BA

stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 30/75 JUNE

JAC.M. ANTHONISSE

A GRAPH-DEFINING LANGUAGE

2nd (revised) edition

BA

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 05C99, 94A20

ACM -Computing Reviews- categories: 4.22, 5.32

1st printing 1973

2nd (revised) edition 1975

A graph-defining language

by

Jac.M. Anthonisse

ABSTRACT

A language for the definition of graphs and networks is introduced. In this language the vertices of a graph are identified by alphabetical or numerical identifiers. The vertices and arcs can be provided with numerical or non-numerical information. Subsets of vertices and arcs can be defined, these subsets are also identified by alphabetical or numerical identifiers. Moreover, subsets of vertex-informations and subsets of arc-informations can be defined which are also identified by alphabetical or numerical identifiers.

An implementation is available in the form of a set of Algol-60 procedures to read a graph and to store it for future analysis. These procedures are used in several programs.

KEY WORDS & PHRASES: *graph, network, data description.*

1. INTRODUCTION

In applications of graph theory, graphs and networks are used as models, and are analysed to derive conclusions about the phenomenon that is the proper subject of research. The graphs are generated from a set of data, analysed, and then the results of the analysis are interpreted in terms of the proper subject of research.

The analysis is defined as a program or procedure to be performed on the graph, in most cases the analysis will be performed by a computer. In order to make the analysis available for many applications the program or procedure should be defined independent of a specific application.

However, the graph itself and the results of the analysis should be available in a form which is very close to each particular application. In the present report a language for the definition of graphs and networks is introduced. In this language the vertices of a graph are identified by arbitrary alphabetical or numerical identifiers; thus, in each application, the 'natural' identifiers can be used. The vertices and arcs can be provided with numerical or non-numerical information. Numerical information consists of a sequence of real numbers, non-numerical information consists of a sequence of symbols. Subsets of vertices, arcs and informations can be defined, which are also identified by arbitrary alphabetical or numerical identifiers.

The language described in this report certainly is not the most general or the most powerful language for the definition of graphs and networks, that can be imagined. Many extensions and generalisations are possible. The present form, however, is sufficiently general and handy for a large number of applications. Moreover, the costs of the implementation are not excessive with respect to the programming effort, the storage requirements and the running time of the programs.

2. FUNDAMENTAL CONCEPTS

A graph is an (abstract) object, consisting of two types of elements, called *vertices* and *edges* respectively, where each edge is incident to one

or two vertices.

More formally a graph G can be defined as a quadruple G = (V,E,t,h) where V denotes a set of elements called *vertices*, E denotes a set of elements called *edges* and both t and h denote a mapping of E into V. Then edge $e \in E$ is incident to $t(e) \in V$ and $h(e) \in V$.

Only finite graphs, i.e. graphs with a finite number of elements, will be considered.

If an edge is incident to only one vertex, i.e. t(e) = h(e), that edge is called a *loop* on that vertex. If an edge is incident with two vertices these vertices are said to be adjacent to each other. Consequently, a vertex is not adjacent to itself. A vertex is isolated if it is not adjacent to any vertex.

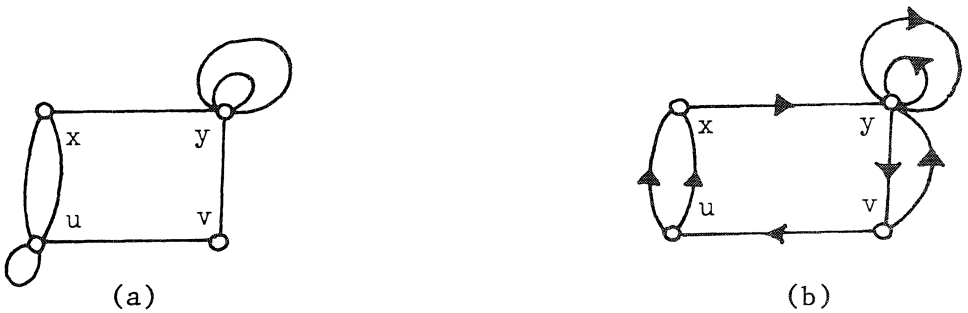
The term multiple edges (or multiple loops) is used to denote the situation that the incidences of two or more edges coincide, i.e. $e_1 \neq e_2$ and $\{t(e_1),h(e_1)\} = \{t(e_2),h(e_2)\}$. A graph which contains multiple edges is called a *multigraph*.

A *digraph* (or directed graph) is a graph in which an orientation (or direction) has been assigned to each edge.

An edge of a digraph, together with its orientation, is called an *arc*. It may be assumed, without loss of generality, that arc e is oriented from t(e) to h(e). Then t(e) and h(e) are called the tail and head of the arc respectively.

The term multiple arcs is used to denote the situation that the tails and heads of two or more arcs coincide, i.e. $e_1 \neq e_2$, $t(e_1) = t(e_2)$ and $h(e_1) = h(e_2)$.

Figure 1 gives a pictorial representation of a graph and a digraph. A pictorial representation, or any other representation, should not be confused with the graph itself.



(a) (b)

Figure 1 : pictorial representation of a graph and a digraph.

The graph (a) has multiple loops on vertex y, two edges are incident to both vertex x and vertex u.
The digraph (b) has multiple loops on vertex y, and has multiple arcs between x and u; the arcs between y and v are not multiple ones as they are oriented in opposite directions.
Both graph and digraph have 12 elements, consisting of 4 vertices and 8 edges.

In the discussion of digraphs it is sometimes necessary to ignore the orientation of arcs. In such cases the term edge will be used to stress the fact that not the digraph but the 'underlying' graph is considered.

Graphs in which both oriented and non-oriented edges occur will not be considered.

3. NETWORKS

In many applications a graph or digraph as defined above is only part of the model. In order to obtain a realistic model the elements of the graph are often provided with information describing characteristic properties of the objects corresponding to the elements of the graph. Thus the existence of a relation between two objects could be described by the existence of an edge between two vertices whereas the contents of the relation could be described by the information associated with that edge.

In fact, the orientation of an arc should be interpreted as information associated with an edge. But the many applications of digraphs and the fundamental concepts which apply to digraphs only lead to the separation of orientation from the other types of information and to the introduction and study of digraphs as a specific type of graph.

A graph, together with the information associated with its elements, is called a *network*. As the orientation of edges is not interpreted as information, directed and non-directed networks can be distinguished.

The type and amount of information associated with vertices and edges (or arcs) depends upon the specific application. In many cases the information consists of one or more numerical values, but non-numerical information occurs in several applications.

Figure 2 gives a pictorial representation of two networks, they are equivalent to the multigraphs of figure 1, if the information associated with each edge (or arc) is interpreted as the multiplicity of that element.

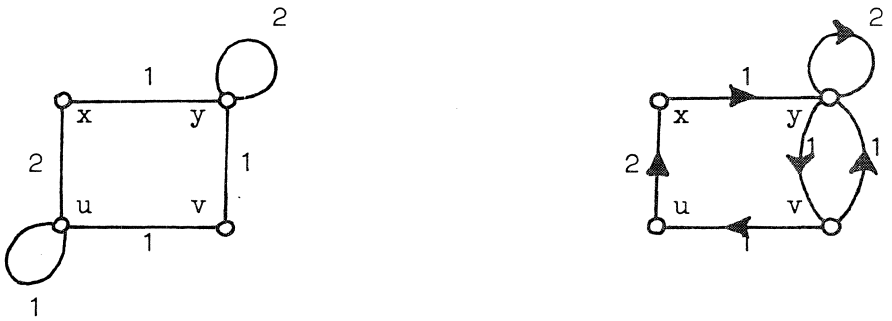


Figure 2 : pictorial representation of two networks.

4. DEFINITIONS OF GRAPHS

Before a graph can be analysed it must be defined. The definition consists of the identification of the vertices, the identification of the edges (or arcs) and the definition of the incidence relations between vertices and edges. The most extensive definition would consist of a list of identifiers denoting the vertices, a list of identifiers denoting the edges and a list of incidence relations (i.e. a list of pairs, each pair consisting of a vertex-identifier and an edge-identifier).

If it is not necessary to have specific identifiers for the edges then each edge can be identified by two vertex-identifiers, i.e. the identifiers of the vertices the edge is incident to. The definition of a graph then consists of a list of vertex-identifiers and a list of pairs of vertex-identifiers, each pair corresponding to an edge. If a pair consists of two identical identifiers the edge is a loop. If the graph is a directed one the first identifier of a pair is the tail of the arc, the second one is the head. The list of vertex-identifiers could be deleted if the graph is without isolated vertices, in such cases the list can be derived from the pairs. In practice, however, it is convenient to have such a list because it is a user-defined ordering of the vertices, which can be applied in the presentation of results of computations.

Another practical requirement is that of defining the set of vertices as the union of several subsets of vertices. In such cases the list of vertex-identifiers is replaced by several lists of vertex-identifiers, and each list is provided with an identifier denoting that list. It should also be possible to replace the list of edges by several lists of edges, where each list is provided with an identifier denoting that list. This option permits the assignment of a specific identifier to each edge, by defining a list for each single edge.

In practice it is often convenient to define several lists of vertex-informations and several lists of edge- (or arc) informations, and to provide each list with an identifier denoting that list.

5. INFORMAL DESCRIPTION

This section is an introduction to the formal definition of the graph-defining language.

In combination with the section on implementation this section could serve as a manual for the preparation of input for a program that reads a graph.

Throughout this and the next sections, the terms arc, head and tail will be used, even if the graph is a non-directed one.

From the above it will be clear that identifiers are essential in the definition of graphs and networks. As mentioned above, two types of identifiers are distinguished.

A *numerical identifier* is a signed or unsigned integer-valued number.

An *alphabetical identifier* is a sequence of symbols. The available symbols are:

the letters,
the digits,
blank,
the (decimal) point,
the lower ten ($_{10}$),
the plus (+),
the minus (-).

By inclusion of the point, lower ten, plus and minus it becomes possible to interprete numerical identifiers as alphabetical identifiers if necessary.

Blanks can not occur as the first or last symbols of an alphabetical identifier.

As mentioned above two types of information are distinguished.

Alphabetical information consists of a sequence of symbols, the available symbols are the same as for alphabetical identifiers.

Numerical information consists of a sequence of real numbers. Each number is required to be preceded by its sign to separate it from the preceding number. Numerical information can be interpreted as alphabetical information if necessary.

The definition of a graph or network can consist of six parts:

the title,
the lists of vertex-informations,
the lists of arc-informations,
the lists of vertices,
the lists of arcs,
the end.

The title consists of:

graph,
the proper title,
a semicolon,

e.g.

graph xy05 : relations of type 5 between set x and set y;

The proper title should not contain a semicolon.

Each list of vertex-informations consists of:

vinfos,
the list-identifier,
an equal,

the proper list,
a semicolon.

The proper list consists of vertex-informations, subsequent informations are separated by a comma,
e.g.

vinfos type = a,ab,z;

Each list of arc-informations consists of:

ainfos (or einfos)
the list-identifier,
an equal,
the proper list,
a semicolon,

e.g.

ainfos 300 = 301,333,317;

Each list of vertices consists of:

vertices,
the list-identifier,
an equal,
the proper list,
a semicolon.

The proper list consists of vertex-identifiers, subsequent identifiers are separated by a comma,
e.g.

vertices set x = x1, x2, x3, x4;

Information associated with a vertex should be given between the parenthesis (and) immediately after the identifier of that vertex,
e.g.

vertices set x = x1(12), x2(17), x3(9), x4(5);

Each list of arcs consists of:

arcs (or edges)

the list-identifier,
an equal,
the proper list,
a semicolon.

In the straightforward form the proper list consists of single arcs, subsequent single arcs are separated by a comma.

A single arc consists of:

the tail,
a colon,
the head,

e.g.

```
arcs  rel5 = x1 : y2, x2 : y4, x2 : y2, x1 : y3, x3 : y3 ;
```

Both tail and head are vertex-identifiers.

A vertex-identifier can occur several times as the tail, of several arcs, in the straightforward form of the proper list. In such cases the first abbreviated form can be used:

```
arcs  rel5 = x1 : (y2,y3), x2 : (y4,y2), x3 : y3 ;
```

A vertex-identifier can occur several times as the head, of several arcs, in the straightforward form of the proper list. In such cases the second abbreviated form can be used:

```
arcs  rel5 = (x1,x2) : y2, x2 : y4, (x1,x3) : y3 ;
```

Information associated with an arc should be given between the parentheses (and) immediately after the tail or head of that arc.
If both tail and head are followed by information the second information supersedes the first.
E.g.

```
arcs  rel5 = x1 : y2 (3), x2 : y4 (5), x2 : y2 (7),
             x1 : y3 (11), x3 : y3 (1) ;

arcs  rel5 = x1 (3) : y2, x2 : y4 (5), x2 (7) : y2 (7),
             x1 (3) : y3 (11), x3 : y3 (1) ;
```

```
arcs  rel5 = x1 : (y2(3),y3(11)),
             x2 : (y4(5),y2(7)),
             x3 : (y3(1)) ;

arcs  rel5 = x1 (3) : (y2,y3(11)),
             x2 : (y4(5),y2(7)),
             x3 (1) : y3 ;

arcs  rel5 = (x1(3),x2(7)) : y2,
             (x2(5)) : y4,
             (x1(11),x3(1)) : y3 ;
```

The end of the graph definition consists of:

fini

The definition of a graph or network may be interspersed with comments, each comment consists of:

a quote,
the proper comment,
a quote,

e.g.

"This list contains 7 elements"

The proper comment should not contain a quote.

If the definition of a graph contains two or more lists of vertices then their list-identifiers are required to be unique, i.e. a list-identifier identifies a single list. A vertex, however, may occur in several lists of vertices, i.e. the lists are not required to be disjoint.

A list of vertices may be defined as an empty one:

e.g.

```
vertices nodes = ;
```

The list of arcs may contain vertices which do not occur in a list of vertices and the arcs incident to them are ignored, unless the last list of vertices was defined as an empty one. In the latter case such new vertices are added to the last list of vertices.

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

Similar remarks hold for the vertex-informations, arc-informations and arcs.

6. FORMAL DEFINITION

This section contains the formal definition of the syntax of the graph defining language. Neither examples nor semantics are included, these are found in the preceding and next sections respectively.

<digit> :: = 0|1|2|3|4|5|6|7|8|9
<unsigned integer> :: = <digit>|<unsigned integer><digit>
<signed integer> :: = +<unsigned integer>|-<unsigned integer>
<integer> :: = <signed integer>|<unsigned integer>
<decimal fraction> :: = .<unsigned integer>
<exponent part> :: = $_{10}$<integer>
<decimal number> :: = <unsigned integer>|<decimal fraction>|
<unsigned integer><decimal fraction>
<unsigned numbers> :: = <decimal number>|<exponent part>|
<decimal number><exponent part>
<signed number> :: = +<unsigned number>|-<unsigned number>
<number> :: = <signed number>|<unsigned number>
<letter> :: = a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<numerical symbol> :: = .|$_{10}$|+|-
<id symbol> :: = <letter>|<digit>|<numerical symbol>|<blank>
<numerical identifier> :: = <integer>
<alphabetical identifier> :: = <id symbol>|
<alphabetical identifier><id symbol>
<identifier> :: = <numerical identifier>|<alphabetical identifier>
<vinfos list identifier> :: = <identifier>
<ainfos list identifier> :: = <identifier>
<vertex list identifier> :: = <identifier>
<arc list identifier> :: = <identifier>
<vertex identifier> :: = <identifier>

```
<numerical information> :: = <number>
                  <numerical information><signed number>
<alphabetical information> :: = <id symbol>|
                  <alphabetical information><id symbol>
<information> :: = <numerical information>|<alphabetical information>
<vertex information> :: = <information>
<vinfo list> :: = <empty>|
                  <vertex information>|<vinfo list>,<vertex information>
<single list of vertex informations> :: =
     vinfos <vinfos list identifier> = <vinfo list> ;
<lists of vertex informations> :: = <empty>|
    <single list of vertex informations>|
                  <lists of vertex informations><single lists of vertex
                                                        informations>
<arc information> :: = <information>
<ainfo list> :: = <empty>|
    <arc information>|<ainfo list>,<arc information>
<single list of arc informations> :: =
     ainfos <ainfos list identifier>=<ainfo list>;|
     einfos <ainfos list identifier>=<ainfo list>;
<lists of arc informations> :: = <empty>|
    <single list of arc informations>|
                  <lists of arc informations><single list of arc
                                                        informations>
<single vertex> :: =
    <vertex identifier>|<vertex identifier>(<vertex information>)
<vertex list> :: = <empty>|
    <single vertex>|<vertex list>,<single vertex>
<single list of vertices> :: =
     vertices <vertex list identifier>=<vertex list>;
<lists of vertices> :: = <empty>|
    <single list of vertices>|<list of vertices><single list of vertices>
<tail> :: =
    <vertex identifier>|<vertex identifier>(<arc information>)
<head> :: =
    <vertex identifier>|<vertex identifier>(<arc information>)
```

```
<list of tails> :: =
    <tail>|<list of tails>,<tail>
<list of heads> :: =
    <head>|<list of heads>,<head>
<primary arc list> :: =
    <tail>:<head>|
    <tail>:(<list of heads>)|
    (<list of tails>):<head>
<arc list> :: = <empty>|
    <primary arc list>|<arc list>,<primary arc list>
<single list of arcs> :: =
     edges <arc list identifier>=<arc list>;|
     arcs  <arc list identifier>=<arc list>;
<lists of arcs> :: =<empty>|
    <single list of arcs>|<list of arcs><single list of arcs>
<other symbol> :: = ,|;|(|)|[|]|=
<symbol>:: = <id symbol>|<other symbol>
<proper comment> :: = <symbol>|
    <proper comment><symbol>
<title> :: = graph <identifier>;
<end>   :: = fini
<comment> :: = "<proper comment>"
<graph> :: = <title>
             <lists of vertex informations>
             <lists of arc informations>
             <lists of vertices>
             <lists of arcs>
             <end>
```

7. IMPLEMENTATION

The graph-defining language, as described and defined in the preceding sections, has been implemented in ALGOL-60, as available on a Control Data CYBER 73 computer.

As underlining is not available, the symbols

graph, vinfos, ainfos, einfos, vertices, arcs, edges, fini

are punched as

'graph', 'vinfos', 'ainfos', 'einfos', 'vertices', 'arcs',
'edges', 'fini',

respectively.

Instead of the lower ten $(_{10})$ the quote (") is punched; instead of the quote (") two subsequent apostrophes ('') are punched.

The term *entity* will be employed to denote an identifier or an information. Four classes of entities are distinguished:
the vertex-informations and the identifiers denoting lists of vertex-informations,
the arc-informations and the identifiers denoting lists of arc-informations,
the vertex-identifiers and the identifiers denoting lists of vertices,
the identifiers denoting lists of arcs.

The length of an alphabetical identifier or information is equal to the number of symbols it contains, e.g. aba has length 3, whereas +1+2+1 has length 6.

The length of a numerical information is equal to the number of elements it contains, e.g. +1+2+1 has length 3.

The length of a numerical identifier is 1.

Before a graph can be read a number of parameters are to be specified. These parameters describe, for each class, the type and number of entities in that class, and are used to reserve sufficient storage for the graph.

The parameters are specified with the help of a number of keywords. If a parameter is used to describe the number of entities in a class then its keyword is followed by the symbol = and an integer, e.g. : ivm = 15. The value assigned to the parameter is an upper bound for the number of entities.

If a parameter describes the type of a class of entities then several keywords are available and the right keyword must be selected to define the

type.

If a parameter is not specified explicitly then a default value is selected automatically.

The list of keywords is preceded by

'parameters'

subsequent keywords are separated by a comma, a semicolon closes the list.

The keywords concerning the vertex-informations are:

ivm, ivnum, ivalph, ivin, ivnin, ivfl, ivr, ivl, ivlm, ivlfl, ivll.

ivm

is an upperbound for the number of vertex-information. The specifications ivm = 15 means that the definition of the graph contains at most 15 different vertex-informations.

Default: ivm = 0, with the effect that all vertex-informations, if present, are ignored.

ivnum, ivalph

The specification ivnum means that the vertex-informations and their list-identifiers are of type numeric. The specification ivalph means that they are of type alphabetic.

Default: ivnum.

ivin, ivnin

The specification ivin means that blanks occurring within vertex-informations or their list-identifiers should be considered as symbols, with the result that a a and aa will be considered as different informations or identifiers. The specification ivnin means that blanks occurring within these entities should be skipped, with the result that a a will be recognized as the entity aa.

Default: ivnin.

ivfl

is an upperbound for the sum of the lengths of the different vertex-informations, plus the length of the longest vertex-information.

Default: ivfl = ivm + ivm + 2 if ivalph, ivm + 1 if ivnum.

ivr,ivl

These parameters are used for the presentation of results of computations. The specification ivr means that the entities will be right-justified in an appropriate field, the specification ivl means that they will be left-justified. If there are three entities aaa, aa, a then they will be printed as

aaa, aa, a in case of ivr, or as

aaa,aa , a in case of ivl.

Default: ivl.

ivlm

is an upperbound for the number of lists of vertex-informations.

Default: ivlm = 0, with the effect that the list-identifiers are ignored. The informations are not ignored, but stored into a single list with 1 as its identifier.

ivlfl

is an upperbound for the sum of the lengths of the list-identifiers, plus the length of the longest one.

$$\text{Default : ivlfl} = \begin{cases} \text{ivlm + 1} & \text{if ivnum,} \\ \text{ivlm + ivlm + 2} & \text{if ivalph.} \end{cases}$$

ivll

is an upperbound for the sum of the sizes of the lists of vertex-informations.

Default: ivll = ivm.

The keywords concerning the arc-informations are:

iam, ianum, iaalph, iain, ianin, iafl, iar, ial, ialm, ialfl, iall.

Their interpretation is completely analoguous to the interpretation of the vertex-informations-keywords.

The keywords concerning the vertex-identifiers are:

vm, vnum, valph, vin, vnin, vfl, vr, vl, vlm, vlfl, vll.

Their interpretation is completely analoguous to the interpretation of the vertex-infomations-keywords.

The keywords concerning the list-identifiers of the lists of arcs are:

alm, alnum, alalph, alin, alnin, alfl, alr, all.

alm

is an upperbound for the number of list-identifiers.

Default: alm = 0, with the effect that the list-identifiers are ignored, and that the arcs are stored into a single list with 1 as its identifier.

alnum, alalph

The list-identifiers are of type numeric or alphabetic respectively.

Default: alnum.

alin, alnin

Blanks within these list-identifiers are considered as symbols or are ignored respectively.

Default: alnin.

alfl

is an upperbound for the sum of the lengths of the list-identifiers, plus the length of the longest one.

Default: $alfl = \begin{cases} alm + 1 & \text{if alnum,} \\ alm + alm + 2 & \text{if alalph.} \end{cases}$

alr, all

The list-identifiers will be right-justified or left-justified respectively.

Default: all.

The options of coding will be described in the next section, the keywords concerning the use of these options are:

codedvinf, vinf

In the lists of vertices the vertex-informations are coded or not coded respectively.

Default: vinf.

codedainf, ainf

In the lists of arcs the arc-informations are coded or not coded

respectively.
Default: ainf.

codedvert, vert
In the lists of arcs the vertex-identifiers are coded or not coded respectively.
Default: vert.

The remaining keywords are:

am, tl, directed, nondirected, test, real.

am
is an upperbound for the number of arcs in the graph.
Default: am = 0, with the effect that all arcs are ignored.

tl
is an upperbound for the number of symbols constituting the proper title of the graph.
Default: tl = 80.

directed, nondirected
The graph is a directed one or nondirected one respectively.
Default: nondirected.

test, real
Probably unimportant errors in the definitions of the graph are counted or ignored respectively.
Default: real.

8. CODING

In the course of reading the lists of vertex-informations a label is assigned to each information. The label is a natural number, the first information obtains label 1, the second information obtains label 2, etc., the label of an information that corresponds to the order in which the informations are encountered.

If, in the course of reading the lists of vertices, a vertex-informa-

tion is encountered, the label of that information is determined and associated with the vertex-identifier. The information associated with a vertex is then available by means of the label associated with the vertex-identifier, the label serves as a pointer to the information.

The specification codedvinf in the set of parameters means that the option of coding the vertex-informations has been used. In this case the lists of vertices do not contain the proper vertex-informations, but their labels. Thus

```
'vinfos' zz  : xyz, pqrs, a, b;
'vertices' x : x1(xyz), x2(xyz), x3(pqrs), x4(b);
```

is equivalent to

```
'vinfos' zz  : xyz, pqrs, a, b;
'vertices' x : x1(1), x2(1), x3(2), x4(4);
```

provided the list of vertex-informations zz is not preceded by any other non-empty list of vertex-informations.

The specification codedainf means that the arc-informations are coded, i.e. the lists of arcs do not contain the proper informations, but their labels. The specification codedvert indicates that the vertex-identifiers are coded, i.e. the lists of arcs do not contain the vertex-identifiers, but their labels.

Both cases are handled similar to the case of coded vertex-informations.

9. EXAMPLE

```
'parameters'
 ivm = 5, ivlm = 1,
  vm = 6, valph, vlm = 4, vlfl = 15, vll = 12,
 alm = 3, am = 15;
'graph' example;
'vinfos' 0 = ;
'vertices' all = a, b, c, d, e, f;
'vertices' odd = a(1), c(2), e(3);
```

```
'vertices' even = b(1), d(2), f(3);
'arcs' 1 = b : a;
'arcs' 12 = a : (c,d), b : (c,d);
'arcs' 123 = (a,c,e) : f, b : (a,c,e);
'fini'
```