Homogeneous event indexes

By F. FEIND, E. KNUTH, P. RADÓ, J. VARSÁNYI

1. Discrete event simulation

General purpose discrete simulation languages are based on the so-called "event notice" concept. It means special data patterns assigned to each simulation event and handled by the run-time timing routines of the systems.

These routines have the following main functions:

1) scheduling future events (generated in the course of program execution);
2) registering the events in a properly linked order so as to be able to produce the "next event" in any case.

We assume in this paper that whenever an event is scheduled its event time is always known. The examination of more general, e.g. conditional scheduling possibilities would lead to much more complicated structures. Therefore we can assume that definite time values are assigned to each event notice and they are necessarily ordered according to their time values.

We are not dealing with the problems of multiple schedulations into the same time point, which may be a question of disciplines or priorites but has no importance as to the performance of the algorithms we do.

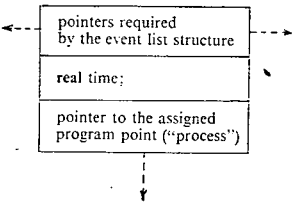

Fig. 1. General structure of an event notice

Now discrete event simulation works as follows: at the initiation and during the whole execution event notices are generated and inserted into the event list for all arisen simulation activity demanding a definit timing. The program execution is controlled by the event list i.e. having finished an activity assigned to an event notice the activity corresponding to the next one is going to be carried out according to the instantaneous state of the list structure. (More detailed descriptions can be found in references [1], [2] and [3].)

2. Event list algorithms

The functional activities of event list algorithms may be comprised by the following four procedures:

1. scan (t)

The procedure finds the event after which a new event will have to be inserted if its event time value is t. Thus using the SIMULA formalism [4] the procedure specification is

ref (event) **procedure** scan (t); **real** t;

or according to the PASCAL formalism [7]

function scan (t:real): event;

where we denoted the data structure "event notice" simply by "event".

2. insert (E, P, t)

E is the event after which the insertion must be done. P is the simulation activity having to be timed. The procedure must generate a new event notice of time value t and insert it after E referring to the simulation activity P.

The formal spedification is

procedure insert (E, P, t); **ref** (event) E; **ref** (process) P; **real** t;

or

procedure insert (E:event, P:process, t:real);

and the most typical call of the procedure is insert (scan (t), P, t);.

3. delete (E)

This procedure must delete the event E from the event list preserving the correctness of the remaining list. Formally:

procedure delete (E); **ref** (event) E;

or

procedure delete (E: event);

4. delete current

This procedure is equivalent to the call *delete* (current); where "current" is always the first event of the list. For a deletion implied by the termination of any simultation activity is always related to the current event, it is worth doing to develop this procedure in a more special way than the previous, general one.

3. Linear list structure

The simplest structure, the linear list is widely used in simulation languages including SIMULA [4], SIMSCRIPT [5], GPSS [6].

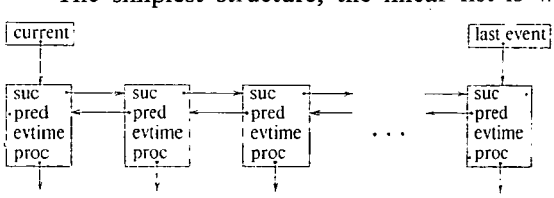

Fig. 2. Structure of the linear list

Suc and *pred* are the linking pointers. *Evtime* is the time value assigned to the notices. If E is an event notice then the relation

$$E.suc.evtime \geqq E.evtime^*$$

must always be satisfied. The pointers *current* and *last event* always point to the first and last events respectively.

The event list algorithms for linear lists are very simple:

1. scan (t)

Begin to compare t with the values *evtime* from the *last event* and follow it sequentially while *evtime*$>t$ holds. (This simple algorithm is described in the Appendix in detail.) The reason of scanning from the end of the list is its better performance.

2. insert (E, P, t)

The procedure is to perform a usual insertion into a two-way list as it is shown in Fig. 3 and in formal way in the Appendix.

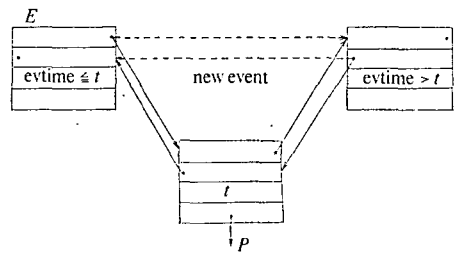

Fig. 3. Insertion into a linear list

3—4. delete (E)

Now a deletion simply means resetting the pointers having been set by the insertion Fig. 3 (see Appendix). The special procedure "delete current" need not be done in different way.

4. Further structures proposed

With the linear list structure, the overhead time taken by a call of the procedure scan is proportianal to N, the number of events in the list. (It leads to the amount of scheduling overhead proportional to N^2.)

Myhrhaug [8] gave the first results to improve it replacing the linear list by binary tree structures. Knuth [9,p. 150] also gave a brief account under the title of "priority queues" and suggested the use of the so-called "post-order trees".

A complete investigation on this topic can be found in the work [10] with tests using a set of typical stochastic scheduling distributions. The paper explicitely produces the algorithms of three different structures and compares them with the linear one.

These structures are the following:

- post — order tree,
- end — order tree,
- indexed list.

The behaviour of all the structures highly depends on the probabilistic nature of the event stream, but the indexed list structure provides the best overall performance. This structure, however, needs an adaptive mechanism to set an interval parameter according to the operating conditions.

---

* In connection with dot notation we refer to [4] and [6].

In this paper we introduce a dynamic version of the indexed list structure which has only a bit worse performance than a well chosen static one, but it needs no adaptive mechanism and it is completely independent of the probabilistic properties of the arrival stream because of its homogeneous nature.

5. General characteristic of homogeneous structures

Let $k>1$ be an integer. Suppose that we have a linear list and every k-th of the elements is pointed to by indexes constituting a new linear list. All the k-th elements of this list are also pointed to by second level indexes and the structure of levels is continued terminating at a highest level consisting of only one element.

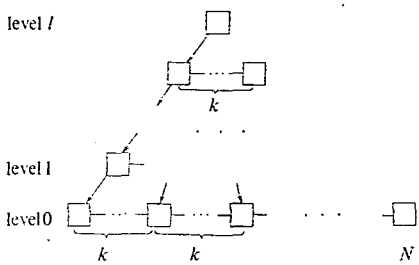

Fig. 4. Fixed homogeneous indexes

The first task is to find the optimal value of the indexing step k:

Theorem 1. The average number of comparisons needed by a call of the procedure scan is minimal if $k=3$.

Proof. The procedure scan works in a natural way: beginning at the highest level the procedure executes a linear list scan in each level from the entry point designated by the pointer having been found at the previous level. Thus, supposing that the probabilities of terminating the scan in a given level are all the same for any of the elements, the average number of conparisons in any level is equal to $k/2$. Hence the average number $m(k)$ of the comparisons in all levels will be

$$m(k) = \frac{k \log N}{2 \log k}.$$

Considering $m(x)$ as a continuous function of $x>1$ simple derivation shows that its only local minimum is achieved at the point $x=e$, and referring to the monotonity when $x>e$ the simple comparison $m(2)>m(3)$ proves the statement.

6. The „2/3-structure"

The following structure (proposed by one of the authors of this paper E. Knuth) is theoretically based on the result of Theorem 1.

Let us allow to use steps of size both $k=2$ and $k=3$ at random in the following way:

- If an arrival occurs into a "molecule" of 2 elements it will simply become a "molecule" of 3 elements.
- If an arrival occurs into a "molecule" of 3 elements it will form two "molecules" of 2 elements and a new index will have to be inserted into the next level in the very same way just described. This process continues

up to a level in which the insertion can be done into a "molecule" of only 2 elements.

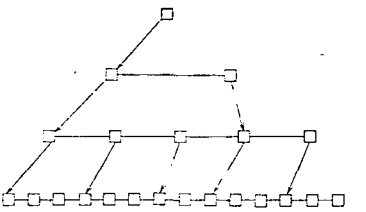

Fig. 5. Example of a "2/3-structure"

The main properties of the structure we defined are the following:

1. The average number of new elements having to be inserted when calling the procedure insert is less than two. This follows from the simple fact that in the worst case, when all the molecules have two elements the whole structure has $2N-1$ elements. (The exact value of the mean number will be determined in paragraph 7.)

2. The average number of comparisons needed by a call of the procedure scan is less than $m(3)$ i.e. it is nearer to the ideal value $m(e)$. (Also proved in paragraph 7.)

3. The "2/3-structure" has a homogeneous nature i.e. it is independent of the distribution of the arrival stream. (This follows from its logical symmetry and has been empirically tested too.)

4. The static structure described in paragraph 5. is naturally unsuitable to practical use for preserving the fixed structure would need complicated insertion procedures. This is also solved by allowing variable size molecules.

7. Stochastic behaviour

Let α_i and β_i be the numbers of the molecules of 2 elements and 3 elements respectively after the i-th insertion.

Theorem 2. $\frac{\alpha_i}{\beta_i}$ stochastically tends to 2.

Proof. Let ξ_k be the number of elements contained in all the molecules of 2 elements after the insertion of the $k+7$-th element. Then ξ_k is an inhomogeneous Markov process with $\xi_0=4$ (seven elements can be arranged in a unique way).

Let

$$\varphi_k = \begin{cases} 0 \text{ if the } k+7\text{-th arrival changes a molecule of} \\ 2 \text{ elements to a molecule of 3 elements;} \\ 1 \text{ if the } k+7\text{-th arrival cuts a molecule of} \\ 3 \text{ elements to two molecules of 2 elements.} \end{cases}$$

Considering the effect of an arrival we have

$$\xi_k = \xi_{k-1}+4\varphi_k+2(\varphi_k-1) = \xi_{k-1}+6\varphi_k-2$$

hence

$$\xi_k = \xi_0+6\sum_{i=1}^{k}\varphi_i-2k.$$

From the construction

$$P\{\varphi_k = 1|\varphi_{k-1}, \ldots, \varphi_1\} = 1-\frac{\xi_{k-1}}{k+6} = 1-\frac{4+6\sum_{i=1}^{k-1}\varphi_i-2(k-1)}{k+6}$$

holds with probability 1. For

$$P\{\varphi_k = 1\} = E\varphi_k$$

it follows

$$E\{\varphi_k|\varphi_{k-1}, \ldots, \varphi_1\} = \frac{3k-6\sum_{i=1}^{k}\varphi_i}{k+6},$$

therefore

$$E\varphi_k = \frac{3k-6\sum_{i=1}^{k-1}E\varphi_i}{k+6}.$$

Since $\xi_0=4$ we get $E\varphi_1=\frac{3}{7}$ and by induction $E\varphi_k=\frac{3}{7}$ for $k\geqq 1$.

Now we have to find the variance

$$D^2(E(\varphi_k|\varphi_{k-1}, \ldots, \varphi_1)).$$

From the relation

$$P\{\varphi_i = 1|\varphi_j = 0, \varphi_{j+1} = y_{j+1}, \ldots, \varphi_{i-1} = y_{i-1}, \xi_{j-1} = x) -$$

$$-P\{\varphi_i = 1|\varphi_j = 0, \varphi_{j+1} = y_{j+1}, \ldots, \varphi_{i-1} = y_{i-1}, \xi_{i-1} = x) = \frac{6}{i}$$

for any $y_{j+1}, \ldots, y_{i-1}=0, 1$ and $0\leqq x\leqq j-1$; we get

$$P\{\varphi_i = 1|\varphi_j = 0\} - P\{\varphi_i = 1|\varphi_j = 1\} = \frac{6}{i} \quad (j < i).$$

And from

$$\frac{4}{7}P\{\varphi_i = 1|\varphi_j = 0\} + \frac{3}{4}P\{\varphi_i = 1|\varphi_j = 1\} = \frac{3}{7}$$

it follows

$$P\{\varphi_i = 1|\varphi_j = 1\} = \frac{3}{7} - \frac{24}{7i}.$$

Hence

$$E(\varphi_i\varphi_j) - E(\varphi_i)E(\varphi_j) = P\{\varphi_j = 1\}(P\{\varphi_i = 1|\varphi_j = 1\} - P\{\varphi_i = 1\}) = \frac{3}{7}\cdot\frac{24}{7i}.$$

On this basis

$$\lim_{k\to\infty} D^2(E(\varphi_k|\varphi_{k-1}, \ldots, \varphi_1)) = \lim_{k\to\infty}\left[\frac{\sum_{i=1}^{k} D^2(\varphi_i)}{(k+6)^2} + \frac{2\sum_{1\leqq i<j\leqq k}\frac{72}{49i}}{(k+6)^2}\right] = 0$$

and referring to the Tchebycheff inequality, it implies the stochastic convergence

$$\frac{\xi_k}{k+6} \to \frac{4}{7}$$

which is equivalent to the statement of the theorem.

Corollary 1. From theorem 2. we get by simple conputation that the average number of events having to be inserted when calling the procedure insert is equal to 1.75.

Corollary 2. The average number of comparisons needed by a call of the procedure scan is $m\left(\frac{7}{3}\right)$. Comparing it to the best fixed structure $m(3)$ we find $m\left(\frac{7}{3}\right) < m(3)$.

8. Algorithms for the 2/3-structure

To build the algorithms in detail first we have to define the following pointers:

- *suc* (successor) and *pred* (predecessor) are the usual linkage pointers for linear lists.
- *for* (forward link) is the indexing pointer between the levels, see Fig. 4. (When being in the lowest level we use *for* to point at the process assigned.)
- *back* (backward link) is a redundant pointer not shown in Figures 4 and 5. The reason of introducing it is to make the building of procedure delete easier. The pointer value is defined by

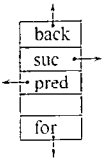

Fig. 6. Event notice pointers at homogeneous structures

for $F:-E.for$, $F.suc$ **while** $F=/=E.suc.for$ **do** $F.back:-E$;

where E is any event not at the lowest level.

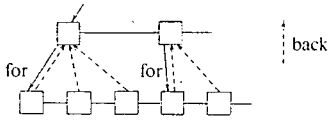

Fig. 7. Definition of the pointer *back*

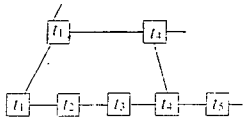

Fig 8. evtime values

Now let us define the *evtime* values at all the levels in the natural way:

$$E.evtime = E.for.evtime.$$

On these bases the logical structures of the procedures we tested are the following. (The exact versions are contained in the Appendix.)

The general procedure delete (E) is contained only by the Appendix for its structure is quite similar one barring that technically more complicated.

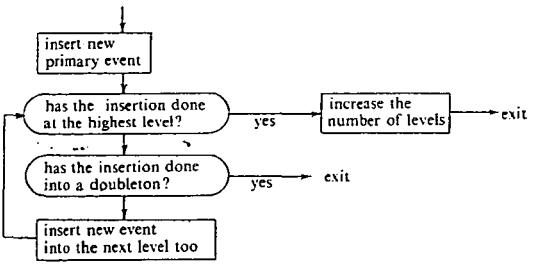

Fig. 10. Logical structure of procedure insert (E, P, t)

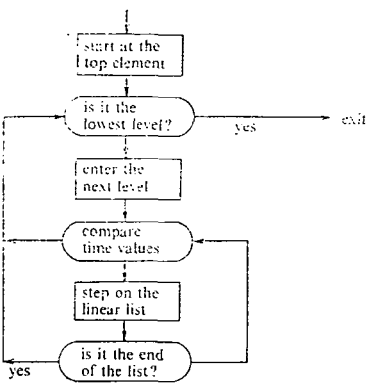

Fig. 9. Logical structure of procedure scan (t)

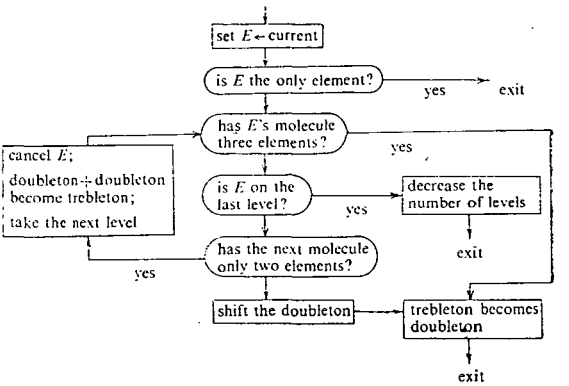

Fig. 11. Logical structure of procedure delete current

9. Experimental results

We chose the execution times of the typical call insert (scan (t), P, t) to compare. The algorithms used are exactly those contained in the Appendix. The test were performed on a Control Data 3300 computer.

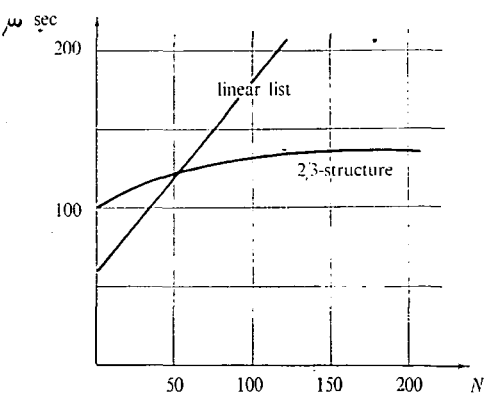

Fig. 12. Comparison of scan + insertion times

In the test the parameter t in the call insert (scan (t), P, t) was exponentially distributed. As it is known the increase of the line describing the performance of the linear list depends on the choice of the distribution of t, but our experimental results showed that using homogeneous indexes the performance was quite independent of it.

The deletion procedure proved to be about 1.8.-times slower than the simple linear one with a small variance. (This is 1.85 for the general version.)

These results altogether designate a limit of about 100 events below which the simple linear method may well be used and the relative performance of the homogeneous structure fastly increases beyond.

10. Further problems

There are several further questions which seem very useful to study. Constructing more and more effective structures is important not only for discrete event simulation but for any other linked structures sorted by continuous keys too. We propose the following problems:

1. We used duplicated events at the higher levels indexing the original ones of the lowest level. We have seen that it leads to an average number $3N/4$ of duplications. However, it seems possible that using certain further pointers even a linear list can be supplied by a crafty additional structure ensuring effective search without duplicated events.

2. The performance can be improved by not cancelling the events when their activity is terminated but leaving them for some kind of "garbage collector". It will not decrease the effectivity of the procedure scan if it is always starts from the end of the list.

3. Finally we notice that during a discrete event simulation the region of time values interested is permanently being shifted to higher values. If we could describe stochastically the distributions of events resulted by such a consistent shift, then it would be possible to develop special structures based on this probabilistic behaviour.

Appendix

In this part we use the SIMULA 67 formalism [4] but the reader needs not to know it precisely because the denotations used are self-explanatory.

A. Algorithms for linear lists

```
ref (event) procedure scan (t); real t;
    begin  ref (event) F;
           F: — last event;
           for F: — F while F. evtime>t do F: — F.pred;
           scan: — F;

    end;

procedure insert (E, P, t);
    ref (event) E; ref (process) P; real t;
```

```
begin ref (event) F;
      F:— new event (t, P);
      F.pred:— E;
      F.suc:— E.suc;
      F.pred.suc:— F.suc.pred:— F;
end;
```

(The last procedure body illustrates how to set the new pointer references. If the subclass relation *link* **class** *event* is assumed using the standard SIMULA list processing facilities the whole procedure body may be simply replaced by **new** event (*t*, *P*). follow (*E*).)

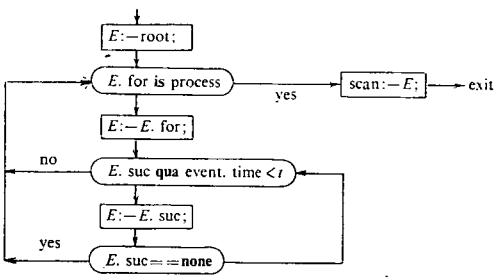

Fig. 13. **ref** (event) **procedure** scan (*t*);

```
procedure delete (E); ref (event) E;
  begin
      E. suc. pred:— E. pred;
      E. pred. suc:— E. suc;
      E. suc:— E. pred:— none;
  end;
```

(The procedure body is equivalent to the standard SIMULA procedure *E*.out.)

B. Algorithms for the 2/3-structure

The structure of the event notice could be defined by

```
class event (back,for, time);
  ref (event) back, for; real time;
  begin
      ref (event) suc,pred;
  end;
```

but for practical reasons in the algorithms it is replaced by the following version using the SIMULA linkage possibilities:

```
link class event (back,for, time);
  ref (event) back,for; real time;;
```

We use the global pointers:

ref (event) root, current;

and the initiation is

root: — current: — **new** event (**none**, **new** process, 0);
root.into (**new** head);

The algorithms are contained by figures 13—16.

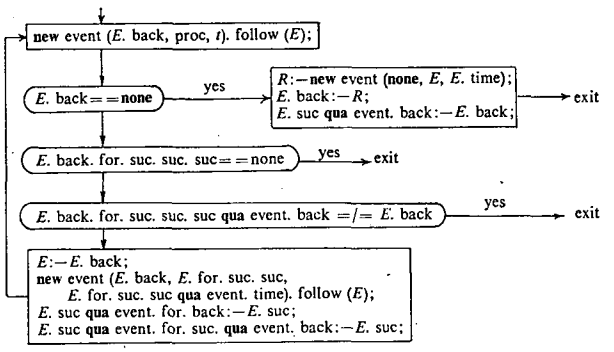

Fig. 14. **procedure** insert (*E*, proc, *t*);

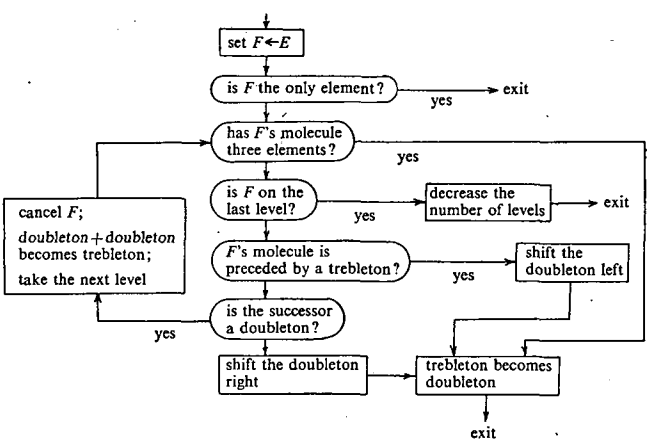

Fig. 15. **procedure** delete current;

Finally we give the logical structure of the general procedure delete. The diagram may readily be programmed in a similar way as the procedure delete current.

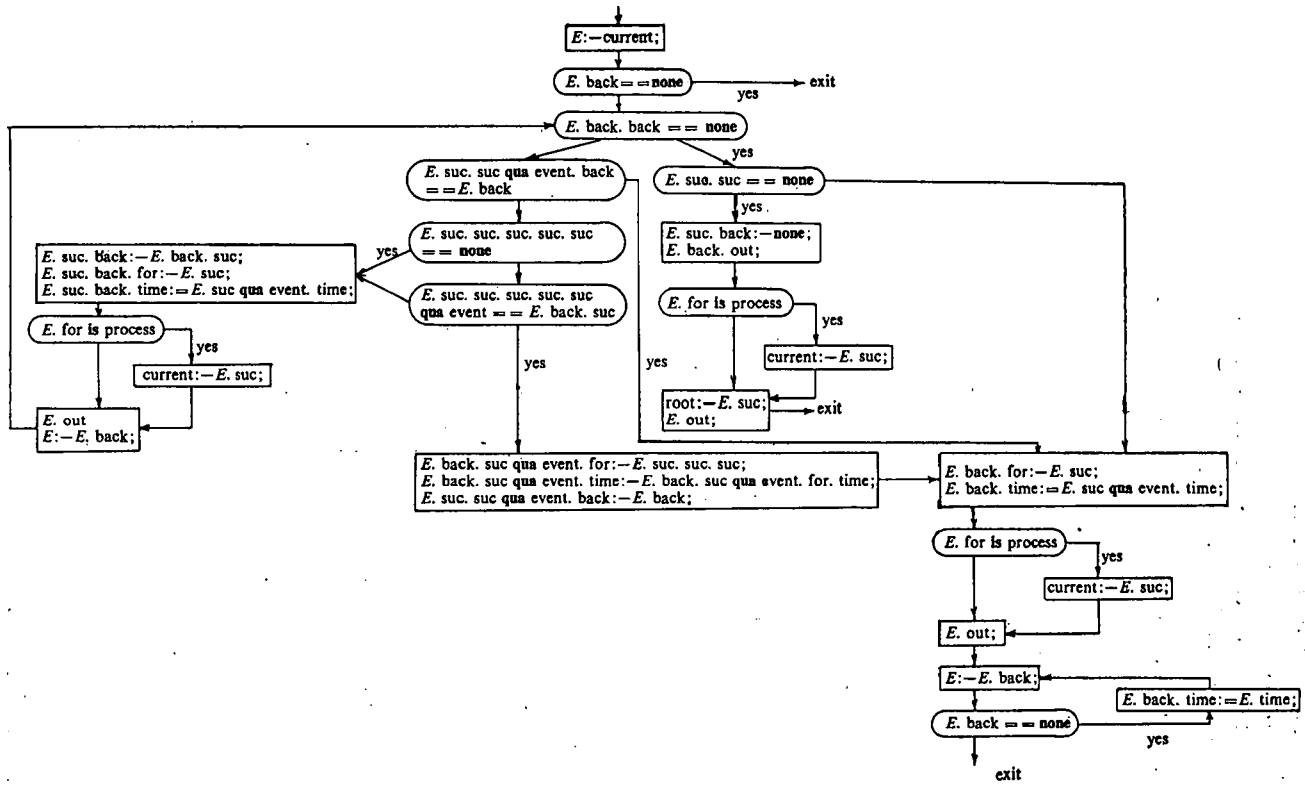

Fig. 16. Logical structure of procedure delete (*E*);

Abstract

Simulation event scheduling algorithms based on linear lists are generally used in simulation languages. Under heavy traffic conditions these algorithms have poor performance. The best of the more complicated algorithms having proposed to improve it is the indexed list method.

In this paper we introduce a multiple-indexed list structure of fully homogeneous nature to eliminate certain disadvantages of the use of static indexes and to gain further improvements. We give all the necessary program routines in detail. The sense of probabilistic behaviour is also given and simulation results are presented to make a comparison with linear list algorithm.

COMPUTER AND AUTIMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
H-1502 BUDAPEST, HUNGARY

References

[1] BUXTON, J. N. (ed.), *Simulation programming languages*, North—Holland, Amsterdam, 1968.
[2] GENUYS, F. (ed.), *Programming languages*, Academic Press, N. Y., 1968.
[3] GORDON, G., *System simulation*, Prentice — Hall, Englewood Cliffs, N. J., 1968.
[4] DAHL, O. J., B. MYHRHAUG, K. NYGAARD, SIMULA *67 common base language*, S22, Norwegian Computing, Centre, 1967.
[5] KIVIAT, P. J., R. VILLANVEVA, H. H. MARKOWITZ, *The* SIMSCRIPT II *programming language*, Prentice—Hall, Englewood Cliffs, N. J., 1968.
[6] *General purpose simulation system* 360 — User's manual, H20—0326, IBM Corp., White Plain, N. Y., 1968.
[7] WIRTH, N., The programming language PASCAL, *Acta Informat.*, v. 1, 1971, pp. 35—63.
[8] MYHRHAUG, B., *Sequencing set efficiency*, A9, Norwegian Computing Centre.
[9] KNUTH, D. E., *The art of computer programming*, v. 3, Addison Wesley, Reading, Mass., 1973.
[10] VAUCHER, J. G. & P. DUVAL, A comparison of simulation event list algorithms, *Comm. ACM*, v. 18, 1975, pp. 223—230.

(Received March 11, 1976)