from User Chains

In Section 3, use of the UNLINK Block D and E Operands to remove Transactions from (a) the front, or (b) the back of User Chains was introduced. In neither of these cases does a Transaction have to meet a particular condition, other than "relative position on the chain", to qualify for unlinking. There are three other D and E Operand combinations which can be used to impose on potential Unlinkee Transactions the require-

Pre-defined Name [h]	Value
CAj, or CA$sn	Integer portion of the average number of Transactions on the chain
CCj, or CC$sn	The total number of Transactions hooked onto the chain during the course of the simulation
CHj, or CH$sn	The number of Transactions currently on the chain
CMj, or CM$sn	Maximum number of Transactions simultaneously resident on the chain during the simulation; the maximum value CHj (or CH$sn) has attained
CTj, or CT$sn	Integer portion of the average Transaction residence time on the chain

Table 1 User Chain Standard Numerical Attributes

Combination Number	D Operand	E Operand	Condition Required for Unlinking
1	Any Standard Numerical Attribute	Not used	Let "j" represent the value of the D Operand; the User Chain Transaction qualifies for unlinking if its j-th Parameter value equals the value of the Unlinker's j-th Parameter
2	Any Standard Numerical Attribute	Any Standard Numerical Attribute	Let "j" represent the value of the D Operand; the potential Unlinkee qualifies if its j-th Parameter value equals the value of the E Operand
3	BVj, or BV$sn	Not used	The potential Unlinkee qualifies if the Boolean Variable numbered j (or symbolically named sn) is true when it is evaluated with that Transaction's Priority Level and Parameter values

Table 2 Additional D and E Combinations Possible for the UNLINK Block

ment that they satisfy a specified condition. These other three combinations are shown in Table 2.

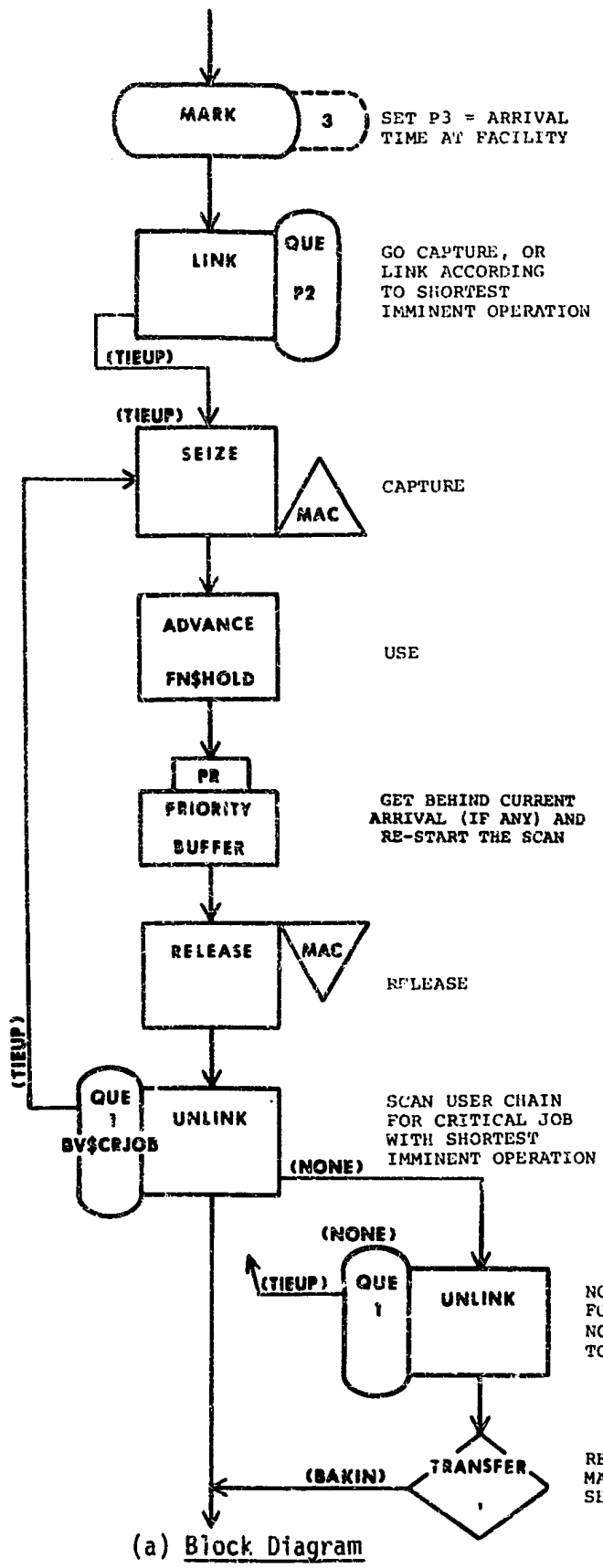
For all three of the Table 2 combinations, the User Chain is scanned from front to back by the Processor until the Unlink Count has been satisfied, or the back of the chain has been reached, whichever occurs first. In Combination 1, the value of a specified Parameter of the potential Unlinkee must equal the value of the same Parameter of the Unlinker. The UNLINK Block's D Operand indicates the number of the applicable Parameter; the E Operand is not used. In Combination 2, the value of a specified Parameter of the potential Unlinkee must equal some other arbitrarily-specified value. The UNLINK Block's D Operand again provides the number of the potential Unlinkee's applicable Parameter; the E Operand is the "Match Argument", i.e., provides the value which the Unlinkee's Parameter value must equal.

In Combination 3, the D Operand references a Boolean Variable, and the E Operand is not used. For each Transaction on the User Chain, the Processor evaluates the Boolean Variable. Only if its value is true does the User Chain Transaction qualify for unlinking. The question naturally arises, "how can the value of a Boolean Variable be made to depend on properties of a Transaction on a User Chain?" The answer is that if numeric data references in the Boolean Variable include Priority Level and/or Parameter values, the User Chain Transaction currently being examined supplies these values, not the Transaction at the UNLINK Block.

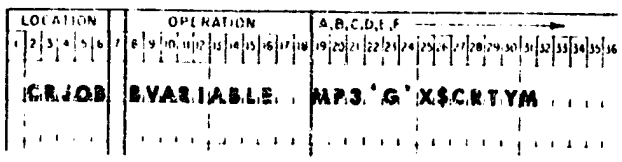
An example will now be given to show use of a Boolean Variable with the UNLINK Block. Consider the "shortest imminent operation" queue discipline, as illustrated in Figure 11. A disadvantage of this queue discipline is that jobs with a large imminent operation time can be delayed for very long times waiting for the Facility. This happens if jobs with shorter operation times keep arriving at the Facility before the bigger jobs can capture it. The problem can be avoided by dividing all waiting jobs into two groups, as determined by how long they have been waiting. Highest priority is given to those jobs that have been waiting longer than some pre-determined time, called the critical threshhold. Jobs in this group are termed "critical". Within the set of critical jobs, queue discipline is "shortest imminent operation". Queue discipline for the non-critical jobs is also "shortest imminent operation". The overall queue discipline, then, is "serve critical jobs first, then serve non-critical jobs; in each of these two categories, select jobs according to shortest imminent operation."

[h] "j" is understood to be the number of the User Chain, if it has been named numerically; "sn" is understood to be its symbolic name, if it has been named symbolically.

A Block Diagram for this overall queue discipline is shown in Figure 12(a). When a job-Transaction enters the segment, its time-of-arrival is first marked in Parameter 3. The Transaction then captures the Facility MAC immediately if possible, and otherwise goes onto the User Chain, ordered according to its imminent operation time as carried in Parameter 2. When a job-Transaction is finished using the Facility, it enters an UNLINK Block to request a Boolean-mode scan of the User Chain. The User Chain is scanned from front-to-back in a search for the first job-Transaction, if any, for which the Boolean Variable CRJOB [defined in Figure 12(b)] is true, i.e., for which residence time on the User Chain exceeds the critical threshhold, as held in the Savevalue CRTYM. If such a Transaction is found, it is unhooked and sent to capture; meantime, the Unlinker continues to the sequential Block. If there are no critical jobs, however, the Unlinker takes the non-sequential exit from the first UNLINK Block to a second UNLINK Block, where it unhooks the front-end Transaction from the User Chain and sends it to capture.



(a) Block Diagram



(b) Boolean Variable Definition

Figure 12 A Fifth Example of User Chain Use

8. Other Sources of Examples

The various examples presented in this paper should alert the GPSS analyst to the fact that care must be exercised, especially with respect to the "simultaneity of events" problem, when User Chains are applied in the language. Space does not permit presentation of additional, larger-scale examples here. Those interested should refer to case studies 7A and 7C in reference [1]. Case study 7A employs User Chains in comparing a one-line, multiple-server queuing system to a multiple-line, multiple-server queuing system in a banking context. For the latter system, Facilities are used in parallel to simulate the parallel servers, and a separate User Chain is associated with each Facility. In case study 7C, in the context of a "city's vehicle-maintenance garage", parallel servers are also simulated with Facilities in parallel. In this case, pre-emptive use of Facilities is allowed. Due emphasis is given to a simultaneity-of-events problem which can arise when pre-emption and a normal "release" of a Facility occur at the same time.

There are examples in other "pedagogical" sources, but they do not abound. In [2], examples of User Chain use are given (a) for first-come, first-served queue discipline with a single Facility, (b) for random queue discipline with a single Facility, (c) when unlinking is based on a "Parameter-match" between the unlinker, and the potential unlinkee, and (d) when a Boolean Variable

must be true before unlinking can occur. The same examples are repeated in [3] and [4]. There are no examples in these sources that consider User Chain use for constrained resources simulated with more than a single Facility, and the simultaneity-of-events problem is not mentioned. In [5], a "random queue discipline" model is shown which consumes less CPU time than the one given in the above sources. Use of User Chains to implement "least job slack per remaining operation" queue discipline in a job shop problem is also shown in [5]. In [6], Greenberg gives 6 examples of User Chain use, restricting the constrained resource to a single Facility. The purpose of having 6 examples is to explain various LINK-UNLINK Block Operand combinations. No additional applications are illustrated, and no mention is made of the simultaneity-of-events problem.

9. Summary

This paper, presented as a tutorial, elaborates on the User Chain entity in GPSS. The concept of User Chains is presented, the potential benefits to be gained from their use are described, and a detailed description of the two GPSS Blocks supporting User Chain use is provided. Examples illustrating a range of User Chain applications are introduced and discussed. The potential "simultaneity-of-events" problem that can occur even with only modestly imaginative User Chain use is brought to light in several of these examples. Reference is made to additional sources of examples in the literature.

10. Biography

Thomas J. Schriber is a Professor of Management Science at the University of Michigan. He regularly teaches a 5-day "introductory" course and a 3-day "advanced" course on GPSS in the University of Michigan's Engineering Summer Conference Series. He gives GPSS courses in industry, and has written two introductory books on the subject.

11. References

[1] Schriber, Thomas J., A GPSS Primer (currently available in softbound form from Ulrich's Books, Inc., Ann Arbor, MI; being published in hardbound by John Wiley & Sons, Inc., with publication date not yet set; title only tentative for hardbound form)

[2] GPSS/360 User's Manual (IBM; Form Number GH20-0326)

[3] GPSS/360 Version 2 User's Manual (IBM; Form Number SH20-0694)

[4] GPSS V User's Manual (IBM; Form Number SH20-0851)

[5] Schriber, Thomas J., GPSS/360: Introductory Concepts and Case Studies (Ulrich's Books, Inc., 1968; 1969; 1971)

[6] Greenberg, Stanley, GPSS Primer (Wiley-Interscience, 1972)