JOURNAL OF COMPUTATIONAL PHYSICS 22, 189–214 (1976)

Automatic Generation of Feynman Graphs in QED*

TATEAKI SASAKI

The Institute of Physical and Chemical Research, Wako-Shi Saitama, 351, Japan

Received July 1, 1975; revised May 11, 1976

A computer program which generates all Feynman graphs for given initial and final states and an interaction order in QED (Quantum Electro-Dynamics) is described. The program is based on a topological analysis and has two features: short execution time owing to minimization of the number of graphs to be constructed, and elimination of topologically equivalent graphs as well as physically unused ones. Many elimination tests are applied to each graph after its construction without referring to other graphs. Hence, even a large number of graphs can be generated within reasonable time and memory size. Applications of the program to several interesting processes in physics are briefly reported.

1. INTRODUCTION

A good agreement of QED predictions to date with experimental results encourages further calculations in this field. The higher order calculations are, however, very tedious to do and many computer programs have been written to make them as automatic as possible (for example, SCHOONSCHIP by Veltman, REDUCE by Hearn, and ASHMEDAI by Levine [1]). Furthermore, automatic generation of Feynman graphs also has been attempted, and the descriptions of three such programs have been published so far. One, by Campbell and Hearn, relies on Wick's contraction theorem; the second, by Calmet and Perrottet, is based on a generating functional formalism. The third is by Perrottet and uses a combinatorial analysis [2].

An important point in generating graphs is not only the construction of all necessary graphs but also the elimination of topologically equivalent graphs, due to the fact that many more graphs than necessary are produced by ordinary graph construction methods. By relabeling an n-point graph, we can produce $(n! - 1)$ graphs which are topologically identical to the original one; in the worst case, unnecessary graphs are $(n! - 1)$ times as numerous as the necessary ones. Further-

* Work supported in part by the Project, "Advanced Information Processing of Large Scale Data over Broad Area," sponsored by the Ministry of Education, Japan.

more, there are many graphs which are not used in physics. The programs mentioned above are very useful, but not entirely satisfactory, in that the elimination of unnecessary graphs is insufficiently performed in Campbell and Hearn, and the comparison method in Calmet and Perrottet requires excessive memory for storing graphs for comparison. This memory requirement will become very severe when many, say a thousand topologically different graphs, are generated.

One way to avoid this difficulty is to check the graph's topology and determine, without referring to other graphs, whether or not the graph should be eliminated. One may wonder if all unnecessary graphs can be eliminated in this approach. As we will see later, this is in fact the case for the QED interaction. Moreover, this approach will save computing time as well; the computer processing time in our approach is roughly proportional to n_{graph}, the number of graphs to be constructed in memory. On the other hand, it is roughly proportional to $n_{\mathrm{graph}} \cdot N_{\mathrm{graph}}$ in the comparison method, where N_{graph} is the number of graphs to be output. When applying a program constructed along this line to actual problems, the author found it very powerful. It has generated about 12,000 graphs within the bounds of reasonable time and standard memory size. The program has been written in LISP, because its data structure is considered to be one of the best suitable for handling graphs. However, knowledge of LISP is not required for reading this paper.

Section 2 is devoted to terminology. In Section 3 we describe a graph construction algorithm. We will restrict ourselves to treating only the QED interaction, because in other cases the graph elimination algorithm will be considerably more complicated. This restriction will, however, enable us to make a graph construction algorithm which is not only efficient but also effective for classifying the generated graphs. A graph elimination algorithm is explained in Section 4. Results of applications of our program to actual problems are reported in Section 5. Applicability of our idea to more complicated type of interactions will also be discussed there briefly.

2. Preliminary

2.1. *Definitions*

We first define some terms for QED Feynman graphs, some of which have meanings different from those in conventional graph theory.

(1) *Vertex* and *order*. A vertex (or *point*) is a node of a graph. The order of a graph is the number of all vertices in the graph.

(2) *Line*, *initial* and *final states*. A line is a branch (or edge) of a graph. A Feynman graph is defined with the concept of an initial and a final state, which are

the states at $-\infty$ and $+\infty$, respectively. Each end of a line may either start at an initial state, end at a final state, or otherwise be incident to a vertex. In the Feynman graph considered here, both ends of a line must not be incident to the same vertex.

(3) *El-line* and *ph-line*. There are two kinds of lines, an electron line and a photon line (an el-line and a ph-line), drawn by a solid line and a dotted line, respectively.[1] An el-line has a direction, along which an electron (or negative charge) flows. At each vertex, three lines are adjoined: one ph-line and two el-lines, one arriving and the other leaving, conserving the charge current.

(4) *External* and *internal* lines. A line is external if it joins to one or both of an initial and a final state, otherwise it is internal.

(5) *Chain* and *loop*. These terms are applied only to el-lines. A chain is a (nonempty) set of all el-lines which are adjoined to each other, one by one. A chain has a direction, since each el-line has a direction. A loop is a special chain which starts and ends at the same vertex.

(6) *Skeleton* of a graph is a subgraph containing all chains but no ph-lines in the graph.

(7) *Vertex labeling number* (hereafter abbreviated as *vln*). A vln is a label of a vertex. All vertices in a graph are distinctively labeled with numbers 1, 2,..., N (= the order of the graph). Vertices in a chain are numbered in ascending order along its direction.

(8) *Base* of a chain is the vertex having the smallest vln in the chain.

(9) *Length* from vertex A to vertex B in a chain is the number of el-lines counted from A to B along the direction of the chain.

(10) *Height* of a vertex in a chain is the length from the base of the chain to the vertex.

(11) *Ph-neighbor*. A vertex is the ph-neighbor of the other if both are joined to each other by an internal ph-line.

(12) *Equivalent*. Chains in a graph (or vertices in a loop) are equivalent *w.r.t. a given criterion* if they have topologically the same properties w.r.t. the criterion. Two corresponding vertices in two equivalent chains are also said to be equivalent. This term (as well as "periodic" and "period"; see below) has crucial importance in our discussion. It should be noted that the precise meaning of "equivalence" changes as the criterion to be used changes (cf. Fig. 1).

(13) *Periodic* loop, *period*, *quasi-base*, and *quasi-height*. Suppose that all vertices of a loop are cyclically relabled (cf. Fig. 2). If the relabeled loop has

[1] We do not consider muons in this paper. Graphs including muon lines will be easily obtained from electron–photon graphs.

topologically the same vln configuration as the original one *w.r.t. a given criterion*, then the loop is periodic w.r.t. the criterion. Some vertices in a periodic loop are therefore mutually equivalent w.r.t. the same criterion. The period is the nonzero minimum length between two such equivalent vertices. A vertex of a loop equivalent to the base of the loop is called a quasi-base. The quasi-height of a vertex in a periodic loop is its height, modulo the period of the loop; i.e., the length measured from the "nearest" quasi-base or the base.

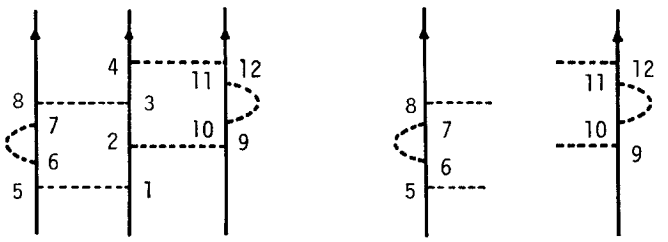

FIG. 1. The tops and the bottoms of these graphs are in the final and the initial states, respectively. This convention will always be used in this paper. The chains (5 6 7 8) and (9 10 11 12) are not equivalent if they are observed in combination with all other chains (l.h.s. graph), but equivalent when only their inner-chain topologies are observed (r.h.s. subgraphs).

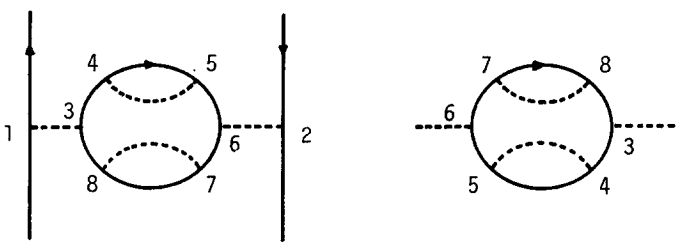

FIG. 2. The loop (3 4 5 6 7 8) is nonperiodic when observed in the whole graph, but periodic with period 3 when observed separately; a relabeled loop (r.h.s. subgraph, which is obtained from the loop in the l.h.s. graph by the cyclic permutation ($3 \to 6$, $4 \to 7$, $5 \to 8$, $6 \to 3$, $7 \to 4$, $8 \to 5$) on their vln's) is equivalent to the original loop. In this case, vertices 4 and 7 have the same quasi-height 1 though they have heights 1 and 4, respectively.

(14) *Preferred order* of vertices (or chains). Vertices (or chains) in a graph can be ordered. Vertices are in the preferred order if their vln's are in ascending order. Chains are in the preferred order if the vln's of their bases are in ascending order.

(15) *Block*. Any set of chains in a graph can be separated into subsets, or blocks, such that each block constitutes a connected subgraph and any two chains belonging to different blocks are disconnected.

(16) *Base, length, height, equivalent, periodic, period, quasi-base, quasi-height,* and *preferred order* are also defined for blocks. The base of a block is the vertex having the smallest vln in it. Equivalent blocks are similarly defined as equivalent chains. Blocks in a graph are in the preferred order if the vln's of their bases are in ascending order. Other terms will be explained later, when necessary.

(17) *e-set* and *p-set.* These are abbreviations for "set of equivalent chains" or "equivalent blocks," and for "periodic loops" or "periodic blocks," respectively. A graph may contain many e-sets.

2.2. *Brief Sketch of the Program*

For given initial and final states and an interaction order, a QED Feynman graph is uniquely determined when its chain structure (i.e., its skeleton) and the connections of all its ph-lines are determined. In our program, a chain and a ph-line are represented by an ordered set of vln's and a pair of vln's (or a pair of a vln and a number representing an initial or a final ph-line), respectively. A graph is represented by these sets of numbers. It should be noted that some graphs which are distinctively represented may be topologically identical.

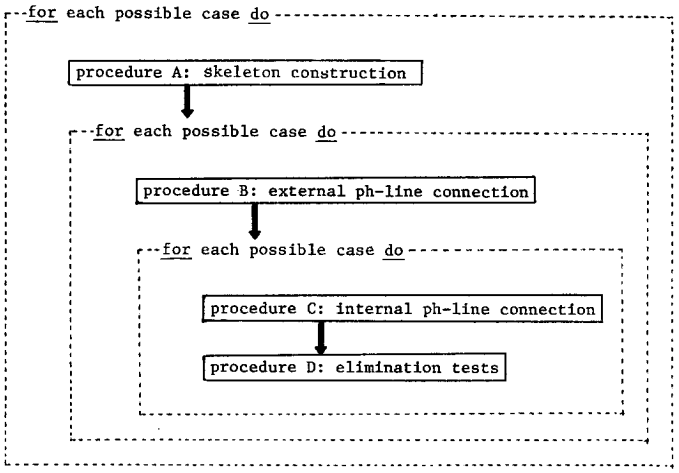

FIG. 3. Simplified flow diagram of the program.

Although details are given later, we outline the function of our program here using Fig. 3, a simplified flow diagram. Procedure A constructs a skeleton of a graph to be constructed. Only all topologically different skeletons are constructed. We label vertices in each skeleton so that they can be ordered uniquely (a preferred order of vertices). Next, procedure B attempts all possible connections of external ph-lines, if any, to the skeleton. Provisions are made so that only a specific

connection among all topologically equivalent ones is kept. Procedure C then attempts all pairings, by internal ph-lines, of ph-line vacant vertices in the subgraph. The number of pairings made by our program is $(2m-1)!!$ for $2m$ ph-line vacant vertices, which is reasonably small, although not all graphs constructed are topologically different. In procedure D all unnecessary graphs are eliminated.

It may be seen from Fig. 3 that no more than one graph is stored simultaneously in memory. Furthermore, the program proceeds so that the graphs having the same skeleton and external ph-line connections will be sequentially generated, and that the skeleton and the external ph-line connections will be systematically changed.

3. Graph Construction Algorithm

We restrict ourselves to constructing only graphs which contain no loops with odd number of vertices. (In our actual program, this restriction is overridden if "OFFFURRY" is input as a command.) Figure 4 is a flow diagram showing all steps of the graph construction algorithm, where we have used solid and dotted lines to describe, as indicated, the nested flow structure, such as shown in Fig. 3.

3.1. *Skeleton Construction*

Let the order of a graph be N and the numbers of electrons, positrons and photons in an initial state be n_{el}, n_{ps}, and n_{ph}, and in a final state be n'_{el}, n'_{ps}, and n'_{ph}, respectively. These numbers are given to the program as input data, and they must satisfy the following conditions (to be checked by a function "jobinput").

$$N \geqslant 1, \qquad n_{\mathrm{el}}, \text{ etc.}, \geqslant 0, \tag{1}$$

$$N \geqslant n_{\mathrm{ph}} + n'_{\mathrm{ph}}, \tag{2}$$

$$N \geqslant (n_{\mathrm{el}} + n_{\mathrm{ps}} + n'_{\mathrm{el}} + n'_{\mathrm{ps}})/2, \tag{3}$$

$$n_{\mathrm{el}} + n'_{\mathrm{ps}} = n_{\mathrm{ps}} + n'_{\mathrm{el}}, \tag{4}$$

$$N - (n_{\mathrm{ph}} + n'_{\mathrm{ph}}) = \text{an even integer.} \tag{5}$$

If (2) and/or (3) are not satisfied, some particles go through the graph without interactions. Condition (4) represents the charge conservation. Condition (5) says that the number of vertices not being connected by external ph-lines must be even, since each internal ph-line joins to two vertices.

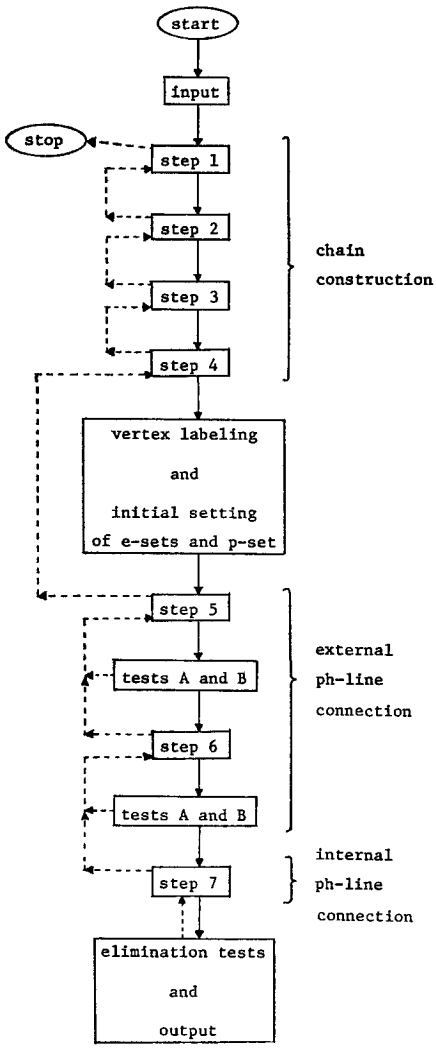

FIG. 4. Graph construction steps. Each step is accompanied with a set of numbers, which is advanced by one degree when we enter that step. If we succeed in producing a new set of numbers, we proceed to the next step (solid line). If all possible sets are exhausted, we go back to the preceding step (dotted line).

All chains are classified into the following five types (see Fig. 5).

Type 1: chains running from initial electrons to final electrons,
Type 2: chains running from initial electrons to initial positrons,
Type 3: chains running from final positrons to final electrons,
Type 4: chains running from final positrons to initial positrons,
Type 5: loops.

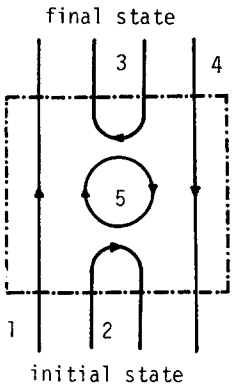

FIG. 5. Five types of chains.

Let the numbers of the above five types of chains in a graph be n_1, n_2, n_3, n_4, and n_5, respectively. Because of (4), only one of n_i, $1 \leqslant i \leqslant 4$, is independent, and others are determined from it as follows. Choosing n_1 as the independent number, we find a possibility of the set of n_i, $1 \leqslant i \leqslant 4$, as

$$n_{\min} \leqslant n_1 \leqslant n_{\max}, \tag{6a}$$

$$n_2 = n_{\mathrm{el}} - n_1, \tag{6b}$$

$$n_3 = n'_{\mathrm{el}} - n_1, \tag{6c}$$

$$n_4 = n_1 - (n_{\mathrm{el}} - n_{\mathrm{ps}}), \tag{6d}$$

where

$$n_{\min} \equiv \mathrm{Max}[0, n_{\mathrm{el}} - n_{\mathrm{ps}}], \tag{7a}$$

$$n_{\max} \equiv \mathrm{Min}[n_{\mathrm{el}}, n'_{\mathrm{el}}]. \tag{7b}$$

Setting an initial value of n_1 to $(n_{\min} - 1)$, we proceed to Step 1.

Step 1. We increase the value of n_1 by 1 and determine n_2, n_3, and n_4 from (6b), (6c), and (6d). If n_1 exceeds $n_{\max}$ (i.e., all cases are exhausted) then stop the computation, otherwise go to Step 2.

Step 2. We distribute N vertices among five types of chains. Assume that the ith type chains have received N_i vertices, where

$$N_i \geqslant n_i, \qquad 1 \leqslant i \leqslant 4, \tag{8a}$$

$$N_5 = \text{a nonnegative even integer}, \tag{8b}$$

$$\sum_{i=1}^{5} N_i = N. \tag{8c}$$

The program assigns $\{N_1 = N - (n_2 + n_3 + n_4), N_2 = n_2, ..., N_4 = n_4, N_5 = 0\}$ as the first distribution of these numbers. Succeeding sets of numbers to be assigned are generated by a function "proceed1." When we succeed in generating a new set, we go to Step 3. If all possible sets are exhausted, we return to Step 1. The function proceed1 is one of the "proceed" functions used in our program, each of which is used to succeedingly exhaust the combinations of numbers in a set. We present a definition of proceed1 as a representative example, in a language which is intuitively easy to undertand, where $N' = N - (n_1 + n_2 + n_3 + n_4)$, $N_1' = N_1 - n_1, ..., N_4' = N_4 - n_4$, and $N_5' = N_5$,

```
proceed1 [N'; N1'; N2'; N3'; N4'; N5'] = prog [ [i]
        i := 2;
A       [if i ≠ 5 then Ni' := Ni' + 1;
          else N5' := N5' + 2];
        N1' := N' − (N2' + N3' + N4' + N5');
        [if N1' ≥ 0 then return [N1'; N2'; N3'; N4'; N5'];
          else if i = 5 then return [NIL]];
        Ni' := 0;
        i := i + 1;
        go [A]].
```

Each possible set $\{N_1, ..., N_5\}$ satisfying (8) is thus generated only once by proceed1. When proceed1 returns "NIL," we know that all possible cases are exhausted.

Step 3. For each i, $1 \leqslant i \leqslant 4$, N_i vertices are distributed among n_i chains of type i, as follows. Let $N_{i,j}$, $1 \leqslant j \leqslant n_i$, be the numbers of the vertices distributed to the chains. We impose restrictions

$$\sum_{j=1}^{n_i} N_{i,j} = N_i, \tag{9a}$$

$$N_{i,j} \geqslant N_{i,j+1} \geqslant 1. \tag{9b}$$

For each i, $1 \leqslant i \leqslant 4$, the program assigns $\{N_{i,1} = N_i - n_i + 1,\ N_{i,2} = 1,\ldots,\ N_{i,n_i} = 1\}$ as the first distribution of these numbers. Succeeding sets of numbers to be assigned are generated only once for each possible set by a function "proceedchn."

Step 4. The chains of type 5 (i.e., the loops) have been assigned with N_5 (an even integer) vertices in Step 2. Hence, the maximum number of loops is $n_5 = N_5/2$. If $n_5 > 0$, we distribute $N_{5,1},\ldots, N_{5,n_5}$ vertices to loops $1,\ldots, n_5$, respectively, imposing restrictions

$$N_{5,j} = \text{an even integer}, \tag{10a}$$

$$\sum_{j=1}^{n_5} N_{5,j} = N_5\,, \tag{10b}$$

$$N_{5,j} \geqslant N_{5,j+1} \geqslant 0. \tag{10c}$$

A function "proceedloop" successively generates each possible set $\{N_{5,1},\ldots, N_{5,n_5}\}$ only once. The starting set of values assigned is $\{N_{5,1} = N_5\,,\ N_{5,2} = 0,\ldots,\ N_{5,n_5} = 0\}$, that is, the case with only one loop with N_5 vertices.

The steps so far are for skeleton construction. It should be noted that our program constructs each possible skeleton only once.

3.2. *Vertex Labeling and Initial Setting of e-Sets and p-Set*

We label all vertices in a skeleton constructed as above with numbers $1, 2,\ldots, N$, using the following rules: (i) All vertices in the skeleton are distinctively labeled; (ii) each chain is labeled with sequentially increasing numbers along its direction; (iii) numbers assigned to chains of type i are smaller than those assigned to chains of type $(i + 1)$; (iv) numbers assigned to a chain with more vertices are smaller than those assigned to another chain with fewer vertices, if both are of the same type.

Next, we set up the e-sets and the p-set of the skeleton labeled above, using all its topological properties. Since ph-lines are not yet taken into consideration, and since chains in the skeleton are disconnected from each other, it is clear that the chains of the same type with the same number of vertices are equivalent to each other, and that all loops are periodic with period 1 (i.e., all vertices but the base in every loop are quasi-bases).

3.3. *Predicate Function "Advancedp"*

One of the most important functions in our program is advancedp. It has two ordered sets of numbers as its arguments, and is defined as

advancedp $[(k_1, k_2, \ldots, k_m); (k_1', k_2', \ldots, k_{m'})]$
$= [$*if* arguments are null (i.e., $m = 0$) *then* FALSE;
else if $k_1 > k_1'$ *then* FALSE;
else if $k_1 < k_1'$ *then* TRUE;
else advancedp $[(k_2, \ldots, k_m); (k_2', \ldots, k_{m'})]]$,

where ordered sets are denoted by pairs of parentheses. In our program, this function is used to define an ordering among ordered sets of numbers. Similarly, an ordering of ordered families of ordered sets of numbers are defined by a function "mltadvancedp":

mltadvancedp $[(K_1, K_2, \ldots, K_m); (K_1', K_2', \ldots, K_{m'})]$
$= [$*if* arguments are null (i.e., $m = 0$) *then* FALSE;
else if advancedp$[K_1; K_1'] =$ TRUE *then* TRUE;
else if advancedp$[K_1'; K_1] =$ TRUE *then* FALSE;
else mltadvancedp$[(K_2, \ldots, K_m); (K_2', \ldots, K_{m'})]]$.

3.4. *Connection of ph-Lines*

In order to connect n_{ph} ph-lines from the initial state, for example, we must select n_{ph} numbers out of vln's, 1, 2,..., and N. To perform this, we use a function "pickanddelete," which has two ordered sets of positive integers as its arguments and works as follows:

pickanddelete $[(m_1, m_2, \ldots m_j); (n_1, n_2, \ldots, n_k)]$ = prog $[[i; x; p; r]$
$i := 1$; $p :=$ (nothing); $r := (n_1, n_2, \ldots, n_k)$;
A [*if* $i > j$ *then* return $[p; r]]$;
$x := m_i$th element of r;
append x to p as the last element;
delete x from r;
$i := i + 1$;
go [A]],

here $j \leqslant k$ and $1 \leqslant m_s \leqslant k - s + 1$.

Step 5. In this step, all ph-lines from the initial state are connected, if $n_{\text{ph}} \neq 0$, to the skeleton constructed so far. Let an ordered set of integers $(m_1, m_2, \ldots, m_{n_{\text{ph}}})$ satisfy

$$1 \leqslant m_i \leqslant m_{i+1} \leqslant N - n_{\text{ph}} + 1. \tag{11}$$

Connect the n_{ph} ph-lines to vertices labeled with numbers belonging to the first set returned by pickanddelete$[(m_1, m_2, \ldots, m_{n_{\text{ph}}}); (1, 2, \ldots, N)]$. Note that, because

of (11), numbers in the first set returned are in the increasing order. Starting from set (1, 1,..., 1), a function "proceed3" generates each possible set $(m_1, m_2, ..., m_{n_{ph}})$ only once.

We note that, when there exists some equivalent vertices in a skeleton, the above method of the external ph-line connection may generate many topologically identical subgraphs (Fig. 6 illustrates this fact). Since chains in a resulting subgraph are still disconnected from each other, we can easily eliminate such unnecessarily duplicated subgraphs as follows. First, when the p-set of a subgraph is not null, we apply the following test to each loop in the set.

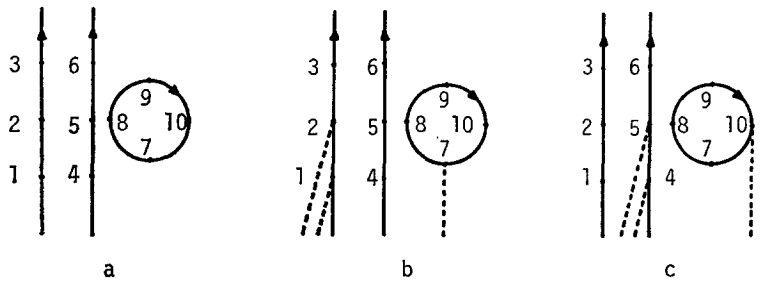

FIG. 6. Allowed and forbidden connections of external ph-lines. Fig. 6a is a skeleton containing two equivalent chains and one periodic loop. Figs. 6b and 6c are subgraphs arising from 6a by the connections of external ph-lines. They are topologically the same. Owing to Tests A and B, 6b is allowed but 6c is forbidden.

Test A. For each vertex in a given periodic loop we assign a number which is either 1 if the vertex is connected with a ph-line, or 0 otherwise. Let the loop contain k vertices and be of period p. Suppose $(x_1, x_2, ..., x_k)$ is the ordered set of numbers thus assigned, where x_j is for the vertex of height j. If advancedp$[(x_1, x_2, ..., x_k); (x_{ip+1}, ..., x_k, x_1, ..., x_{ip})] =$ TRUE for any value of i, $1 \leqslant i \leqslant (k/p) - 1$, then the subgraph is eliminated, otherwise the period of the loop is replaced by the period of $(x_1, x_2, ..., x_k)$, which is evaluated as follows.

```
periodl [(x_1, x_2, ..., x_k); p] = prog [[y; q]
        q := p;
A       y := (x_{q+1}, x_{q+2}, ..., x_k, x_1, ..., x_q);
        [if y = (x_1, x_2, ..., x_k) then return [q]];
        q := q + p;
        [if q > (k + 1)/2 then return [k];
        else go [A]]].
```

If the period becomes k, we delete the loop from the p-set.

Next, when the e-sets of the subgraph are not null, we apply the following test to each e-set:

Test B. For each vertex in each chain in a given e-set, we assign a number which is either 1 if the vertex is connected with a ph-line, or 0 otherwise. Let the e-set contain l chains $c_1, c_2, \ldots, c_l$, each of which contains k vertices. Then we have l ordered sets $y_1, y_2, \ldots, y_l$, each of which contains k numbers thus assigned. Let the ordering of $c_1, c_2, \ldots, c_l$ be in accordance with the preferred order. If advancedp$[y_i ; y_{i+1}]$ = TRUE for any value of i, $1 \leqslant i \leqslant (l-1)$, then the subgraph is eliminated, otherwise $c_1, c_2, \ldots, c_l$ are grouped into smaller e-sets according to the following criterion.

[*if* advancedp$[y_i ; y_j]$ = advancedp$[y_j ; y_i]$ = FALSE
then c_i and c_j are equivalent to each other;
else c_i and c_j are not equivalent to each other].

Step 6. All external ph-lines to the final state are connected, if $n'_{\mathrm{ph}} \neq 0$, from ph-line vacant vertices of the subgraph constructed so far. The method is the same as that in Step 5, except that the first argument of pickanddelete is an ordered set of integers $(m_1', m_2', \ldots, m'_{n'_{\mathrm{ph}}})$ satisfying

$$1 \leqslant m_i' \leqslant m'_{i+1} \leqslant N - n_{\mathrm{ph}} - n'_{\mathrm{ph}} + 1, \tag{12}$$

and that its second argument is the second set returned by pickanddelete in Step 5.

After this step, just as after Step 5, Tests A and B are applied again to the subgraph to check this ph-line connection. Because of conditions (11) and (12) and Tests A and B, every surviving subgraph at this stage is topologically distinct from each other. Furthermore, equivalent chains in a graph are such that they are not only of the same type with the same number of vertices, but also topologically the same w.r.t. the connections of external ph-lines.

The final step of the graph construction is to connect the internal ph-lines to yet ph-line vacant vertices. Let vln's of such vertices be $r_1, r_2, \ldots, r_{2m}$ ($2m = N - n_{\mathrm{ph}} - n'_{\mathrm{ph}}$), and let $r_i < r_{i+1}$.

Step 7. Let an ordered set of integers $(q_1, q_2, \ldots, q_m)$ satisfy

$$i + 1 \leqslant q_i \leqslant 2m - i + 1. \tag{13}$$

Let the first and the second sets returned by pickanddelete$[(q_1, q_2, \ldots, q_m); (r_1, r_2, \ldots, r_{2m})]$ be $(s_1, s_2, \ldots, s_m)$ and $(t_1, t_2, \ldots, t_m)$, respectively. For each i, $1 \leqslant i \leqslant m$, join each pair of vertices having vln's s_i and t_i by an internal ph-line. We call this procedure the pairing of vertices. Starting from set $(2, 3, \ldots, m+1)$, a function "proceed4" generates each possible set $(q_1, q_2, \ldots, q_m)$ only once.

We can see that $s_i < s_{i+1}$, $t_i < t_{i+1}$, and $s_i > t_i$ for all possible i, hence if sets $(q_1, q_2, ..., q_m)$ and $(q_1', q_2', ..., q_m')$ are different, then the resulting pairings are different. Therefore all different pairings are accomplished, only once for each, because (13) tells us that the number of sets to be generated by proceed4 is $(2m - 1)!!$, which is equal to the number of all different pairings of $2m$ different vertices. This step may, however, produce topologically identical graphs many times in general, because some vertices may be equivalent.

4. Graph Elimination Algorithm

Most procedures in this section will be described without detailed definitions, which the reader can grasp by referring to Tests A and B described in subsection 3.4. First of all, it should be noted that no graphs, except the one which is to be tested, are stored in the memory at one time.

4.1. *Elimination of Graphs Unused in Physics*

The graphs unused in physics are: (i) disconnected graphs; (ii) graphs containing electron and/or photon self-energy corrections to external lines; and (iii) improper self-energy graphs. Since their elimination is a mathematically simple procedure, we omit the discussion.

4.2. *Strategy for Graph Elimination*

We first note that the graphs to be tested are only those that contain nonnull e-sets and/or a p-set. The reason is as follows: Consider a subgraph composed of a skeleton and all external ph-lines. We have shown that all such subgraphs produced by our program are topologically distinct from each other; hence, so are any two Feynman graphs respectively constructed by two such subgraphs. Suppose that such a subgraph does not contain any nonnull e-sets and/or a p-set, then all its vertices are topologically distinct from each other. Thus, all Feynman graphs constructed from the subgraph by the internal ph-line connections are also

terizing the subgraph are not altered in Step 7. Figure 7 shows graphs which are topologically identical but different in the vln configurations. Since topologically identical graphs here are due to different internal ph-line connections, it is seen that vln configurations of mutually identical graphs are related to each other in one or both of the following ways: (i) exchange of some equivalent chains (for example, chains (1) and (2), or loops (9 10) and (11 12) in Fig. 7) and (ii) cyclic relabeling of all vln's of some periodic loops by multiples of their respective periods (for example, the permutation $(3 \to 6, 4 \to 7, 5 \to 8, 6 \to 3, 7 \to 4, 8 \to 5)$ in Fig. 7).

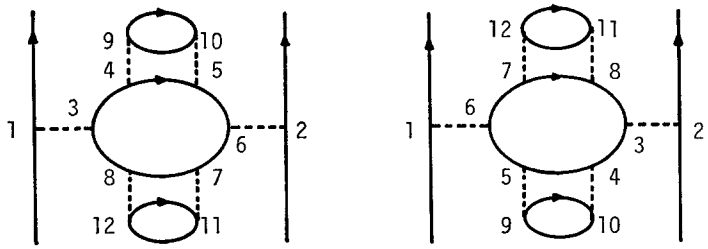

FIG. 7. Graphs which are topologically identical but different in vln configurations.

Next, we note that all chains or all vertices in loops of a graph produced by our program can be uniquely ordered according to the preferred order, which is common to all topologically equivalent graphs. We further note that chains or vertices which are mutually equivalent w.r.t. an equivalence criterion can be ordered if they are not equivalent w.r.t. another equivalence criterion; that is, some equivalent chains or vertices of a graph can be ordered according to the graph's topology. We call this the topological ordering. Of course, chains or vertices in a completely symmetric graph cannot be completely ordered according to the graph's topology only. However, a topological ordering which is relative to the preferred order can actually be determined, as we shall show later. Therefore, by eliminating those graphs which contain chains or vertices whose topological ordering relative to the preferred order is not consistent with the preferred order, we can eliminate all unnecessarily duplicated graphs. By representing the topological properties of chains (or blocks) as ordered sets of numbers (or ordered families of such sets), we can make use of function advancedp (or mltadvancedp) to order them. Note that this strategy has already been used in Tests A and B.

Thus our task is to investigate the topological properties of a given graph. Since the investigation will be performed through a number of steps, a test for chains in an e-set is typically as follows.

Test E. Order all chains in the e-set by examining the equivalence of them w.r.t. a given criterion. If the result is not consistent with the preferred order, then return "NO," otherwise separate the e-set into smaller e-sets according to the equivalence criterion just used.

This test is applied to every e-set. Similarly, a test for a periodic loop is typically as follows.

Test P. Determine the "ranking" of the base and quasi-bases of the loop by examining their equivalence w.r.t. a given criterion. If the base does not take the highest ranking, then return "NO." Otherwise redefine the period according to the equivalence criterion just used. If the loop turns out to be nonperiodic, delete it from the p-set.

This test is applied to every loop in the p-set. It should be noted that in order to

determine the ranking of a quasi-base, we must treat the quasi-base as if it were the base of the loop. The elimination procedure for a graph terminates when some test answers "NO" (then the graph is eliminated), or both the e-sets and the p-set become null (then the graph is printed out). Figure 8 is a flow diagram showing all of the elimination steps.

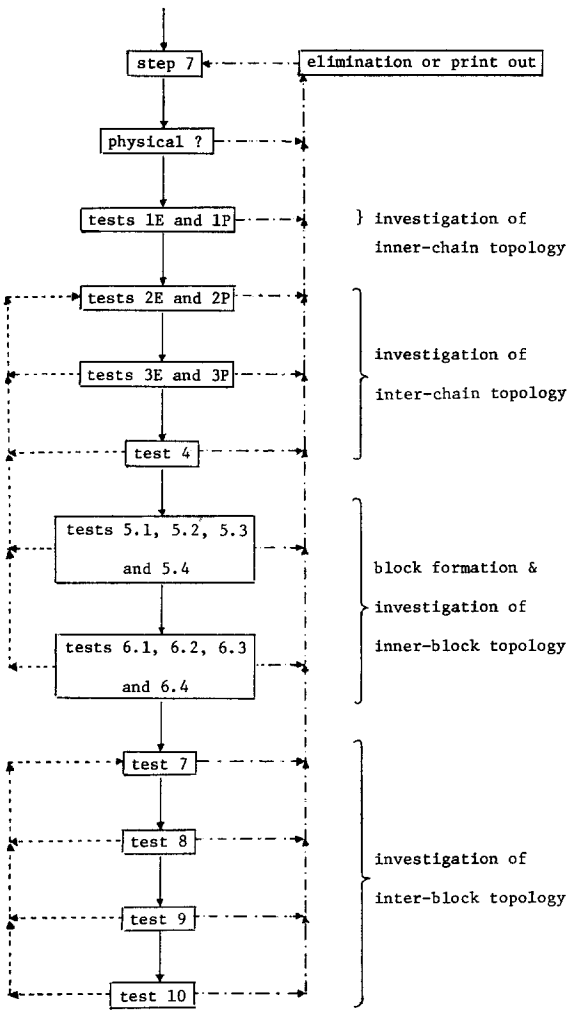

FIG. 8. Graph elimination steps. If some test answers "NO," the graph is eliminated (chained line). If both the e-sets and the p-set become null, the graph is printed out (chained line). If the e-sets and/or the p-set change, we go back to a younger test (dotted line). Otherwise, we go to the next test (solid line).

For convenience we classify every chain and vertex as follows.

Class 1: chain or vertex belonging to neither an e-set nor the p-set,
Class 2: chain or vertex belonging to the p-set but not to any e-set,
Class 3: chain or vertex belonging to an e-set but not to the p-set,
Class 4: chain or vertex belonging to both an e-set and the p-set.

4.3. *Elimination Algorithm, Part I*

The tests to be described here are those which can be applied to each chain without considering how other chains are connected.

CRITERION 1. Let two equivalent vertices in a graph be A and B, and their ph-neighbors be A' and B', respectively. If one of the following two conditions is satisfied, then A and B are distinguishable: (i) A and A' are in the same chain but B and B' are not; (ii) not only A and A' but also B and B' belong to their respective chains, but the length between A and A' is different from that between B and B'. Hence two equivalent chains, or the base and a quasi-base in a periodic loop, can be ordered if one of the above conditions holds for at least one pair of their equivalent vertices.

Tests 1E and 1P. Apply Tests E and P to the e-sets and the p-set, respectively, w.r.t. the equivalence criterion 1.

After these tests, equivalent chains in each surviving graph will have the same inner-chain topology (cf. Fig. 9).

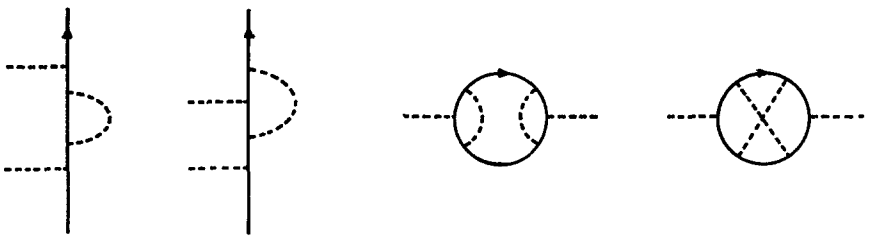

FIG. 9. Chains and loops distinguishable by Tests 1E and 1P.

CRITERION 2. Two equivalent vertices are distinguishable, and hence can be ordered, if their ph-neighbors satisfy one of the following conditions: (i) they do not belong to the same class; (ii) they belong to class 1; (iii) they are in class 2 and they belong to different loops; (iv) they are both in class 3 or in class 4, and they belong to different e-sets.

Tests 2E and 2P. Apply Tests E and P to the e-sets and the p-set, respectively, w.r.t. the equivalence criterion 2.

During these tests the graphs containing vertices whose ph-neighbors are in class 1 are either eliminated or printed out. Furthermore, ph-neighbors of any two equivalent vertices in each surviving graph belong either to the same loop if they are in class 2, or to the same e-set otherwise.

CRITERION 3. Two equivalent vertices are distinguishable, and hence can be ordered, if their ph-neighbors satisfy one of the following two conditions. (i) They belong to class 2 or to class 4, and their quasi-heights are different; (ii) they belong to class 3 and their heights are different.

Tests 3*E and* 3*P.* Apply Tests E and P to the e-sets and the p-set, respectively, w.r.t. the equivalence criterion 3.

After these tests the ph-neighbors of any two equivalent vertices in every surviving graph have the same quasi-height if they are in class 2 or in class 4, or have the same height if they are in class 3.

Since two or more vertices in a periodic loop are equivalent to each other, the number of ph-lines adjoined to both of two given loops in an e-sets may not be the same for all such similar pairs. Figure 10 shows an example. Note that this case arises if and only if loops are in class 4.

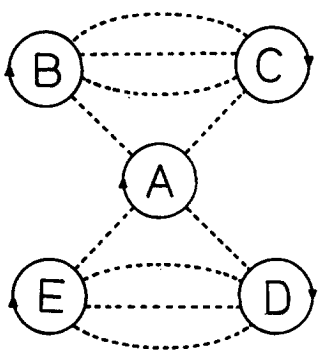

FIG. 10. Five loops A, B, C, D, and E are mutually equivalent w.r.t. criteria 1, 2, and 3, but A is apparently distinct from others.

CRITERION 4. Take all possible pairs of loops belonging to class 4. For each pair, obtain the number of ph-lines adjoined to both loops. Suppose we have n loops belonging to class 4, then each loop is assigned with a set of $(n - 1)$ such numbers (possibly with element duplications). If any two equivalent loops are assigned with different sets, then they are distinguishable, and hence can be ordered.

Test 4. Apply Test E to the e-sets containing loops belonging to class 4 w.r.t. the equivalence criterion 4.

4.4. *Elimination Algorithm, Part II*

To make a further investigation of a graph, we must consider more than one chain simultaneously. Our approach here is to separate the p-set and each of the e-sets into blocks. Figure 11 illustrates this aptly.

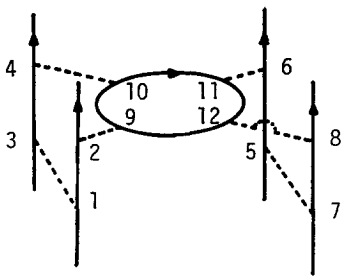

FIG. 11. Chains (1 2), (3 4), (5 6), and (7 8) are completely equivalent if we observe them separately, but set {(1 2), (3 4)} is clearly distinct from set {(1 2), (5 6)}.

We first separate the p-set into blocks. What we must carry out at this stage are (i) to examine how loops in the p-set are grouped into blocks, and (ii) to examine how they are connected within each block. The latter becomes trivial if the block is composed of a single loop. We can easily carry out the former with a suitably chosen criterion (and this is our Test 5.1), and the latter by treating each block as a simpler Feynman graph (and this is Test 5.3). However, we let Test 5.3 to be preceded by the following Test 5.2.

Test 5.2. For each loop in a given block, change cyclically its vln configuration by a multiple (including zero) of its period so that some quasi-base of the relabeled loop comes to the base position of the original one. Compare the block thus constructed with the original one by examining heights of ph-neighbors of all their vertices. Repeat this procedure until the sets of numbers characterizing the former are found to be more "advanced" than those for the latter (then return "NO"), or until all possible permutations of the vln configurations of the block are exhausted. Figure 12 illustrates this in some detail.

This test allows us to regard all loops in each block as nonperiodic so far as their inner-block vln configurations are concerned, which makes Test 5.3 fairly simple. This test is an example that shows a ranking, relative to the preferred order, of equivalent vertices can be determined even in a symmetric block.

We next separate each e-set containing nonperiodic chains into blocks. However, we need not separate the e-sets containing periodic loops because their constituents are already grouped into blocks. We again examine how chains are grouped into blocks (Test 6.1) and how they are connected within each block (Tests 6.2 and 6.3)

(cf. Fig. 13). After these tests, a graph can be regarded as being composed of blocks only. We note that blocks having different inner-block topologies can be ordered. If the result is not consistent with the preferred order of blocks, we eliminate the graph (Tests 5.4 and 6.4). Furthermore, since some blocks of a graph may have the same inner-block topology, sets of equivalent blocks are defined.

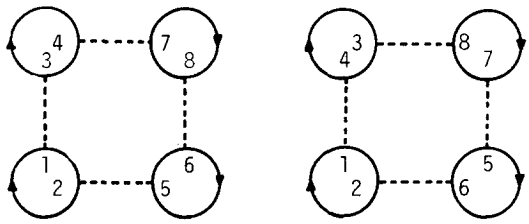

FIG. 12. Blocks composed of periodic loops. Loops in these blocks are of period 1 up to Test 4 and are indistinguishable. According to Test 5.2, 12a and 12b are characterized by families of sets of numbers ((1 1) (1 1) (2 2) (2 2)) and ((2 2) (2 1) (1 2) (1 1)), respectively, where the numbers represent the heights of ph-neighbors of corresponding vertices in ((1 2) (3 4) (5 6) (7 8)). We can distinguish them by applying function mltadvancedp to these families of sets.

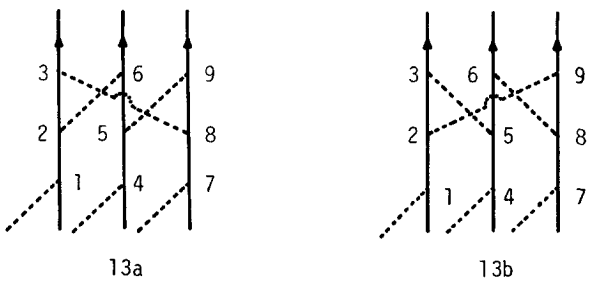

FIG. 13. Blocks composed of nonperiodic chains. Block 13a is allowed by Tests 6.2 and 6.3, but 13b is not.

Now, we proceed to study the topology which has not yet been investigated, i.e., the inter-block topology. Figure 14 shows an example of a graph to be investigated next. We readily observe that all blocks in it are "rotationally symmetric." This is a concept which is somwhat similar to that of periodic loops. In fact, rotationally symmetric blocks can be similarly treated as periodic loops under a suitable redefinition of terms used for loops. Here we give a definition of terms for such a block composed of nonperiodic chains c_1, c_2,..., c_n, as an example.

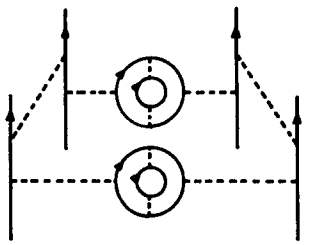

FIG. 14. An example of graph to be investigated further. The graph contains two e-sets, each of which contains two periodic blocks.

DEFINITION. The block $(c_1, c_2, \ldots, c_n)$ is *periodic* w.r.t. a given criterion if c_i, c_{i+p}, $c_{i+2p}, \ldots, c_{i+(n-p)}$, $i = 1, 2, \ldots, p$, $1 \leqslant p \leqslant n/2$, are equivalent to each other w.r.t. the criterion but c_j and c_k, $1 \leqslant j \neq k \leqslant p$, are not. The number p is called the *period* of the block. Let c_1 contain the base of this block, then *quasi-bases* of the block are bases of c_{p+1}, $c_{2p+1}, \ldots, c_{(n-p)+1}$. The *length* from c_i to c_j is the number $(j - i)$ modulo n. The *height* and the *quasi-height* of c_i in the block are number i and i modulo p, respectively. In this way a set of periodic blocks is defined (cf. Fig. 15).

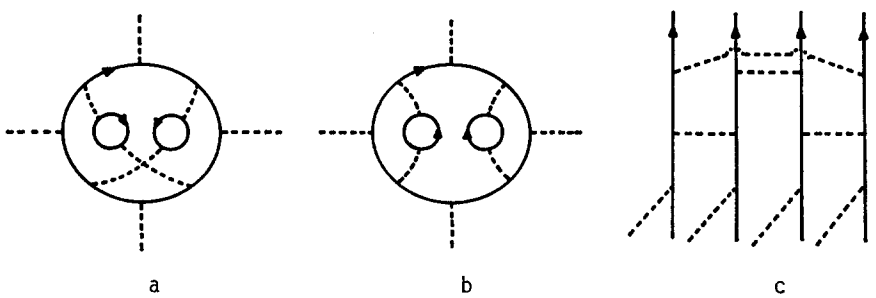

FIG. 15. Examples of periodic blocks. Blocks 15a and 15c are of period 2 but 15b is of period 4.

A graph is now characterized by the e-sets and the p-set defined for blocks. Accordingly, the e-sets and the p-set defined for chains are discarded. Then, the situation is quite similar to the case which we have encountered before with equivalent chains and periodic loops. This fact is an important result of forming blocks. Hence, applications are made, to the graph, of Tests 7 to 9, which are completely similar to Tests 2E and 2P, Tests 3E and 3P, and Tests 5.1 to 5.4, respectively. Here we only explain the necessity of Test 9. We may observe that some of the periodic blocks may be directly connected (see Fig. 14). If that is the case, we must again separate the p-set into "blocks" and apply to them a procedure similar to Test 5. The procedure corresponding to Test 6 is unnecessary because blocks in

each e-set are, by definition, not directly connected with each other. After Test 9, blocks in the p-set are not directly connected.

4.5. *Elimination Algorithm, Part III*

So far, we have examined all inner-block topologies and all inter-block topologies between any two blocks. All graphs to be further investigated may be classified as follows.

Type 1: graphs composed of only nonperiodic equivalent blocks (cf. Fig. 16a),

Type 2: graphs composed of a periodic block and nonperiodic equivalent blocks (cf. Fig. 16b),

Type 3: graphs composed of periodic blocks being connected with each other through nonperiodic equivalent blocks (cf. Fig. 16c).

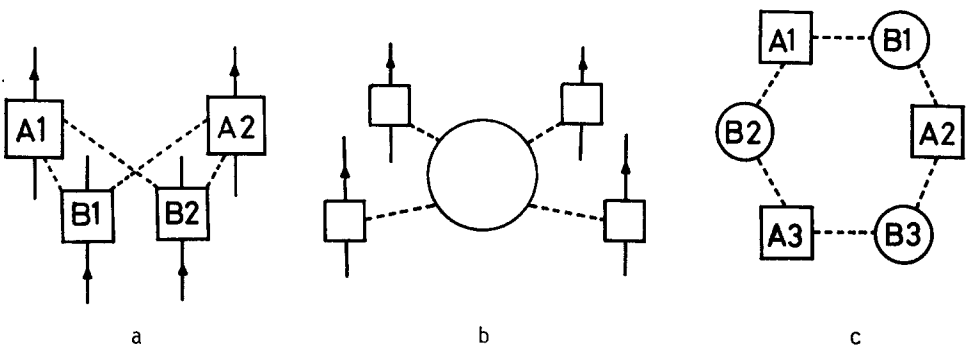

FIG. 16. Three examples of graphs to be tested by Test 10, where we denote equivalent blocks and periodic blocks by, respectively, squares and circles schematically.

We discuss each type of graph separately, and show that the ordering of equivalent blocks and the ranking of quasi-bases or bases, relative to the preferred order, can be completely determined. To make clear the following discussions, we use a term *b-block* to denote a block constructed from a set of blocks.

Graphs of the first type: Since every vertex is nonequivalent in each nonperiodic block, freedom of the vln configurations left for these graphs is restricted to that to which blocks equivalent vertices are connected. Suppose the jth e-set is $\{B_{j_1}, B_{j_2},..., B_{j_n}\}$, where the vln of the base of B_{j_i} is less than that of $B_{j_{i+1}}$. If the ph-neighbor of a vertex belongs to B_{j_i} we assign number i to this vertex. Then a graph can be completely characterized by an ordered family of ordered sets of numbers thus assigned to all of its vertices.

Test. 10.1. Exchange the vln's of all pairwise equivalent vertices in suitably chosen two equivalent blocks. Compare the graph thus constructed with the original one by using the sets of numbers described above. Repeat this block permutation procedure until the sets of numbers characterizing the former graph are found to be more "advanced" than those for the latter (then return "NO"), or until all possible such permutations are exhausted (then output the graph).
It is clear that Test 10.1 eliminates all of the mutually equivalent graphs but one.

Graphs of the second type: We group all nonperiodic blocks into b-blocks according to their connections. We examine how blocks are grouped into b-blocks and how they are connected within each b-block. The former examination is made by Test 6.1. The latter is made by Test 10.1, since each b-block is composed of nonperiodic equivalent blocks. If we find that some b-blocks are topologically different, we go back to Test 7 by redefining the e-sets. Otherwise we examine the connections of b-blocks to the periodic block. This can be performed as follows.

Test 10.2. Find all those vertices of a b-block such that their ph-neighbors belong to the periodic block. Let $\{v_1, v_2, ..., v_m\}$ be such vertices, where the vln of v_i is less than the vln of v_{i+1}. Let the ph-neighbor of v_i be w_i. For all v_i, obtain the length from w_1 to w_i. Then the connection of the b-block to the periodic block can now be completely characterized by an ordered set of numbers thus obtained. Using these sets of numbers, order the b-blocks. If the result is not consistent with the preferred order, then return "NO."

If we find that some connections are topologically different, we go back to Test 7 by redefining the e-sets. Otherwise the graph must be completely symmetric. Freedom of the vln configurations left for such topologically equivalent graphs is only that in which relative order equivalent b-blocks are connected to the periodic block. To remove this freedom, we fix the ranking of quasi-bases and the base of the periodic block (i.e., we set the p-set null). Then equivalent b-blocks can be ordered at once (Test 10.3).

Graphs of the third type: We group all nonperiodic blocks into b-blocks according to their connections, and examine how they are grouped into b-blocks, how they are connected within each b-block, and how b-blocks are connected with each periodic block, just as was described for the second type graphs. If we find during these procedures that mutually equivalent blocks belong, respectively, to two or more b-blocks which are topologically different, we go back to Test 7 by redefining the e-sets. Next, we classify the b-blocks into two classes: b-blocks in the first class connected with only one periodic block, and others in the second class. We construct periodic b-blocks from all periodic blocks and b-blocks of the first class according to their connections. Since each of these periodic b-blocks is completely symmetric, all equivalent b-blocks in it can be ordered by Test 10.3. After this procedure, we can discard the first class b-blocks. Then, the graphs

finally surviving are composed of periodic b-blocks being connected with each other directly or through nonperiodic equivalent b-blocks. The ranking, relative to the preferred order, of quasi-bases and bases of periodic b-blocks can be determined by a procedure similar to Test 5.2 (this is Test 10.4); in fact, if we identify each of the nonperiodic b-blocks as a point, the graphs become quite similar to the blocks composed of only periodic loops. After this procedure, we can discard the p-set, then we can order the nonperiodic equivalent b-blocks at once by Test 10.3. By comparing the resulting ordering with the preferred order, we can select only one graph out of the all topologically equivalent graphs.

This completes the proof that we can eliminate all unnecessarily duplicated

5. Concluding Remarks and Applications

Our program consists of two parts. The first constructs graphs, and the second eliminates unnecessary graphs. The algorithm for the former is fairly efficient (i.e., multiplicity of topologically identical graphs to be constructed in memory is fairly small), and is such that all generated graphs are well classified (i.e., graphs having the same skeleton and the same external ph-line connections are sequentially generated). The algorithm for the latter is devised so that we need not store the generated graphs to cope with the cases where a large number of graphs are involved. A weakness of our program is that it is applicable only to the QED interaction.

We mention here the applicability of our idea of the graph elimination to other types of interactions. Our elimination algorithm for QED Feynman graphs relies essentially on the fact that all skeletons to be constructed are topologically distinct from each other, and that all equivalent vertices can be completely classified (by equivalent chains and periodic loops). If it is the case for graphs describing other interactions, which seems to be very reasonable, then the graph elimination can be similarly performed as for QED graphs. However, we usually encounter many difficulties in general. That is:

(i) The concept of skeleton is not so simple in general.

(ii) If chains are not directed, we must consider the chain reversal symmetry.

(iii) If there are many kinds of lines, we must consider many kinds of chains and e-sets.

(iv) If more than three lines or three lines of the same kind are adjoined to one vertex, classification of subgraphs will surely be complicated. We will have to introduce not only the terms "equivalent" and "periodic," but also the terms that concern the permutation of vertices.

TABLE I

Results for Several Interesting Processes in Physics[a]

Graph type	N_{od}	N_{graph}	N_{subg}	T
Vertex graph	5	7	2	2″
$n_{el} = n_{ph} = n'_{el} = 1$	7	72	4	21″
	9	891	7	5′51″
	11	12672	12	1h50′
Vacuum graph	4	3	2	6″
$n_{el} = \cdots = n'_{ph} = 0$	6	8	3	24″
	8	39	5	1′38″
Photon self-energy graph	4	3	2	1.1″
$n_{ph} = n'_{ph} = 1$	6	18	2	3.7″
	8	153	4	34″
	10	1638	6	8′33″
Electron self-energy graph	4	3	2	0.7″
$n_{el} = n'_{el} = 1$	6	18	4	3.9″
	8	153	7	54″
Photon–photon scattering	4	2	1	4.6″
$n_{ph} = n'_{ph} = 2$	6	16	1	10.3″
	8	195	3	1′13″
Photon–electron scattering	4	8	1	2″
$n_{ph} = n_{el} = n'_{ph} = n'_{el} = 1$	6	74	3	15″
	8	888	6	3′17″
Electron–electron scattering	4	4	3	7″
$n_{el} = n'_{el} = 2$	6	28	7	20″
	8	303	14	3′19″
Electron–positron scattering	4	10	8	2.2″
$n_{el} = n_{ps} = n'_{el} = n'_{ps} = 1$	6	94	20	20.5″
	8	1136	41	5′46″
Four-electron scattering	6	4	3	10.1″
$n_{el} = n'_{el} = 4$	8	90	9	2′15″

[a] The four-electron scattering is supplemented because it is of interest from the graph elimination viewpoint, though it is seldom used in physics. N_{od}, N_{graph}, N_{subg}, and T are, respectively, the order of the corresponding graphs, the number of generated graphs = the number of all topologically different graphs, the number of all topologically different skeletons of the output graphs, and the total computer execution time excluding the initialization time.

Due to the reasons explained above, a generalization of our approach to more complicated interactions, for example, interactions in non-Abelian gauge field theories, will be very difficult. For such cases, however, we propose the following strategy. Find an efficient algorithm which constructs graphs in well-defined classes, then eliminate unnecessary graphs by applying the comparison method to each class of graphs. By specifying the type of interaction, we should be able to find such a graph construction algorithm, as we have done for the QED interaction. Then, the number of graphs to be stored in memory for comparison will be substantially reduced, because the graphs classified in the same set have to be stored at one time. In this way the memory requirement should be largely relaxed.

Table I shows some results of the graph generation for several interesting processes [3]. The computations were carried out on a FACOM 230-75 computer (word length = 36 bits, fixed point multiplication time = 0.45 μsec, and Gibson mix time = 0.207 μsec). The free storage used is 44K words, about half of which were occupied by the program itself. Our results may be compared with those of Perrottet [2] in our favor. Using a FORTRAN program, he generated graphs up to seventh order for several types of interactions. He employed the comparison method to eliminate the duplicated graphs. Considering that our LISP interpreter system [4] is written in FORTRAN, we may say our program is quite efficient.

ACKNOWLEDGMENTS

The author expresses his gratitude to Professor E. Goto for his interest in this work and useful comments, and to Professor H. Yamada for his careful reading and improving of the manuscript. Thanks are due to Messrs. M. Terashima and Y. Kanada of the Faculty of Science, University of Tokyo, for their help in handling a LISP system. He is indebted to Mr. T. Ida for his careful reading of the manuscript and helpful suggestions in correcting English. The communications with Professors B. E. Lautrup, A. C. Hearn, and T. Kinoshita are also acknowledged.

REFERENCES

1. M. VELTMAN, CERN preprint, 1967; A. C. HEARN, "REDUCE-II User's Manual," Univ. of Utah, Salt Lake City, 1973; M. J. LEVINE, *in* "Proceedings of the Third Colloquium on Advanced Computer Methods in Theoretical Physics," Marseille, France, 1973.
2. J. A. CAMPBELL AND A. C. HEARN, *J. Comp. Phys.* **5** (1970), 280; J. CALMET AND M. PERROTTET, *J. Comp. Phys.* **7** (1971), 191; M. PERROTTET, "Computing as a Language of Physics," p. 555, International Atomic Energy Agency, Vienna, 1972.
3. For ninth order vertex graphs, see also B. E. LAUTRUP, *Phys. Lett. B* **38** (1972), 408.
4. E. GOTO, "Monocopy and Associative Algorithms in an Extended LISP," Tech. Rept. 74-3, Information Sci. Lab., Univ. of Tokyo, Tokyo, 1974; M. TERASHIMA, "Algorithms Used in an Implementation of HLISP," Tech. Rept. 75-3, Information Sci. Lab., Univ. of Tokyo, Tokyo, 1975.