Relation Nets and Hypernets

by

Hendrik O. van Rooyen, University of South Africa
Franz Stetter, Universität Mannheim
Aletta E. Geldenhuys, Boston Business College

June, 2001

Table of contents

Acknowledgement: The authors wish to thank J D Brear of Progen in Cape Town, South Africa for moral and financial support during the course of this work. Thanks also to Barbara Stetter and Petro van Rooyen for their patience and courage while accomplishing the often trying task of typing and correcting the manuscript. The authors take full responsibility for any remaining errors.

Prologue

In many respects this report is a companion work of [GVS99]. In some senses it runs parallel to [GVS99], while in others it is a sequel to that book. Readers not familiar with [GVS99] will find themselves refering back to it in several instances to follow some of the subtleties of this work, as these are often bound with aspects of [GVS99], particularly in the case of concept-relationship knowledge stuctures, abbreviated CRKS in what follows; they are not explicitly repeated here.

Some small errors in [GVS99] are corrected in this report and certain additions to the theory of CRKS's are dealt with in a way that covers both CRKS's and their hypernet equivalent. The main application of CRKS's - namely modelling study material - is not explicitly transcribed to this paper, but that whole notion is abstracted and made independent of any specific teaching/learning metalanguage through the implications of this abstraction.

Two key factors emerge from this paper on hypernets. First, unlike the case for CRKS's in which little of the general theory of relation nets - see Part III of [GVS99] - applies to CRKS's, the broad theory of hypernets, as far as it is covered in this report, is often applicable to the hypernet equivalent of a CRKS. Second, we will show a link between relation net isomorphism and hypernet isomorphism which makes it considerably easier to deal with CRKS isomorphism and, thus, with structural analogy as used in a modelling based approach to teaching/learning/analogical reasoning [GVS99].

Finally, we must mention that it appears that the domain of potential practical applications of hypernets must inevitably be wider than that for relation nets. In this connection, it should be noted, however, that this report is written with applications in the field of education in mind, specifically in the realm of the modelling of study material, the planned representation of that material, problem representation and solution, analogical reasoning, and to assist in curriculum planning and student registration, particularly in modelling small course unit systems with relatively complex registration conditions. Such applications in education will not be made explicit here, but are implied by the work in Parts I and II of [GVS99] and in this extension of it.

1. Introduction

In [GVS99] we developed the theory of relation nets. The main application was to the representation of study material in terms of a model called a concept-relationship knowledge structure, abbreviated CRKS, that is a special case of a relation net. Part I of [GVS99] described the theory and an application of CRKS's in some detail, Part II was dedicated to a special example of a CRKS, and Part III laid out the mathematical fundamentals of a theory of relation nets proper.

Early work on the system that was to become known as a relation net introduced a relation net representation of a specific curriculum that consisted of a number of interrelated "small course units", known in that case as modules (see [VR76] and [Wei83] for example). In this paper, we will be bearing in mind two similar systems upon which that part of hypernet theory introduced is founded, in the sense that we will introduce no theory that does not have potential application to this kind of system. We start by introducing these application systems in abstract form.

First we present an description of a curriculum system in abstract form. Imagine, for example, a "small course unit" curriculum that leads to degrees and diplomas. By a ***course unit*** we mean any complete and interrelated section of study material. By a ***prerequisite*** unit for a given course unit U we mean a course unit C, or a condition C, that must be completed or fulfilled before course unit U can be entered. By a ***parallel*** unit for a course unit U we will mean a course unit P that must be completed before, or simultaneously with, course unit U as a requirement for obtaining credit for U. We may extend this by adding another form of parallel for U, namely a course unit that may be entered at the same time as entering U, but is not a necessary precondition for obtaining credit for completion of U.

We visualize such a curriculum system in the form of a labelled graph as follows: Plot a vertex for each course unit in the curriculum, and label each vertex with the unique (code) name of the relevant course unit. Each course unit U has at least one non-empty list of prerequisites, and at least one list of parallels which may be empty. These prerequisite and parallel units constitute a ***condition set*** for U, and U may have more than one condition set, depending on the particular degree or diploma in which U is registered. In each condition set we mark all the prerequisite units, for example with an underbar, and also mark all of the parallel units of the first kind, for example with an overbar. We number each occurence of a condition set uniquely, and notice that distinct condition sets need not be disjoint. From each prerequisite in each condition set for U we draw an arc to U, and we label that arc with that condition set and its number. We do this for all the condition sets for U, and repeat this for all the course units in the curriculum. Such a labelled graph can be read hierarchically from prerequisites to dependants, or vice versa, i.e. from bottom-to-top or from top-to-bottom. As we will see, such a graph is an example of a hypernet.

Such a curriculum system for a host of "small" course units has pro's and con's. It's major advantages are to allow more flexibility of topic choice and degree/diploma structure, easier changes of "direction" of study, and an ability to support multi-disciplinary studies. The major disadvantage is the complexity of registration and administration.

We will see that, in combination with [GVS99], hypernet representation will enable registration, administration, planning, alteration, and analysis of the whole structure or parts thereof by means of formal theory and strong but relatively simple computer support. In the relation net approach to curriculum systems of this nature, an order was forced on the members of the condition sets, which was a handicap in the representation. We will see that the hypernet model is more "natural" in this case.

A similar situation arises in [GVS99] when we introduce the notion of an ***action diagram*** in the course of a discussion of problem formulation and solution by top-down algorithm (see section 8.5 of [GVS99]). Here we leave out the directed arrows in the action diagram and the arbitrary ordering of nodes on the arrow labels in the resulting relation net, producing instead a hypernet associated with the action diagram. Consider, for instance, the diagram on p.139 of [GVS99]; using arcs in place of arrows, we get the following version of that action diagram:

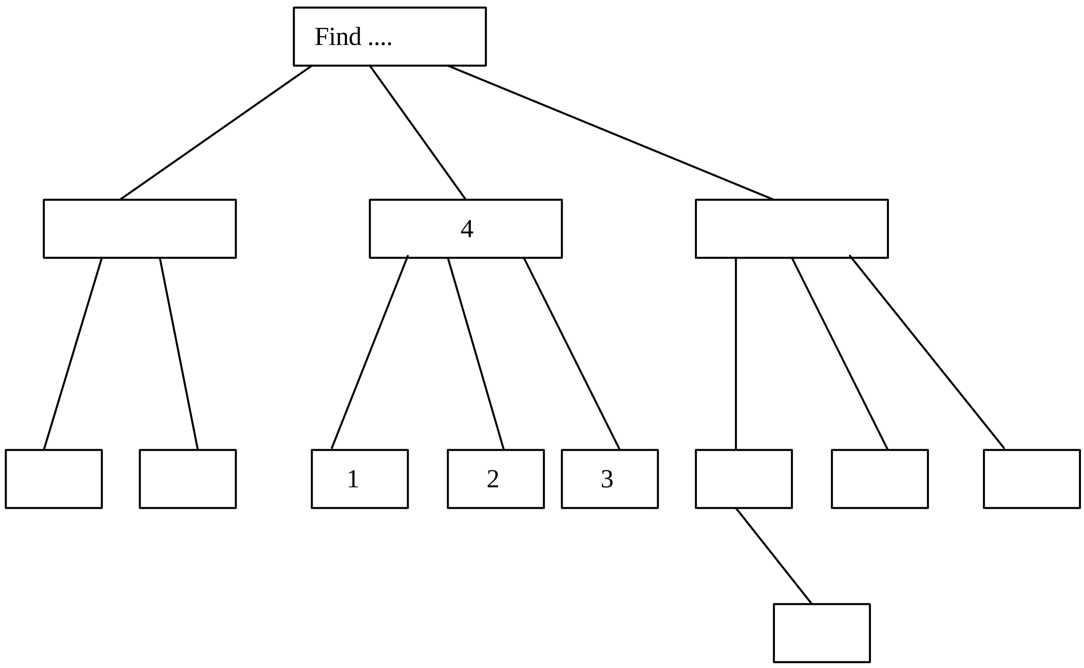


Figure 1.1: An example of a partial action diagram

Part of the resulting hypernet is:

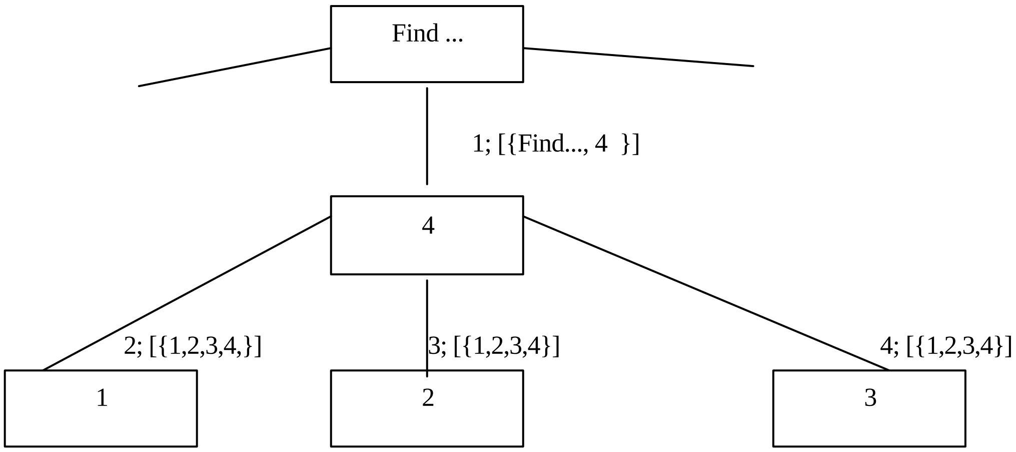


Figure 1.2: A partial hypernet for figure 1.1

In this case there is one "condition set" in each label, and the set of vertices {1,2,3,4} generates three edges, numbered 2;, 3;, and 4;.

There is a connection between our curriculum example and this one. Reading top-to-bottom we see that "Find ..." is a prerequisite of 4, with no parallels, and 4 is a prerequisite of 1, for example, with parallels 2 and 3. Reading bottom-to-top, we must be a bit careful. In this case, 1 is a prerequisite of 4 with 2 and 3 as other prerequisites of 4, and with no parallels , and 4 is a prerequisite of "Find ..." with no other prerequisites and no parallels. It is the intended interpretation which, in each individual case, will determine whether we read such hypernets from top-to-bottom or from bottom-to-top. For the hypernets that arise from action diagrams, top-to-bottom is interpreted as the specification of the top-down algorithm for the solution of the problem(s) and bottom-to-top as the actual solution procedure for the relevant problem(s).

On page 141 of [GVS99] we meet a more general action diagram situation. The hypernet that arises from the section of an action diagram shown there is:

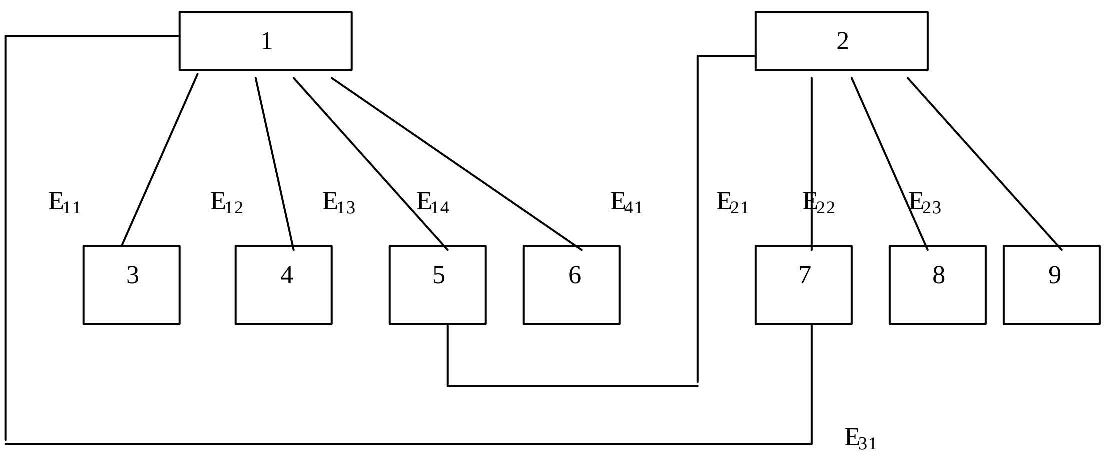


Figure 1.3: A hypernet from the partial action diagram on page 141 of [GVS99]

The first index characterises the set of vertices; the second the edge with that set. Here $E_1=\{1,3,4,5,6\}$, $E_2=\{2,7,8,9\}$, $E_3=\{1,7\}$, $E_4=\{5,2\}$.
Reading top-to-bottom, we have for example:

- In E_{11}, 1 is a prerequisite of 3 with 4, 5 and 6 as parallels.
- In E_{22}, 2 is a prerequisite of 8 with 7 and 9 as parallels.
- In E_{31}, 1 is a prerequisite of 7 with no parallels.

Reading bottom-to-top, these labels mean:

- In $E_{11,}$ 3 is a prerequisite of 1, as are 4, 5 and 6.
- In $E_{22,}$ 8 is a prerequisite of 2, as are 7 and 9.
- In $E_{31,}$ 7 is a prerequisite of 1 with no parallels, and in E_{21}, 7 is a prerequisite of 2 as are 8 and 9.

Such hypernets can, as we will see, easily and formally be compared for common, i.e. structurally analogous, substructures using hypernet isomorphism. This is a potentially extremely useful technique in the development of general problem formulation and solution skills. We note in passing that the same kind of hypernet can be used to display and analyse the relationships between the subroutines that combine to form a program. We will also see that there are some measures of the complexity of certain hypernets that can play a very significant role in the analysis of such hypernets.

2. Relation nets, hypergraphs, and hypernets

Relation nets have been introduced and fairly extensively covered in [GVS99]. The notation used in Part III thereof is detailed and therefore quite complex, but in Part I (the theory of CRKS's) a tuple table notation is used that is much more "user friendly". We begin this section by changing the notation for general relation nets also to the tuple table approach, and then go on to some basic definitions of a theory of hypernets, defining the notion of hypernet in the process.

Definition 2.1: Consider a finite set

$$A = \{A_1, A_2, ... , A_n \}$$

and a family of relations

$$R = \{R_i \mid i \in I, \text{I a finite index set}\}$$

over A where all R_i have an arity of at least 2, i.e. card(R_i) ≥ 2, written | R_i | ≥ 2. We denote such a system by <A, R, I>. By a ***relation net*** representation of <A, R, I> we mean a pair <A, T> where T is the set of all tuples from all of the R_i . ♦

Note that some of the R_i may be identical. Each tuple in T is given a unique code name, generally of the form "i; x" where i indicates the R_i of origin of that tuple and x is usually the number of the tuple in T. We will use only the unique tuple number x if we do not need to take account of the particular R_i from which the relevant tuple arises. In that case we will regard T as a single finite family of tuples T = { T_x }.

Definition 2.2: By a ***diagram*** of a relation net <A,T> we mean a representation drawn as follows. Plot precisely one vertex for each member of A and label each such vertex with the "name" of the appropriate member of A. Next, for each $T_k \in$ T with $T_k = \langle a_o, ..., a_j \rangle$, where j is the arity of the relation R_i from which T_k arises, we draw an arrow from the a_o vertex to the a_j vertex. Now label each such arrow <a,b> with a ***label*** λ(<a,b>) where λ(<a,b>) is defined by $\lambda(\langle a,b \rangle) = \{T_k \in T \mid T_k = \langle a, ..., b \rangle\}$. There is no arrow from a ∈ A to b ∈ A iff λ(<a,b>) = ∅.♦

The notion of a hypernet was inspired by that of a hypergraph [Ber73] and a desire to ignore at least part of the ordering implied by the arrows and paths of a relation net, without moving too far from either hypergraphs or relation nets.

Definition 2.3: By a ***hypernet*** <A, E> we mean a structure in which $A= \{A_1, A_2, ... , A_n \}$ is a finite set and $E = \{ E_i \mid i \in I\}$ is a family of non-empty subsets of A. |A |is called the ***order*** of <A, E> and I the ***index set*** of <A, E>. Each $A_i \in$ A is called a ***vertex*** of <A, E>, and each $E_i \in$ E is called an ***edge*** of <A, E>. Two edges E_i and E_j of <A, E> are distinct iff i ≠ j, even though E_i and E_j may be the same set. ♦

Definition 2.4: Two vertices A_i , $A_j \in$ A of a hypernet <A, E> are said to be ***potentially vertex adjacent*** by edge E_i iff {A_i , A_j }is a subset of E_i. Two edges E_i, $E_j \in$ E are said to be ***potentially edge adjacent*** iff $E_i \cap E_j \neq \emptyset$, and for every $A_k \in$ A with $A_k \in E_i \cap E_j$ we say that E_i is ***potentially edge adjacent*** with E_j by A_k. ♦

Now consider three distinct edges E_i, E_j, $E_k \in E$ with $E_i \cap E_j \neq \emptyset$ and $E_k \cap E_j \neq \emptyset$. Then we say that each $A_r \in E_i \cap E_j$ is potentially vertex adjacent with each $A_s \in E_k \cap E_j$ by E_j. We write (A_r, E_j, A_s) for every pair $\{A_r, A_s\}$ of vertices with $A_r \in E_i \cap E_j$ and $A_s \in E_k \cap E_j$ if A_r and A_s are vertex adjacent by E_j in <A, E>. If $E_i = \{ A_r \}$ for some $A_r \in A$ and some $E_i \in E$ then we call E_i a ***singleton edge***. A singleton edge at $A_r \in A$ is also called a ***loop edge*** at A_r.

Note well that a hypernet need not have in it all the potential vertex adjacencies, nor need it have all the potential edge adjacencies; in each case it may have all, or some, or none of the potential adjacencies.

Definition 2.5: Given a hypernet <A, E>, [if the edges $E_i \in E$ are all non-empty distinct subsets of A and] if $\cup_i \; E_i = A$, and if two edges E_k, E_l are adjacent iff $E_k \cap E_l \neq \emptyset$, then <A, E> is a [***simple***] ***hypergraph.*** ♦

We will ignore the standard diagrammatic representation of hypergraphs [Ber73] and draw hypergraph diagrams as we do hypernet diagrams. The class of hypergraphs can be regarded as a subclass of the class of hypernets.

Definition 2.6: Given any hypernet <A, E>, we produce a ***diagram*** of <A, E> as follows. Plot precisely one vertex for each member of A and label each vertex with the relevant "name" from A. Next, for every vertex adjacency of $A_i \in A$ and $A_j \in A$ in <A, E>, draw an arc between A_i and A_j, and label that arc with all the members of $\lambda(\{A_i, A_j\}) = \{ E_k \in E \mid (A_i, E_k, A_j \}$, where $\lambda: A \times A \rightarrow \wp(E)$ is called the ***labelling function*** of <A, E> and $\lambda(\{A_i, A_j\})$ is defined for every pair of members $\{A_i, A_j\}$, and $\lambda(\{A_i, A_j\}) = \emptyset$ iff there is no arc between A_i and A_j in <A, E>, i.e. if A_i and A_j are not adjacent vertices in <A, E>. Singleton edges are not usually represented by any arc. ♦

The definitions given above are illustrated in figure 2.1:

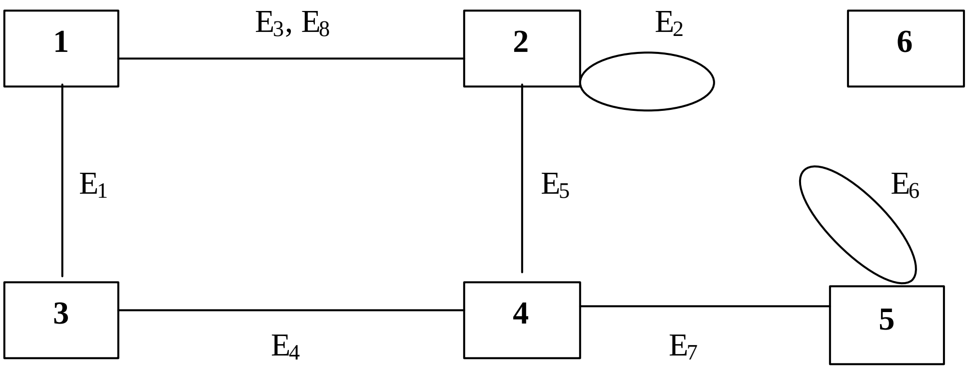


Figure 2.1: An example of a hypernet <A, E>

where $A = \{ 1, 2, 3, 4\ 5, 6\}$, $E = \{ E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8 \}$ with $E_1 = \{ 1, 2, 3 \}$, $E_2 = \{ 2\}$, $E_3 = \{ 1, 2 \}$, $E_4 = \{ 3, 4 \}$, $E_5 = \{ 2, 3, 4 \}$, $E_6 = \{ 5 \}$, $E_7 = \{ 4, 5, 6 \}$and

E_8 = { 1, 2, 3 }. Notice how we have chosen to deal with E_7, between 4 and 5, and with E_8, between 1 and 2, in this particular hypernet.

- Vertex adjacency: vertices 1 and 2 by edge E_3 and by edge E_8 for example.
- Edge adjacency: edge E_3 = { 1, 2 }and edge E_5 = { 2, 3, 4} by vertex 2 for example.
- Singleton (loop) edge: edge E_2 = { 2} and edge E_6 = { 5 }for example.

Notice that a singleton edge $\{A_k\}$, $A_k \in A$, can only have label $\{A_k\}$. Singleton edges can be thought of as representing predicates. Thus, for example, $\{A_k\}$could represent " A_k is red".

Definition 2.7: By the ***degree*** d (A_i) of a vertex $A_i \in A$ in a hypernet <A, E> we mean the sum of all the $|\lambda(\{A_i, A_j\})|$ over all $A_j \in A$ for which $\lambda(\{A_i, A_j\}) \neq \emptyset$. (Notice that we may have $A_i = A_j$, but singleton edges are not usually included.) ♦

Definition 2.8: By an ***isolate*** of a hypernet <A, E> we mean an $A_i \in A$ for which A_i is not incident with any $A_j \in A$ but A_i does belong to at least one vertex adjacency (A_r, E_j, A_s) in <A, E> with $A_r, A_s \in A$, $E_j \in E$, and $A_i \in (E_j - \{A_r, A_s\})$. By a ***complete isolate*** of <A, E> we mean an $A_i \in A$ which belongs to no edge in <A, E>.♦

Definition 2.9: By a ***walk*** in a hypernet <A, E> we mean an alternating sequence of vertices and edges,

$$A_1, E_1, A_2, E_2, A_3, \ldots, A_q, E_q, A_{q+1}, \text{ written } A_1 — A_{q+1},$$

of <A, E>, where for each k = 1, ..., q, A_k and A_{k+1} are vertex adjacent by E_k in <A, E>. The ***length*** of a walk is the number of edge entries in the sequence, in this case q. If all but possibly A_1 and A_{q+1} are distinct vertices and all the E_k, k = 1, ..., q, are distinct edges, then
$A_1 — A_{q+1}$ is called a ***path***. If $A_1 = A_{q+1}$ for a path $A_1 — A_{q+1}$, and the length of that path is any number but 2, then we call $A_1 — A_{q+1}$ a ***circuit.*** ♦

Closed paths of length 2 may exist, but we do not permit any traversal of them. Note that a closed path has length 2 iff it uses two edges from the same label.

We go back to our example in figure 2.1 and illustrate the definitions above:

- degree: d(1) = 3 and d(2) = 4 for example.
- isolate: vertex 6 is an isolate, but, by virtue of E_7 = { 4, 5, 6 }, 6 is not a complete isolate. Notice that a vertex with only a singleton edge incident with it is taken to be an isolate, even though the degree of such a vertex is 1.
- walk: 1, E_3, 2, E_2, 2, E_5, 4, E_7, 5, E_7, 4 is an example for a walk of length 5.
- path: 1, E_8, 2, E_5,, 4.
- circuit: 1, E_3, 2, E_5, 4, E_4, 3, E_1, 1.

Notice that every edge $E_i \in E$ labels one and only one vertex adjacency in <A, E>. The same set may label several vertex adjacencies, but each occurrence of that set is a distinct member of the family E. Further, any given vertex adjacency may be labelled with a number of distinct edges. Next, the reason for the introduction of singleton edges, $E_i = \{A_r\}$ for example, is to cover cases in which there is no path "through" A_r but A_r is vertex adjacent to some A_s by E_j for instance, so that $E_i \cap E_j = \{A_r\}$ and, as a result, we can legitimately talk of a path A_r, E_j, A_s incident with A_r. Finally, the reason for not regarding a closed path of length 2 as a circuit is that we should ignore this situation, which arises every time $|\lambda(\{A_r. A_s\})| \geq 2$.

The kind of structure met in the introduction is a hypernet. We should note that the final diagram in that section is that of a hypernet with circuits, but that reading such a hypernet from top-to-bottom imposes a "downward" direction an all the arcs and that with this imposed direction the circuits disappear in the sense that they become digraph semi-circuits. A similar situation arises if we read that hypernet from bottom-to-top, and we will see that this potential to rid this kind of hypernet of circuits by means of reading imposed direction can be a very significant technique in the interpretation of such structures.

To further illustrate some of the definitions that we have met, we consider the following example adapted from that given on page 110 of [Wei 83]. It deals with part of an actual module system that once existed in the Faculty of Science at the University of South Africa. The code of each module consists of a subject code of three letters followed by a level code of three digits of which the first indicates the level of study towards a degree in the faculty and the next two a module code. The modules concerned are as follows:

- Computer science: COS111, COS121, COS211, COS212, COS221, COS201, COS311, COS321, COS322, COS331, COS351, COS301.
- Information Systems: INF101, INF201, INF303.
- Mathematics: MAT101, MAT102.

What we have here is the sub-hypernet retrieved from the hypernet for the whole module system by selecting every condition set that involves COS211. As we will see, this sub-hypernet is the "context hypernet" of COS211 in the whole module system: It represents all the intermodule relational information about COS211 in that whole system. The set of module codes generates, one for one, the set of vertices of our hypernet, and the condition sets generate its edges. The parallels in each condition set are marked with an underline.

The condition sets are as follows.

1. {COS111, COS121, INF101, COS211}
2. {COS111, COS121, INF101, COS211, COS221, COS212}
3. {COS111, COS121, INF101, COS211, COS221}
4. {COS111, COS121, INF101, COS211, COS221, COS201}
5. {COS211, COS221, COS311}
6. {COS211, COS221, COS311, COS321}
7. {COS211, COS212, COS221, COS322}
8. {COS211, COS221, MAT101, MAT102, COS331}
9. {COS211, COS221, COS311, COS351}
10. {COS201, COS211, COS221, COS311, COS321, COS301}
11. {INF201, COS211, INF303}

The condition sets are those stipulated, in the system, for obtaining credit for the final module in each membership list. We can choose any prerequisite from a list as the other end vertex of that list. Bearing in mind potential edge adjacencies it is of course possible, then, to plot each condition as a number of edges, but to avoid unnecessary repetition of condition sets we use each condition set only once, and as a heuristic it is advisable to "start" each edge at a module of lower level than that of the module for which the condition is stipulated, thus making the interpretation of the diagram simpler.

A diagram for these modules and these condition sets, a hypernet diagram, is given in figure 2.2. Note that there are four isolates, but none of them is a complete isolate.

Reading from left to right (bottom-to-top) we can determine how credit may be obtained for an end vertex of each edge and of each path. Reading from right to left (top-to-bottom) we get the same information in a different form. It will become clear later, when we deal with "cascades", that this difference of form is not trivial.

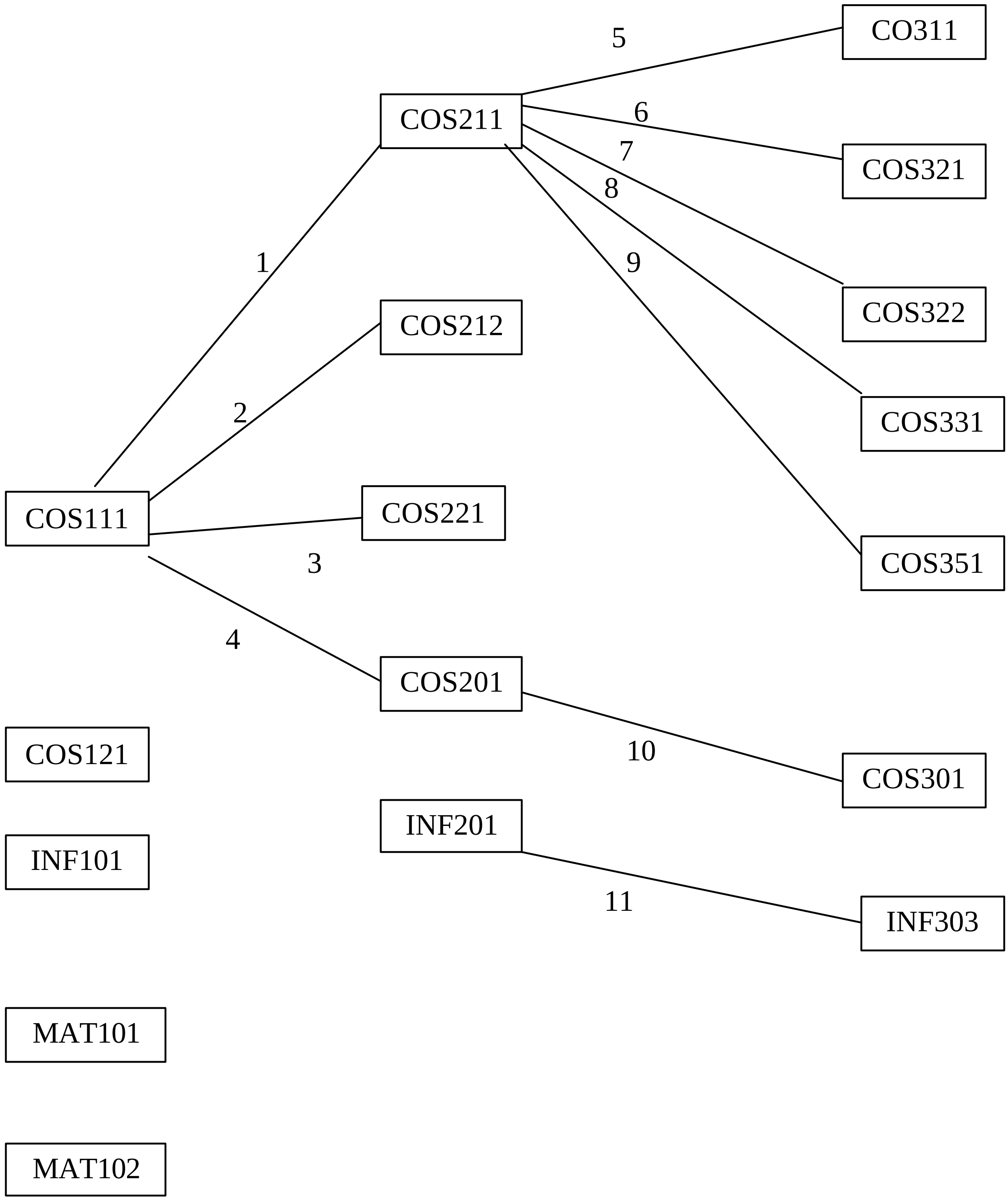


Figure 2.2: A diagram for part of a module system

Next we look at the connection between hypernets and the relation nets introduced and explored in [GVS99].

Definition 2.10: By a ***tuple-specific relation net interpretation***, or simply an ***interpretation***, of a hypernet <A, E> we mean a one-to-one correspondence I: A → A that maps <A, E> to a relation net <A, T> as follows. For every vertex adjacency (A_r, E_i, A_s) in <A, E> with $E_i = \{A_1, A_2, \ldots, A_{\oplus}, \ldots, A_{n(i)}\} \subseteq A$, $A_r \in E_i$, $A_s \in E_i$, (A_r, E_i, A_s) is mapped to at least one tuple $T_i \in T$ with $T_i = < B_1, B_2, \ldots, B_k, \ldots, B_{m(i)} >$ and with either $B_1 = A_r$ and $B_{m(i)} = A_s$ or $B_1 = A_s$ and $B_{m(i)} = A_r$ and for every B_k, k = 2, ...m(i) - 1, $B_k = I(A_{\oplus})$ for some one $A_{\oplus} \in E_i$, $\oplus$ = 1, 2, ..., n(i), and every member of E_i is used at least once as an entry in T_i so E_i is the tuple set of T_i, $| T_i | = m(i) \geq | E_i | = n(i)$, and this holds for each vertex adjacency by each $E_j \in E$ and for each $T_j \in T$. We write $T_j = I [E_j]$ and <A, T> = I [<A, E>], and $| T_i |$ is equal to the number of distinct vertex adjacencies in <A, E>.♦

Definition 2.11: Each hypernet <A, E> has a countably infinite set of distinct interpretations, and this set is called a ***realization*** of <A, E>.♦

Next we describe the move from relation nets to hypernets.

Definition 2.12: Consider any given relation net <A, T>. By an ***edge-specific hypernet abstraction***, or simply an ***abstraction***, of <A, T> we mean a one-to-one correspondence M: A → A that maps <A, T> to a hypernet <A, E> and is defined as follows. For every tuple $T_i = < A_1, A_2, \ldots, A_{\oplus}, \ldots, A_{n(i)} > \in T$ in <A, T> the mapping M produces a set $E_i = \{M(A_1), \ldots, M(A_{\oplus}), \ldots, M(A_{n(i)})\} \in E$ with $| E_i | \leq | T_i |$, the tuple set of T_i, and a vertex adjacency $(M(A_1), E_i, M(A_{n(i)}))$ in <A, E> for every $T_i \in T$. This results in the hypernet <A, E> and we write $E_i = M [T_i]$ and <A, E> = M [<A, T>] . ♦

Each relation net <A, T> has a unique abstraction M [<A, T>] but a countably infinite set of distinct relation nets can all have the same abstraction. Obviously,

Theorem 2.1: Every abstraction M of a relation net <A, T> with M [<A, T>] = <A, E> is the inverse of some interpretation I of <A, E> with I [<A, E>] = <A, T>, and the converse is also true. ♦

In dealing with relation nets in [GVS99] we faced the problem (in Part I) that each tuple came from a statement of relationship among concept-names, and could thus be permuted by re-wording that statement without changing the relationship among those concept-names involved in that statement. The following definition opens up all the possible permutations of tuples in a CRKS for examination and choice of "appropriate" ones.

Definition 2.13: By the ***completion*** of a hypernet <A, E> we mean that unique hypernet that is constructed from <A, E> by adding to <A, E> every potential edge adjacency, and hence every potential vertex adjacency, of <A, E> that is not in <A, E>, i.e. we "fill in" all the sets $E_i \cap E_j$, and thus all the vertex adjacencies that then arise, for all distinct E_i and E_j, i.e.for all $i \neq j$. ♦

For each $A_r \in A$ for which we have (E_i, A_r, E_j) for some E_i and E_j, i.e. $A_r \in (E_i \cap E_j)$, $i \neq j$, in the completion of M [<A, T>], the tuples T_i and T_j with M $[T_i] = E_i$ and M $[T_j] = E_j$ can be permuted so that they are adjacent at A_r in a new CRKS that models the same relationships as does T.

Given the completion of an abstraction M [<A, T>], we can interpret sub-hypernets of that completion to produce goal oriented application CRKS's from <A, T>. This leads to the following definitions.

Definition 2.14: By a ***sub-hypernet*** of a hypernet <A, E> we mean a hypernet <B, U> with $B \subseteq A$, $U \subseteq E$, and every $E_i \in U$ is such that $E_i \in E$. Further, every vertex adjacency of <B,U> by E_j is a vertex adjacency of <A, E> by E_j. If B = A then we call <B, U> a ***spanning*** sub-hypernet of <A, E>. We write <B, U> $\angle$ <A, E>.♦

Definition 2.15: The ***maximum sub-hypernet*** <B, E↑B>, of a hypernet <A,E>, that is induced by $B \subseteq A$, is such that $E_i \in E$ belongs to E↑B iff $E_i \subseteq B$. ♦

Let <A,E> be any hypernet and let X be the set of all those sub-hypernets of <A,E> that are of the form < A – B, E ↑(A – B) > where $B \subseteq A$. Then < X, $\angle$ > is a distributive lattice under $\cup$ and $\cap$ of hypernets, with null element <ϕ ,ϕ> and universal element <A,E>. This can be shown easily because $\cup$ and $\cap$ for hypernets are defined in terms of set $\cup$ and set $\cap$ respectively.

There is a one-to-one correspondence between the set of walks in a hypernet <A, E> and the set of semi-walks in any given interpretation I [<A, E>] of <A, E>.

To close this section we turn our attention to the question of isomorphism. In Part I of [GVS99] we defined structural analogy of CRKS's in terms of CRKS isomorphism, giving - to the best of our knowledge - the first formal definition of analogy. The notion of formalized analogical reasoning, and of teaching/learning by analogical modelling, is critical to the work in Part I of [GVS99], and a key to the practical use of structural analogy is the rather complex constructional scheme given there for finding isomorphic (sub-) relation nets. It appears that we can do a little bit better, through the medium of hypernets, by side-stepping the problems involved in relative permutation differences between potentially isomorphic (sub-) relation nets. To begin, we revise the definition of isomorphism of relation nets.

Definition 2.16: Given two relation nets <A, S> and <B, T> with | A | = | B | and | S | = | T |, we say that <A, S> and <B, T> are ***isomorphic*** iff there exists a pair of one-to-one correspondences g: A→ B and h: S→ T which are such that tuple $T_i = \langle A_1, ..., A_r, ..., A_n\rangle$, where each entry is an entry of a member of A, belongs to S iff tuple $h(T_i) = \langle B_1, ..., B_s, ..., B_m\rangle$, belongs to T, where m = n and where each entry is an entry of a member of B, and $B_1 = g(A_1)$, $B_m = g(A_n)$, and every entry A_r, $r \neq 1$ and $r \neq n$, in T_i is mapped to some $B_s = g(A_r)$ with r not necessarily equal to s. ♦

The equivalent for hypernets is rather less taxing, and is as follows.

Definition 2.17: Two hypernets $<A_1, E_1>$ and $<A_2, E_2>$, with $| A_1| = | A_2|$ and $| E_1| = | E_2|$, are said to be ***isomorphic*** iff there exists a pair of one-to-one correspondences g: $A_1 \rightarrow A_2$ and h: $E_1 \rightarrow E_2$ such that $A_{1i} \in A_1$ belongs to $E_{1j} \in E_1$ iff $g(A_{1i})$ belongs to $h(E_{1j})$ and (A_{1i}, E_{1j}, A_{1k}) is a vertex adjacency in $<A_1, E_1>$ iff $(g(A_{1i}), h(E_{1j}), g(A_{1k}))$ is a vertex adjacency in $<A_2, E_2>$.♦

Given two hypernets $<A_1, E_1>$ and $<A_2, E_2>$, how can we find an isomorphism between them? We can use the following.

Constructional scheme 2.1

Step 1: Check that $| A_1| = | A_2|$ and $| E_1| = | E_2|$. Indeed if $| A_1| < | A_2|$ and/or $| E_1| < | E_2|$ we may be able to find an isomorphism between $<A_1, E_1>$ and a sub-hypernet $<B, U> \angle <A_2, E_2>$ with $| A_1| = | B|$ and $| E_1| = | U|$.

Step 2: Let (A_{11}, E_{1i}, A_{12}) be any vertex adjacency in $< A_1, E_1>$. Try to match (A_{11}, E_{1i}, A_{12}) with some vertex adjacency (A_{21}, E_{2j}, A_{22}) in $< A_2, E_2>$ for which we can begin to define g and h by setting $g(A_{11}) = A_{21}$, $g(A_{12}) = A_{22}$, and $h(E_{1i}) = E_{2j} \in E_2$ such that $E_{2j} = \{g(A_{11}), g(A_{12})\} \cup \{g(A_{1k}) \in A_2 \mid A_{1k} \in A_1 \text{ and } A_{1k} \in (E_i - \{A_{11}, A_{12}\})$ and $| E_{1j} | = | E_{2j} |$ so that $| E_{1i} | = | h(E_{1i}) = E_{2j} |$. If we can find no such matching then no isomorphism <g, h> exists.

Step 3: If we can find one such partial matching of an (A_{11}, E_{1i}, A_{12}) and some (A_{21}, E_{2j}, A_{22}), then the next step is as follows. Try to expand the present domains of g and h to incorporate all vertex adjacencies that involve A_{11} and/or A_{12} in $< A_1, E_1>$. Do this for as many "new" vertex adjacencies of this kind as possible. If there are "new" adjacencies that cannot be covered, disregard them. Move to step 4. If there are no "new" vertex adjacencies that can be covered in this step, return to step 2 and start over with another vertex adjacency in $< A_1, E_1>$.

Step 4: Try, as in step 3, to expand the present domains of g and h to cover all vertex adjacencies in $< A_1, E_1>$ that involve at least one of the "already covered" vertices of $< A_1, E_1>$. If no expansion is possible, return to step 2 and start over with another vertex adjacency in $< A_1, E_1>$; otherwise move to step 5.

Step 5: Repeat step 4 until no more vertex adjacencies in $< A_1, E_1>$ can be covered, or until we get any contradiction. At that stage we have an isomorphism from a sub-hypernet of $< A_1, E_1>$ into $< A_2, E_2>$. If that sub-hypernet is not $< A_1, E_1>$ then we store the isomorphism and start over with step 2, eventually finding several hopefully non-trivial (i.e. not just a single vertex adjacency that is isomorphic with some vertex adjacency in $< A_2, E_2>$ sub-hypernets of $< A_1, E_1>$ that are isomorphic with some sub-hypernet of $< A_2, E_2>$. From those isomorphisms that we find, we can choose the most appropriate for our purpose at the time of choice. Recall from [GVS99] that several different sub-hypernets of $< A_1, E_1>$ can serve as isomorphic structural models/analogies of the same sub-hypernet of $< A_2, E_2>$, and one sub-hypernet of $< A_1, E_1>$ can serve as an isomorphic structural model/analogy for several different sub-hypernets of $< A_2, E_2>$.♦

In applying constructional scheme 2.1 we must take account of

- other edges in E_1 by which A_{11} and A_{12} are vertex adjacent,
- other edges in E_2 that are "set equal" to E_{2j}, and
- potential edge adjacencies by > 1vertices in $< A_1, E_1 >$, when trying to find an initial vertex adjacency match in step 2 of the scheme.

Now it appears that it may well be easier in general to automate the search for hypernet isomorphisms than for relation net isomorphisms due to the necessity to take into account matching tuples "modulo relative permutation" in the latter case. With this in mind, we present the following two theorems.

To set the scene, let $< A_1, E_1 >$ and $< A_2, E_2 >$ be hypernets and let $< A_1, T_1 >$ and $< A_2, T_2 >$ be relation nets and let $| A_1 | = | A_2 |$, $| E_1 | = | E_2 |$, and $| T_1 | = | T_2 |$. Further, let
$D_1 = \{(A_{11}, E_{1j}, A_{12}) \mid A_{11}, A_{12} \in A_1, E_{1i} \in E_1$, and (A_{11}, E_{1i}, A_{12}) is a vertex adjacency in $< A_1, E_1 > \}$,
$D_2 = \{(A_{21}, E_{2i}, A_{22}) \mid A_{21}, A_{22} \in A_2, E_{2i} \in E_2$, and (A_{21}, E_{2i}, A_{22}) is a vertex adjacency in $< A_2, E_2 > \}$,
and let $| D_1 | = | D_2 |$. Now consider the following diagram:

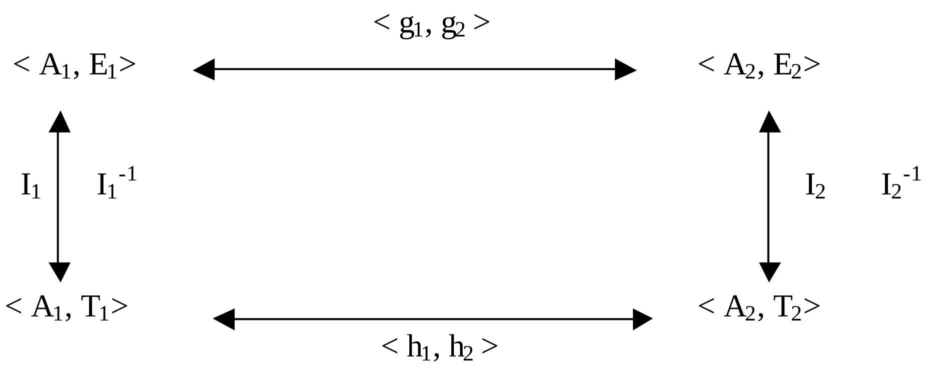


Figure 2.3: Isomorphisms and interpretations

Here $< g_1, g_2 >$ is a hypernet isomorphism and $< h_1, h_2 >$ is a relation net isomorphism, I_1 and I_2 are interpretations, and all these mappings are one-to one-correspondences, so their inverses are well defined simple reversals.

Theorem 2.2: Let $< A_1, E_1 >$ and $< A_2, E_2 >$ in the diagram be isomorphic hypernets. Then there exist interpretations $I_1 [< A_1, E_1 >] = < A_1, T_1 >$ and $I_2 [< A_2, E_2 >] = < A_2, T_2 >$ such that $< A_1, T_1 >$ and $< A_2, T_2 >$ are isomorphic relation nets. ♦

Proof: Consider any vertex adjacency (A_{11}, E_{1i}, A_{12}) in $< A_1, E_1 >$. The matching vertex adjacency is $(g_1(A_{11}), g_2(E_{1i}), g_1(A_{12}))$ in $< A_2, E_2 >$. I_1 is defined as follows. I_1 takes (A_{11}, E_{1i}, A_{12}) to precisely one n_i-tuple $T_{1i} \in T_1$ in $< A_1, T_1 >$. Let
$T_{1i} = < I_1 (A_{11}), ..., I_1 (A_{1r}), ..., I_1 (A_{12}) >$ where the entries other than $I_1 (A_{11})$ and $I_1 (A_{12})$ consist of n_i - 2 entries of some $I_1 (A_{1r})$ with $A_{1r} \in E_{1i}$ and A_{1r} may be A_{11} or A_{12} and A_{11} may be equal to A_{12}, and where every member of E_{1i} is mapped to at least one entry in $T_{1i} = I_1 [E_{1i}]$. I_2

is now defined to map $(g_1(A_{11}), g_2(E_{1i}), g_1(A_{12}))$ in $< A_2, E_2>$ to precisely one tuple $T_{2j} \in T_2$ where $T_{2j} = < I_2 (A_{21}), ..., I_2 (A_{2k}), ..., I_2 (A_{22})>$ with $A_{21} = g_1(A_{11})$, $A_{22} = g_1(A_{12})$ and every entry $A_{2k} = g_1(A_{1r})$ with k not necessaryly equal to r, and where every member of $g_2(E_{1i})$ is mapped to at least one entry in $T_{2j} = I_2 [(g_2(E_{1i}))]$. Now we define $< h_1, h_2 >$ such that $h_1: A_1 \to A_2$ and $h_2: T_1 \to T_2$ are both one-to-one correspondences, and for every $T_{1i} = < I_1 (A_{11}), ..., I_1 (A_{1r}), ..., I_1 (A_{12})> \in T_1$, $h_2 (T_{1i}) \in T_2$ is given by
$h_2 (T_{1i}) = < h_1 (I_1 (A_{11})), ..., h_1 (I_1 (A_{1r})), ..., h_1 (I_1 (A_{12})) >$ with
$h_1 (I_1 (A_{11})) = I_2 (A_{21}) = I_2 (g_1(A_{11}))$,
$h_1 (I_1 (A_{12})) = I_2 (A_{22}) = I_2 (g_1(A_{12}))$,
$h_1 (I_1 (A_{1r})) = I_2 (A_{2k}) = I_2 (g_1(A_{1r}))$
where the number of entries in T_{1i} and $h_2 (T_{1i})$ is clearly the same, and every $I_1 (A_{1r})$, $r \neq 1$ and $r \neq 2$, in T_{1i} is mapped to some $I_2 (A_{2k})$ with k not necessarily equal to r. Thus, $< h_1, h_2 >$ is a relation net isomorphism that maps $< A_1, T_1>$ onto $< A_2, T_2>$.♦

Theorem 2.3: Let $< A_1, T_1>$ and $< A_2, T_2>$ in our diagram be isomorphic relation nets. Then there exist abstractions $M_1[< A_1, T_1>] = < A_1, E_1>$ and $M_2[< A_2, T_2>] = < A_2, E_2 >$ such that $< A_1, E_1>$ and $< A_2, E_2 >$ are isomorphic hypernets. ♦

Proof: In the proof of theorem 2.2 we constructed $< h_1, h_2 >$. Here we will construct $< g_1, g_2 >$, given that $< h_1, h_2 >$ is an isomorphism. Essentially, what we do is to set $M_1 = I_1^{-1}$ and $M_2 = I_2^{-1}$ and reverse the process of the proof of theorem 2.2. An arbitrary tuple T_{1i} in $< A_1, T_1>$, with $T_{1i} = < A_{11}, ..., A_{1r}, ..., A_{12}>$ is matched with precisely one tuple $h_2 (T_{1i}) = < A_{21}, ..., A_{2k}, ..., A_{22}>$ with $A_{21} = h_1 (A_{11})$, $A_{22} = h_1 (A_{12})$ and $A_{2k}= h_1 (A_{1r})$ with $k \neq 1$ and $k \neq 2$ and k not necessaryly equal to r. Now apply I_1^{-1} to $< A_1, T_1>$ and I_2^{-1} to $< A_2, T_2>$. $T_{1i} = < A_{11}, ..., A_{1r}, ..., A_{12} >$ is mapped, by I_1^{-1}, to the tuple set, $E_{1i} \in E_1$, of T_{1i} and a vertex adjacency $(I_1^{-1} (A_{11}) , E_{1i}, I_1^{-1} (A_{12}))$, and $h_2 (T_{1i}) = < h_1 (A_{11}), ..., h_1 (A_{12})> \in T_2$ is mapped, by I_2^{-1}, to the tuple set, $E_{2j} \in E_2$, of $h_2 (T_{1i})$ and a vertex adjacency $(I_2^{-1} (h_1 (A_{11})) , E_{2j}, I_2^{-1} (h_1 (A_{12}))$. Now it is easy to see that we can define a hypernet isomorphism $< g_1, g_2 >$ from $< A_1, T_1>$ onto $< A_2, T_2>$ simply by setting
$I_2^{-1} (h_1 (A_{11})) = g_1 (I_1^{-1} (A_{11}))$,
$I_2^{-1} (h_1 (A_{12}))= g_1 (I_1^{-1} (A_{12}))$,
$g_2 (E_{1i}) = E_{2j} = g_2 (\{ A_{11}, ..., A_{1r}, ..., A_{12}\}) = \{ g_1(A_{11}), ..., g_1(A_{1r}), ..., g_1(A_{12})\}$, and $E_{1i} \neq \emptyset$.♦

These two theorems can be of considerable assistance. In particular, theorem 2.2 can help in finding relation net isomorphisms.

Definition 2.18: Let $<A_1, E_1>$ and $<A_2 ,E_2 >$ be hypernets, and $<A_1,T_1>$ and $<A_2,T_2>$ be relation nets, and consider the following diagram:

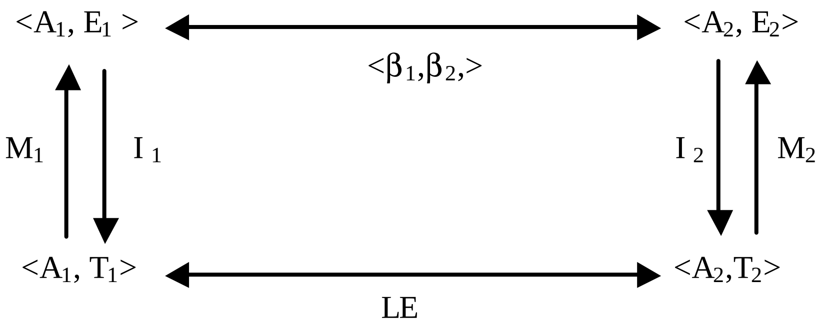


Figure 2.4: Abstraction isomorphism

Here the abstraction M_1 is the inverse of the interpretation I_1 and M_2 the inverse of I_2, and $<\beta_1, \beta_2>$ is a hypernet isomorphism. Each tuple $T_{1i} \in T_1$ is mapped to its tuple set $M_1 [T_{1i}]$ in $<A_1, E_1>$, then by $<\beta_1, \beta_2>$ to the isomorphic tuple set $< \beta_1, \beta_2> (M_1 [T_{1i}])$ in $<A_2, E_2>$, and thence by I_2 to a tuple $I_2 (<\beta_1, \beta_2> (M_1 [T_{1i}])) = T_{2j}$, where if T_{1i} is an n_{1i}-tuple then $I_2(< \beta_1,\beta_2>(M_1 [T_{1i}]))$ is an n_{2j} – tuple with n_{1i} and n_{2j} both at least $| M_1 [T_{1i}] | = | < \beta_1, \beta_2> (M_1 [T_{1i}])|$ and n_{1i} and n_{2j} are not necessarily equal, and $T_{1i} = I_1 (< \beta_1, \beta_2> (M_2[T_{2j}]))$, and this holds for each $T_{1i} \in T_1$ and each $T_{2k} \in T_2$. We call $<A_1, T_1>$ and $<A_2, T_2>$ ***language equivalent (LE) relation nets*** iff for each $T_{1i} \in T_1$ there is at least one $T_{2j} = I_2(<\beta_1, \beta_2>(M_1[T_{1i}])) \in T_2$ and for each $T_{2j} \in T_2$ there exists at least one $T_{1i} = I_1 (<\beta_1, \beta_2>(M_2 [T_{2j}])) \in T_1$. ♦

Given a study material CRKS – see [GVS 99] – for which the statements are set out in language A, we can use the definition to find a "language equivalent" CRKS in which the statements are set out in another teaching language B. LE is an equivalence relation on the class of relation nets.

3. First intermission

Suppose we think of a hypernet <A, E> in terms of its diagram. Let $S \subseteq A$, and let $\{a,b\} \subseteq S$, $a, b \in A$ and $|S| \geq 2$. Then a and b are potentially adjacent vertices by S. Now we should notice that if {a,b} is an actual vertex adjacency by S in <A, E> then $S \in \lambda(\{a,b\})$, and S can belong to the label of more than one vertex adjacency in <A, E>, and furthermore the set S can appear in a given label such as $\lambda(\{a,b\})$ more than once. To handle this we let every distinct occurence of S in any label or labels be entered as a separate member of the family E of edges of <A, E>. Thus, if set S occurs m times in vertex adjacency labels, some of these occurences perhaps in the same label, each label is indeed a set as S will appear m times in the family of edges E of <A, E>, i.e.

$$E_n, E_{n+1}, \ldots, E_{n+m-1}$$

all of which are entries of the same set S in the family E. If {a,b} is a vertex adjacency by $E_i \in E$ then $E_i \in \lambda(\{a,b\})$ and not $E_i - \{a,b\}$, a significant difference from the similar situation for relation nets – see [GVS99]. Here each edge characterizes one and only one vertex adjacency, except of course for singleton edges.

Deleting an edge $E_i \in E$ from <A, E> takes E_i out of one vertex adjacency label, $\lambda(\{a,b\})$ for example. The arc between a and b will then disappear only if $\lambda(\{a,b\}) = \{E_i\}$. Deleting a vertex adjacency (a, E_i, b) of a and b by E_i from <A, E> also means taking E_i out of $\lambda(\{a,b\})$. We sometimes refer to "the vertex adjacency {a,b}". Deleting a vertex $v \in A$ from <A, E> entails removing v from <A, E> together with every $E_i \in E$ that has $v \in E_i$.

Much of the theory of relation nets covered in [GVS99] can be transcribed to hypernet theory. The key to such transcription is basically the following:

Relation Nets	Hypernets
tuple occurence a,i; j,b	vertex adjacency (a, E_i, b)
single (F(i))(j)	single edge E_i
set R of (F(i))(j)'s	set R of edges

Figure 3.1: Connection between relation nets and hypernets

In this report we will be transcribing to hypernets only a selection of the theory of relation nets covered in [GVS99]. We begin with some general theory of hypernets and then move on to transcription of some of the theory of Concept-Relationship Knowledge Structures, bearing in mind our potential examples of hypernets as described in the first section of this report.

4. Introduction to a theory of general hypernets

Definition 4.1: For $A_i \in A$ of a hypernet <A, E> we define:

(1) The set $E(A_i) \subseteq E$ of all edges ***in the name of*** A_i by $E(A_i) = \{ E_j \in E \mid$ for every vertex adjacency of the form (A_r, E_j, A_s) in <A, E> we have $A_i \in (E_j - \{ A_r, A_s \}) \}$.

(2) The set $E[A_i] \subseteq E$ of all ***edges with*** A_i by $E[A_i] = \{ E_j \in E \mid A_i \in E_j \}$.

(3) E(B) denotes the set of all $E(A_i)$ with $A_i \in B$ and a similar statement applies to E [B], with $B \subseteq A$. ♦

Definition 4.2:

(1) The ***meet*** <A, E> of two hypernets <B, F> and <C, G> is defined by <A, E> = $< B \cap C, F \cap G >$ and <A, E> is a unique hypernet.

(2) The ***join*** <A, E> of two hypernets <B, F> and <C, G> is defined by <A, E> = $< B \cup C, F \cup G >$ and <A, E> is a unique hypernet.

(3) The meet of <B, F> anc <C, G> is written <B, F> $\cap$ <C, G>, and their join is written <B, F> $\cup$ <C, G>. ♦

In part (1) the only way in which F and G can share edges is that those shared edges are subsets of $B \cap C$. Thus we have the following

Theorem 4.1: If $E_i \in E$, and hypernet <A, E> = $< B \cap C, F \cap G >$ is the meet of hypernets <B, F> and <C, G> , then $E_i \subseteq (B \cap C)$, but the converse is not necessarily true. ♦

Proof: The first part is trivial. For the converse, we notice that $E_i \subseteq (B \cap C)$ can be true if E_i belongs to only one of F or G. ♦

The join and meet operations may of course be successfully applied to the sub-hypernets of a given hypernet.

Definition 4.3: The ***adjacency function*** $\Gamma: A \rightarrow \wp(A)$ of a hypernet <A, E> is defined by, for all $A_r \in A$, $\Gamma(A_r) = \{ A_s \in A \mid (A_r, E_j, A_s)$ for some $E_j \in E\} \cup \{A_r\}$. ♦

Definition 4.4: By a ***walk-family*** $f(A_r — A_s)$ in a hypernet <A, E> we mean a non-empty set of walks between A_r and A_s in <A, E>, the members of which all have the same subsequence over A while being pairwise distinct in edge subsequences over E. By a ***sub-walk-family*** of $f(A_r — A_s)$, we mean a walk-family $f(A_m — A_n)$, $r \leq m < n \leq s$, for which every member is a subsequence of at least one member of $f(A_r — A_s)$. ♦

A walk family can have just one member.

Definition 4.5:

(1) Let A_r, A_j, $A_s \in A$ in a hypernet <A, E> , and let $A_r — A_s$ be a given walk in <A, E>. Then A_j is said to be ***vertex between*** A_r and A_s on $A_r — A_s$ iff A_j belongs to the vertex

subsequence of A_r — A_s or to at least one edge E_i that lies in the walk A_r — A_s, or both. (Of course both cases are covered if A_j belongs to at least one of the edges of the walk.)

(2) A_j is said to be ***reachable*** from A_r in <A, E> iff there is a path between A_r and A_j in <A, E>.

(3) The ***reachability function*** $\mathfrak{R}: A \rightarrow \wp(A)$ of a hypernet <A, E> is defined by, for all $A_r \in A$, $\mathfrak{R}(A_r) = \{ A_s \in A \mid A_s$ is reachable from A_r in <A, E>$\}$.

(4) The meanings of $\Gamma(B)$ and $\mathfrak{R}(B)$ for $B \subseteq A$ are obvious. ♦

Next we tackle the notion of a cascade, starting with a revision of the definition for relation nets given in [GVS99].

Definition 4.6: The nested sequence $\{\langle B_k, R_k \rangle \mid k \geq 0\}$ of subnets of a relation net <A, T> is called a ***fast access cascade*** from B_0 iff

(1) $B_0 \subseteq A$ and $R_0 = \emptyset$, and

(2) $R_1 \subseteq T$ is chosen in such a way that $T_i = \langle A_1, A_2, \ldots, A_{\oplus}, \ldots, A_{n(i)} \rangle \in T$ belongs to R_1 iff $A_1 \in B_0$, and

(3) $B_1 = B_0 \cup$ (the union of the tuple sets of the members of R_1), where the tuple set of $T_i \in T$ is the set of all $A_r \in A$ such that A_r is at least one entry in T_i, and in general for k = 2, 3, ...,

(4) $R_k \in T$ is chosen in such a way that $T_i = \langle A_1, A_2, \ldots, A_{\oplus}, \ldots, A_{n(i)} \rangle \in T$ belongs to R_k iff $A_1 \in B_{k-1}$, so $R_{k-1} \subseteq R_k$, and

(5) $B_k = B_{k-1}$ È (the union of the tuple sets of the members of R_k), so $B_{k-1} \subseteq B_k$.

Such a cascade is said to be a ***limited access cascade*** from B_0 in <A, T> iff at each step k = 1, 2, ... we choose $T_i = \langle A_1, A_2, \ldots, A_{\oplus}, \ldots, A_{n(i)} \rangle \in T$ in such a way that $T_i \in R_k$ iff $\{A_k \in A \mid k = 1, 2, \ldots, n(i)-1\} \subseteq B_{k-1}$, and where $A_{n(i)} \in A$ may or may not belong to B_{k-1}. ♦

A cascade will stop when $\langle B_k, R_k \rangle = \langle B_{k-1}, R_{k-1} \rangle$ or when $\langle B_k, R_k \rangle = \langle A, T \rangle$.

For hypernets we have the following transcription.

Definition 4.7: The nested sequence $\{\langle B_k, D_k \rangle \mid k \geq 0\}$ of sub-hypernets of a hypernet <A, E> is called a ***fast access cascade*** from B_0 iff

(1) $B_0 \subseteq A$ and $D_0 = \emptyset$, and

(2) $D_1 \subseteq E$ is chosen in such a way that for each vertex adjacency (A_r, E_j, A_s), $E_j \in E$, E_j belongs to D_1 iff A_r or A_s belongs to B_0, and

(3) $B_1 = B_0$ È (the union of all the E_j that belong to D_1), and in general for k = 2, 3, ...

(4) $D_k \in E$ is chosen in such a way that for each vertex adjacency (A_r, E_j, A_s), $E_j \in E$, E_j belongs to D_k iff A_r or A_s belongs to B_{k-1}, so $D_{k-1} \subseteq D_k$, and

(5) $B_k = B_{k-1}$ È (the union of all the E_j that belong to D_k), so $B_{k-1} \subseteq B_k$.

Such a cascade is said to be a ***limited access cascade*** from B_0 in <A, E> iff, at each step k = 1, 2, ..., we choose $E_i \in D_k$ iff all, but possibly one, of the members of E_i belong to B_{k-1}, and that one is either A_r or A_s in each vertex adjycency (A_r, E_i, A_s) used in choosing the $E_i \in D_k$, k = 1, 2, ... ♦

Note that that particular one of A_r or A_s in each case does of course belong to A, but may or may not belong to B_{k-1}. Again such cascades stop on the same conditions as for relation net cascades.

Hypernets all exhibit ***strong vulnerability***: If we delete $A_k \in A$ from a hypernet <A, E> then we delete every edge adjacency by A_k in <A, E>, and also every edge $E_i \in E$ for which $A_k \in E_i$, i.e. we delete E [A_k]. Because strong vulnerability expresses context sensitivity in certain hypernets - see [GVS99] and later work in this report - we introduce the following notion.

Definition 4.8: By the ***context hypernet*** <A, E>[A_k] of $A_k \in A$ in a hypernet <A, E> we mean that sub-hypernet of <A, E> that consists of every $E_i \in E$ that has $A_k \in E_i$, i.e. E [A_k] together with the set of vertices $\{A_j \in A | A_j \in E_i$ and $E_i \in E [A_k]\}$. <A, E>[A_k] is a hypernet. ♦

We return to our example in figure 2.1 and illustrate the different notions defined so far in this section:

- set E (A_k) and E [A_k]: E(3) = { E_5} and E[3] = { E_1, E_6, E_5, E_4 }.
- adjacency function: $\Gamma(4) = \{2, 3, 4, 5\}$, and $\Gamma(5) = \{5, 4\}$.
- walk-family: f (2 — 5) = {(2, E_5, 4, E_7, 5), (2, E_2, 2, E_5, 4, E_7, 5), (2, E_5, 4, E_7, 5, E_6, 5), (2, E_2, 2, E_5, 4, E_7, 5, E_6, 5} or any non-empty subset of this set.
- reachable: $\Re$ (2) = A - {6}, $\Re$ (2) = $\Re$ (1) = $\Re$ (3) = $\Re$ (4) = $\Re$ (5).
- fast access cascade: B_0 = {2}, B_1 = {2, 1, 4, 3}, B_2 = {2, 1, 4, 3, 5}.
- limited access cascade: B_0 = {1, 2}, B_1 = {1, 2, 3}, B_2 = {1, 2, 3, 4}, B_3 = {1, 2, 3, 4}, stop.
- context hypernet: The context hypernet of $4 \in A$, i.e. <A, E>[4] , has vertex set A[4] = {4, 2, 3, 5} and edge set E[4] = { E_5, E_4, E_7}.

We will see that the notion of a cascade, which may be regarded here as a controlled search technique, becomes an essential part of the theory of the hypernet equivalent of a CRKS.

5. Menger's theorem

We will introduce the theorem, and state and prove it, stage by stage in parallel for relation nets and hypernets; (a) denotes the part for relation nets, (b) that for hypernets.

Definition 5.1:

(a) The ***path-net*** N(P) of a path P in a relation net <A, T> is the minimum subnet <B, U> ∠ <A, T> that contains P. By this we mean that U ⊆ T is the set of tuples that appear in P, and B is the union of all the tuple sets of the members of U. N(P) is a minimum subnet inasmuch as if we delete any member of U or any member of B then P no longer lies in the resulting relation net.

(b) The ***path-hypernet*** N(P) of a path P in a hypernet <A, E> is the minimum sub-hypernet <B, U> ∠ <A, E> that contains P. By this we mean that U ⊆ E is the set of edges that appear in P, and B is the union of all the members of U. N(P) is a minimum sub-hypernet inasmuch as if we delete any member of U or any member of B then P no longer lies in the resulting hypernet.♦

Definition 5.2:

(a) Two u → v paths, P_k and P_m, in a relation net <A, T>, are said to be ***interdependent paths*** iff the meet $N(P_k) \cap N(P_m)$ of their path-nets has at least one vertex other than u and v in it. A set $\{P_0, ..., P_n\}$ of u → v paths in <A, T> is called an ***interdependent set*** iff Ç $N(P_r)$, r = 0, 1, ..., n, has at least one vertex other than u and v in it, and it is a ***maximal interdependent set*** iff it is not a proper subset of any interdependent set of u → v paths in <A, T>.

(b) Two u — v paths, P_k and P_m, in a hypernet <A, E>, are said to be ***interdependent paths*** iff the meet $N(P_k) \cap N(P_m)$ of their path-hypernets has at least one vertex other than u and v in it. A set $\{P_0, ..., P_n\}$ of u — v paths in <A, E> is called an ***interdependent set*** iff Ç $N(P_r)$, r = 0, 1, ..., n, has at least one vertex other than u and v in it, and it is a ***maximal interdependent set*** iff it is not a proper subset of any interdependent set of u — v paths in <A, E>.♦

Notice that the semi-paths in <A, T> are equivalent with the paths in <A, E> = M [<A,T>].

Theorem 5.1:(see theorem 12.6, p. 205 of [GVS99])

(a) Let $\{P_0, ..., P_n\}$ be any interdependent set of u → v paths in <A, T>. Deletion of any w ∈ (A - {u,v}) that belongs to the vertex set of Ç N(Pr) from <A, T> will "cut" all the paths P_r, i.e. none of the paths of the set exists in the subnet which results when w is deleted from <A, T>.

(b) Let $\{P_0, ..., P_n\}$ be any interdependent set of u — v paths in <A, E>. Deletion of any w ∈ (A - {u,v}) that belongs to the vertex set of Ç $N(P_r)$ from <A,E> will "cut" all the paths P_r, i.e. none of the paths of the set exists in the sub-hypernet which results when w is deleted from <A, E>.♦

Proof:

(a) for <A, T>: see [GVS99].

(b) for <A, E>: We must show that if w is a vertex, with w ≠ u and w ≠ v, of Ç $N(P_r)$, then it is between u and v on every P_r. Let w be a vertex of Ç $N(P_r)$, and assume that w is not between u and v on some P_t. Then w does not belong to the vertex set of $N(P_t)$, and hence it is not a vertex of Ç $N(P_r)$, which contradicts the hypothesis. ♦

Theorem 5.2: (see theorem 12.7, p. 205 of [GVS99])

(a) Let S = {P_0, ..., P_n} be a maximal interdependent set of u → v paths in <A, T>. Deletion of any w ∈ (A - {u,v}) that belongs to the vertex set of Ç $N(P_r)$ from <A,T> cuts precisely those u → v paths in <A, T> that belong to S.

(b) Let S = {P_0, ..., P_n} be a maximal interdependent set of u — v paths in <A, E>. Deletion of any w ∈ (A - {u,v}) that belongs to the vertex set of Ç $N(P_r)$ from <A, E> cuts precisely those u — v paths in <A, E> that belong to S. ♦

Proof:

(a) for <A, T>: see [GVS99].

(b) for <A, E>: From theorem 5.1 we know that deletion of w cuts all the P_r ∈ S. Assume that deletion of w from <A, E> cuts at least one u — v path P ∉ S. Then w is between u and v on P, so w belongs to the vertex set of N(P). But then, since w also belongs to the vertex set of every $N(P_r)$ with P_r ∈ S, S is not a maximal interdependent set because the vertex set of (Ç $N(P_r)$) ∩ N(P) contains {u,v,w}. The theorem follows. ♦

Theorem 5.3: (see theorem 12.8, p. 205 of [GVS99])

(a) The set of all u → v paths, in <A, T>, that are cut by the deletion of w ∈ (A - {u,v}) from <A, T> is an interdependent set of u → v paths in <A,T>, but it is not necessarily maximal.

(b) The set of all u — v paths, in <A, E>, that are cut by the deletion of w ∈ (A - {u,v}) from <A, E> is an interdependent set of u — v paths in <A, E>, but it is not necessarily maximal. ♦

Proof:

(a) for <A, T>: see [GVS99].

(b) for <A, E>: Let S = {P_0, ..., P_n} be the set of all u — v paths, in <A, E>, that are cut by the deletion of a given w ∈ (A - {u,v}) from <A, E>. Then w is between u and v on every P_r ∈ S, and hence w belongs to the vertex set of every $N(P_r)$, P_r ∈ S. It follows that Ç $N(P_r)$ has at least one vertex w, other than u and v, in it, and hence S is an interdependent set. It is clear that S is not necessarily maximal. ♦

Just as for relation nets – see p. 206 of [GVS99] – it is always possible to partition the set of all u — v paths in a hypernet <A, E> by the following procedure.

(1) Start with any u — v path P_{00}, and develop a maximal interdependent set of u — v paths S_0 = {P_{0k}| k = 0, 1, ... , n_0} in <A, E> to which P_{00} belongs.

(2) Delete any $w_0 \in (A - \{u,v\})$ such that w_0 belongs to the vertex set of Ç $N(P_{0r})$, $r = 0, 1, 2, ..., n_0$ from <A, E>. This cuts all the u — v paths of S_0, and only those u — v paths.
(3) Start with any u — v path P_{10} in the sub-hypernet that results when w_0 is deleted from <A, E>, i.e. $< A - \{w_0\}, E\uparrow(A - \{w_0\})>$, and develop a maximal interdependent set $S_1 = \{ P_{1k} | k = 0, 1, ..., n_1\}$ of u — v paths, in $<A - \{w_0\}, E\uparrow(A - \{w_0\})>$, to which P_{10} belongs.
(4) Delete any $w_1 \in (A - \{u, v, w_0\})$ such that w_1 belongs to the vertex set of Ç $N(P_{1r})$, $r = 0, 1, 2, .., n_1$ from $< (A - \{w_0\}), E\uparrow(A - \{w_0\})>$. This cuts precisely those u — v paths that belong to S_1. Further, w_0 is not between u and v on any P_{1i}, $i = 0, 1, 2, ..., n_1$.
(5) Continuing in this way we get a partition $\{S_0, ..., S_n\}$ of the set of all u — v paths in <A, E> such that each S_r, $r = 0, 1, 2, ..., n$, is a maximal interdependent set of u — v paths in $< (A - \{w_0, ..., w_{r-1}\}), E\uparrow(A - \{w_0, ..., w_{r-1}\})>$, $r = 0, 1, 2, ..., n$, and S_0 is a maximal interdependent set of u — v paths in <A, E>. ♦

To see that such a partition is well defined we notice that every u — v path in <A, E> will belong to at least one S_r, and that if a particular u — v path P belongs to both. S_r and S_t with $r < t$, then it is a path in the sub-hypernet
$<A - \{w_0, ..., w_{r-1}, w_r, ..., w_{t-1}\}, E\uparrow(A - \{w_0, ..., w_{r-1}, w_r, ..., w_{t-1}\})>$
which is impossible because, since $P \in S_r$, we have w_r between u and v on every member of S_r and hence on P.

Definition 5.3:
(a) A subset $B(u \to v) \subseteq A$ of <A, T> is called a ***separation*** for u and v in <A, T> iff $< A - B(u \to v), T\uparrow (A - B(u \to v))>$, i.e.the maximum subnet of <A, T> that has vertex set $A - B(u \to v)$, has no $u \to v$ paths.
(b) A subset $B(u — v) \subseteq A$ of <A, E> is called a ***separation*** for u and v in <A, E > iff $< A - B(u — v), E\uparrow(A - B(u — v))>$ has no u — v paths. ♦

Theorem 5.4: (see theorem 12.9, p. 206 of [GVS99])
(a) If $\{S_0, ..., S_m\}$ is a partition of the set of all $u \to v$ paths in <A, T> such that S_0 is a maximal interdependent set of $u \to v$ paths in <A, T> and, for each $r = 0, 1, ..., m$, S_r is a maximal interdependent set of $u \to v$ paths in $<A - \{w_0, ..., w_{r-1}\}, T\uparrow(A - \{w_0, ..., w_{r-1}\})>$, where w_0 belongs to the vertex set of Ç $N(P_t)$ over $P_t \in S_o$ and w_r belongs to the vertex set of Ç $N(P_t)$ over $P_t \in S_r$, then there exists a separation $B(u \to v)$ for u and v in <A, T> that has precisely m elements.
(b) If $\{S_0, ..., S_m\}$ is a partition of the set of all u — v paths in <A, E> such that S_0 is a maximal interdependent set of u — v paths in <A, E> and, for each $r = 0, 1, ..., m$, S_r is a maximal interdependent set of u — v paths in $<A -\{ w_0, ..., w_{r-1}\}, E\uparrow(A -\{ w_0, ..., w_{r-1}\})>$, where w_0 belongs to the vertex set of Ç $N(P_t)$ over $P_t \in S_o$ and w_r belongs to the vertex set of Ç $N(P_t)$ over $P_t \in S_r$, then there exists a separation B(u — v) for u and v in <A, E> that has precisely m elements. ♦

Proof: See [GVS99]. Proof follows at once from the partitioning and previous theorems and definitions. ♦

Corollary 5.1: (Corollary 12.1, p. 207 of GVS99])

(a) The minimum number of elements in a partition of the u $\rightarrow$ v paths in <A, T> into maximal interdependent sets, constructed as in Theorem 5.4, is equal to the minimum number of vertices in a separation B(u $\rightarrow$v) for u and v in <A, T>.

(b) The minimum number of elements in a partition of the u — v paths in <A, E> into maximal interdependent sets, constructed as in Theorem 5.4, is equal to the minimum number of vertices in a separation B(u — v) for u and v in <A, E>.♦

Corollary 5.2: (Corollary 12.2, p. 207 of [GVS99])

(a) Any separation for u and v in <A, T> can be used to generate a partition of the set of all u $\rightarrow$ v paths in <A, T> into interdependent sets which are not necessarily maximal.

(b) Any separation for u and v in <A, E> can be used to generate a partition of the set of all u — v paths in <A, E> into interdependent sets which are not necessarily maximal. ♦

Proof:

(a) for <A, T>: see [GVS99].

(b) for <A, E>: Suppose that we are given a separation B(u — v) = $\{w_0, ..., w_m\}$. Let S_0 be the set of all u — v paths in <A, E> that are cut by the deletion of w_0 from <A, E>. Next let S_1 be the set of all u — v paths in <A - $\{w_0\}$, E$\uparrow$(A - $\{w_0\}$)> that are cut by the deletion of w_1 from <A - $\{w_0\}$, E$\uparrow$(A - $\{w_0\}$)>. Then let S_2 be the set of all u — v paths in <A - $\{w_0, w_1\}$, E$\uparrow$(A - $\{w_0, w_1\}$)> that are cut by the deletion of w_2 from <A - $\{w_0, w_1\}$, E$\uparrow$(A - $\{w_0, w_1\}$)>.

Proceeding in this way we develop sets S_0, ..., S_m. It is clear that each S_r, r = 0, 1, ..., m, is an interdependent set of u — v paths, and if P is an arbitrary u — v path in <A, E> then at least one of w_0, ..., w_m is between u and v on P, so P belongs to at least one of the S_r, r = 0, 1, ..., m. As we showed before, it is impossible for P to belong to more than one S_r, so the corollary follows because it is clear that the S_r are not necessarily maximal. ♦

Defintion 5.4:

(a) Let P_r and P_t be u $\rightarrow$ v paths in <A, T>, where u $\neq$ v and the underlying sets of both N(P_r) and N(P_t) strictly contain {u,v}. P_r and P_t are said to be ***quasi-disjoint*** u $\rightarrow$ v paths in <A, T> iff they belong to distinct maximal interdependent sets of u $\rightarrow$ v paths in <A, T>.

(b) Let P_r and P_t be of u — v paths in <A, E>, where u $\neq$ v and the underlying sets of both N(P_r) and N(P_t) strictly contain {u,v}. P_r and P_t are said to be ***quasi-disjoint*** u — v paths in <A, E> iff they belong to distinct maximal interdependent sets of u — v paths in <A, E>. ♦

We can now restate corollary 5.1 in Mengerian form.

Corollary 5.3:

(a) The maximum number of pairwise quasi-disjoint u $\rightarrow$ v paths in <A, T> is equal to min |B(u $\rightarrow$ v)|.

(b) The maximum number of pairwise quasi-disjoint u — v paths in <A, E> is equal to min |B(u — v)|. ♦

Proof:

(a) for <A, T>: see p. 207/208 of [GVS99].

(b) for <A, E>: Assume that we have achieved a partition of the $u — v$ paths in <A, E> into min $|B(u \to v)|$ maximal interdependent sets as referred to in Corollary 5.1, and that $B(u — v)$ is one of the corresponding separations. How many pairwise quasi-disjoint $u — v$ paths can we find in <A, E>? Certainly we can find at least min $|B(u \to v)|$ such paths, each in a distinct member of the partition, and each thus cut by a unique member of $B(u — v)$, since if deletion of a given $b \in B(u — v)$ cuts more than one of these paths then those paths cut are not pairwise quasi-disjoint paths. Further, we cannot find more than min $|B(u \to v)|$ such paths, because in that case at least two of them must belong to the same maximal interdependent set of the partition, which violates the condition that they should be quasi-disjoint $u — v$ paths. It follows that min $|B(u \to v)|$ equals the minimum number of elements of a partition of the $u — v$ paths in <A, E> into maximal interdependent sets, constructed as in theorem 5.4, which, in turn, is equal to the maximum number of pairwise quasi-disjoint $u — v$ paths in <A, E>.♦

Menger’s theorem is important because examining “flow” through a hypernet can contribute to analysis of its structure. We will return to this point for a special kind of hypernet in a later section.

6. Connectedness

Definition 6.1: A hypernet <A, E> is said to be ***connected*** iff for every a, b ∈ A there is at least one path a — b in <A, E>.♦

Theorem 6.1: A hypernet <A, E> is connected iff it has a closed spanning walk, i.e. a walk that meets every a ∈ A at least once or, in other words, a walk in which every a ∈ A occurs at least once in the subsequence over A. ♦

Proof: trivial. ♦

Definition 6.2: A sub-hypernet <B, U> of a hypernet <A, E> is called a ***component*** of <A, E> iff it is a maximal connected sub-hypernet of <A, E>, where by maximal we mean that to add any a ∈ (A - B) or any E_i ∈ (E-U) to <B, U> will result in a sub-hypernet of <A, E> that is not connected. ♦

Theorem 6.2: If <B_0, U_0> and <B_1, U_1> are distinct components of a hypernet <A, E> then B_0 and B_1 are disjoint, i.e. $B_0 \cap B_1 = \emptyset$.♦

Proof: Suppose that b ∈ $B_0 \cap B_1$, and let a ∈ B_0 and c∈ B_1. Then there is a path a — b in <B_0, U_0> and a path b — c in <B_1, U_1>, so there is a path from any vertex in <B_0, U_0> to any vertex in <B_1, U_1>, which means that <B_0, U_0> ∪ <B_1, U_1> lies in a single component of <A, E>. The theorem follows. ♦

Theorem 6.3: Let <A, E> be any hypernet. Then
(1) every a ∈ A belongs to precisely one component of <A, E> and
(2) every vertex adjacency, and hence also every edge E_i, belongs to at most one component. ♦

Proof:
(1) Assume that a ∈ A belongs to two distict components of <A, E>. Then, as in the proof of theorem 6.2 above, we reach a contradiction.
(2) Suppose that vertex adjacency (a, E_i, b) is such that a is in a component <B_0, U_0> of <A, E> and that b is in a distinct component <B_1, U_1> of <A, E>. Then it is easy to see that since every vertex in <B_0, U_0> is reachable from a, and every vertex in <B_1, U_1> is reachable from b, every vertex in <B_0, U_0> is reachable from every vertex in <B_1, U_1>. The theorem follows from this contradiction. ♦

Theorem 6.4: The distinct components of a hypernet <A, E> induce an equivalence relation on A. ♦

Proof: It is easy to see that reachability is reflexive, as we regard each vertex as reachable from itself by a path of length zero, symmetric and transitive. ♦

It follows immediately from theorem 6.4 that

Corollary 6.1: Reachability in a hypernet <A, E> partitions A into equivalence classes that are precisely the vertex sets of the components of <A, E>.♦

7. Vertex bases

Definition 7.1: A ***vertex basis*** for a hypernet <A, E> is a set V ⊆ A such that every a ∈ A is reachable from at least one v ∈ V, and V is minimal in the sense that no proper subset of V has this property. ♦

Theorem 7.1: Every a ∈ A of a hypernet <A, E>, that has only a loop incident with it or is an isolate or a complete isolate in <A, E>, belongs to every vertex basis of <A, E>.♦

Proof :Follows from the fact that no such vertex is reachable from any vertex but itself. ♦

Theorem 7.2: V ⊆ A of a hypernet <A, E> is a vertex basis of <A, E> iff
(1) every a ∈ A is reachable from at least one v ∈ V, i.e. ℜ(V) = A, and
(2) no v ∈ V ⊆ A is reachable from any w ≠ v, w ∈ V, in <A, E>. ♦

Proof: We need only show that (ii) is equivalent to minimality of V. Suppose that V is a vertex basis of <A, E> and that w, v ∈ V and that w and v are mutually reachable in <A, E>. Then every a ∈ A that is reachable from v is also reachable from w, so v is not necessary in V, i.e.V is not minimal. The theorem follows. ♦

Corollary 7.1: No two members of V lie in the same component of <A, E>. ♦

Proof: Follows from the definitions of vertex basis and of component. ♦

Corollary 7.2: Every hypernet <A, E> has at least one vertex basis. ♦

Proof: A certainly fulfills condition (1) of theorem 7.2, so we can find at least one V ⊆ A that fulfills condition (2) as well. ♦

Theorem 7.3: If V ⊆ A is a vertex basis of a hypernet <A, E> then there is precisely one v ∈ V in each component of <A, E>, and | V | is precisely the number of components of <A, E>.♦

Proof: Follows at once from the definition of component as we only need one vertex from each component to reach every a ∈ A. Suppose that v, w ∈ V lie in the same component of <A, E>. Then it is clear that we do not need both v and w in a vertex basis. V is not minimal, contradicting the given fact that V is a vertex basis of <A, E>.♦

Theorem 7.4: If <B, U> ∠ <A, E> then every vertex basis of <A, E> contains a vertex basis of <B, U>. ♦

Proof: Let V ⊆ A be any vertex basis of the hypernet <A, E>. Then every a ∈ A is reachable from some one vertex v ∈ V. Since <B, U> ∠ <A, E> it is clear that every vertex b ∈ B ⊆ A is reachable from at least one $v \in V$ in <B, U>. Thus we can find a vertex basis of <B, U> inside V by applying the minimality condition to V inside <B, U>. ♦

Theorem 7.5: If hypernet <A, E> has no non-loop circuits and we ignore all closed paths of length 2, i.e. that use two edges from the same label, then <A, E> has a unique vertex basis that consists of precisely those $a \in A$ at which there is only a loop or which are isolates or complete isolates. ♦

Proof: Let $V_1 \subseteq A$ of <A, E> be the set of all $a \in A$ at which there is only a loop or are isolates or complete isolates in <A, E>, and let $V_2 \subseteq A$ be any vertex basis of <A, E>. By theorem 7.1, $V_1 \subseteq V_2$. Now suppose that V_2 is not included or equal to V_1 , i.e $V = V_1 - V_2 \neq \phi$. Let $v \in V$. Then v must be reachable from at least one $a \in V_2$ because V_2 is a vertex basis of <A,E>. But $v \in V_1$, so v is only reachable from itself. It follows that $V_2 \subseteq V_1$, and thus $V_1 = V_2$. Finally, because <A,E> has no non-loop circuits and we ignore all closed paths of length 2, i.e. they may exist but we never traverse them, we will never be faced with the possiblity of choosing any member of a circuit as the relevant member of a vertex basis for <A, E>, so $V_2 = V_1$ is a unique vertex basis for <A,E>.♦

Theorem 7.6: Given $a \in A$ of a hypernet <A, E>, the hypernet < $\Re(a)$, $E\uparrow(\Re(a))$>, i.e. the maximum sub-hypernet of <A, E> that is generated by $\Re(a)$, is connected. ♦

Proof: Every $b \in \Re(a)$ is reachable from a, and every $E_i \in E\uparrow(\Re(a))$ is a subset of $\Re(a)$. The theorem follows. ♦

We close that section with a few observations. Given a hypernet <A, E>, let $U_1 \subseteq U_2 \subseteq E$. Then

(1) for all $a \in A$, d(a) in <A, U_1> $\leq$ d(a) in <A, U_2> $\leq$ d(a) in <A, E>.
(2) For all $a, b \in A$, if b is reachable from a in <A, U_2> then it is reachable from a also in <A, U_2> and in <A, E>.
(3) For all $a, b \in A$, if a is adjacent to b in <A, U_1> then it is also adjacent to b in <A, U_2> and in <A, E>.
(4) If <A, U_1> is connected then so are <A, U_2> and <A, E>.
(5) Every component of <A, U_1> is a connected sub-hypernet of a component of <A, U_2> which is, in turn, a connected sub-hypernet of a component of <A, E>.
(6) If <A, E> has no circuits then <A, U_2> has no circuits, and if <A, U_2> has no circuits then <A, U_1> has none.
(7) Every vertex basis of <A, U_1> contains a vertex basis of <A, U_2>, which, in turn, contains a vertex basis of <A, E>.

8. Introduction to Vulnerability

Definition 8.1: Let a, b $\in$ A of a hypernet <A, E>. We say that a and b are ***joined*** in <A, E> iff there is at least one path a — b in <A, E>. Otherwise a and b are said to be ***non-joined*** in <A, E>.♦

Definition 8.2: Let a, b $\in$ A of a hypernet <A, E>, a $\neq$ b, and consider any edge $E_i \in$ E. E_i is said to be ***between*** a and b in <A, E>, written (a - E_i - b), iff a and b are joined in <A, E> and every path a — b in <A, E> ***goes via*** E_i, i.e. E_i is a member of the edge subsequence of every path a — b in <A, E>. ♦

Note that we have defined "between" for vertices in a similar fashion – see definition 4.5 (i) .

Theorem 8.1: Let a, b $\in$ A of a hypernet <A, E>, a $\neq$ b, and let $E_i \in$ E. We have (a - E_i - b) iff a and b are joined in <A, E> and every path a — b in <A, E> is such that at least one vertex adjacency by E_i is a subsequence, of length 1, of a — b. ♦

Proof: If a — b goes via E_i then there must be at least one vertex adjacency by E_i in a — b. ♦

Corollary 8.1: If a and b of the theorem are adjacent vertices then $\lambda(\{a, b\}) = \{E_i\}$.♦

Corollary 8.2: If (a - E_i - b) then deletion of E_i from <A, E> deletes all a — b paths in <A, E>.♦

Note that deleting the vertex adjacency (a, E_i, b) does not necessarily mean that a and b are no longer adjacent: We may have $\{E_i\} \subset \lambda(\{a,b\})$.

Let C_1 be the class of connected hypernets and C_0 be the class of non-connected, i.e. disconnected, hypernets.

Definition 8.3: Let <A, E> be a hypernet with $E_i \in$ E. We write $E^c{}_i$ for E - $\{E_i\}$. We call E_i an ***(x, y)- edge*** of <A, E> iff <A, E> is in C_x and <A, $E^c{}_i$ > is in C_y. E_i is said to be a ***strengthening edge*** of <A, E> iff E_i is (x, y) with x > y, and a ***neutral edge*** of <A, E> iff x = y. ♦

Theorem 8.2: There is no (0,1)-edge in any hypernet. ♦

Proof: Every path in <A, $E^c{}_i$ > is also in <A, E>, so the connected class of <A, $E^c{}_i$ > is at most that of <A, E>, i.e. deleting E_i from <A, E> can not increase the connectedness of <A, E>.♦

At once, from theorem 8.2, there follows

Corollary 8.3: Every $E_i \in$ E of a disconnected hypernet <A, E> is a (0,0)-edge, i.e. is neutral. ♦

Theorem 8.3: Let $E_i \in E$ of any hypernet <A, E>. Suppose that <A, E> is in C_1. Then <A, $E^c{}_i$ > is in C_0 iff every (closed) spanning walk in <A, E> goes via E_i. ♦

Proof: By theorem 6.1 <A, E> is connected iff <A, E> has a (closed) spanning walk. If every spanning walk goes via E_i then deletion of E_i from <A, E> leaves no spanning walk in <A, $E^c{}_i$ >, so <A, $E^c{}_i$ > is in C_0. If <A, $E^c{}_i$ > is in C_0 then every (closed) spanning walk in <A, E>, which is given to be in C_1, must go via E_i. ♦

Definition 8.4: Let $E_i \in E$ be an edge of a connected hypernet <A, E>. E_i is called a ***bridge*** iff there exist a, b ∈ A with (a - E_i - b). ♦

Theorem 8.4: $E_i \in E$ of a connected hypernet <A, E> is a bridge in <A, E> iff E_i is a (1, 0)-edge. ♦

Proof: If E_i is a bridge then (a - E_i - b) for some a, b in <A, E>. Thus a and b are joined in <A, E>, and if we delete E_i from <A, E> then a and b are non-joined in <A, $E^c{}_i$ > so a and b lie in different components in <A, $E^c{}_i$ > and thus <A, $E^c{}_i$ > is in C_0, and hence E_i is a (1, 0)-edge. If E_i is a (1, 0)-edge then there must exist a, b ∈ A that are joined in <A, E> but non-joined in <A, $E^c{}_i$ >, so we must have (a - E_i - b), i.e. E_i is a bridge, in <A, E>. ♦

Theorem 8.5: If $E_i \in E$ of a connected hypernet <A, E> is a bridge in <A, E> then every subset $U \subseteq E$ of edges with $E_i \in U$ is a disconnecting set of edges in <A, E>, i.e. <A, E – U> is disconnected. ♦

The proof follows at once from the fact that $E_i \in U$ is a bridge in <A, E>. Furthermore, it follows from the definition of a bridge and theorem 8.4 that we have

Theorem 8.6: Every strengthening edge, i.e. (1, 0)-edge, in a connected hypernet <A, E> is a bridge in <A, E>.♦

Theorem 8.7: Let hypernet <A, E> be in C_1, and let $E_i \in E$. Then E_i is (1, 1) in <A, E> iff E_i is not a bridge in <A, E>.♦

Proof: If E_i is a (1, 1)-edge then it is not a bridge in <A, E>, by the definition of a bridge. If E_i is not a bridge in <A, E> then E_i can only be a (1,1)-edge in <A, E> since it cannot be a (0, 1)-edge by theorem 8.2. ♦

Theorem 8.8: Let <A, E> be a hypernet with $E_i \in Q \subseteq E$.
(1) If E_i is a bridge in <A, E>, and <A, Q> is in C_1, then E_i is a bridge in <A, Q>.
(2) If E_i is strengthening in <A, E> then E_i is strengthening or neutral in <A, Q>.♦

Proof:

(1) E_i is a bridge in <A, E> but <A, Q> is connected, so deletion of E_i from <A, Q> must disconnect <A, Q> and so E_i must be a (1, 0)-edge, i.e. a bridge, in <A, Q> since <A, Q> $\angle$ <A, E> and both are connected.

(2) E_i is strengthening in <A, E>, i.e. it is a (1, 0)-edge in <A, E>, so it is a bridge in <A, E>. Now if <A, Q> is in C_1 then E_i is strengthening, i.e. a bridge, in <A, Q> by part (i). If <A, Q> is in C_0 then, since there is no (0, 1)-edge in any hypernet, E_i must be neutral, i.e. a (0, 0)-edge, in <A, Q>. ♦

Corollary 8.4: If $E_i \in E$ of a hypernet <A, E> with $E_i \in Q \subseteq E$, and if E_i is a (1, 1)-edge in <A, Q>, then E_i is a (1, 1)-edge in <A, E>.♦

Proof: Follows since both <A, Q> and <A, E> are in C_1, and because E_i cannot be a (0, 1)-edge in any hypernet, E_i must be a (1, 1)-edge in <A, E>.♦

Corollary 8.5: Let <A, E> be a hypernet with $E_i \in Q \subseteq E$. Let E_i be a (1, 0)-edge in <A, Q> and let <A, E> be in C_1. If whenever E_i is between vertices a and b in <A, Q> there is a path a — b in <A, E> that is not in <A, Q>, then E_i is neutral in <A, E>. The converse is also true. Next, if E_i is between a and b in <A, Q>, and there is no path a — b in <A, E> that is not in <A, Q>, then E_i is a (1, 0)-edge in <A, E>.♦

Proof: Both <A, Q> and <A, E> are in C_1, and E_i is a bridge in <A, Q>. Thus there exist a, b $\in$ A such that E_i is between a and b in <A, Q>, i.e. every path a — b in <A, Q> goes via E_i. Now if there is at least one path a — b in <A, E> that does not go via E_i, then E_i is not between a and b in <A, E> so E_i is a (1, 1)-edge in <A, E>, i.e neutral in <A, E>. If E_i is neutral in <A, E> but a bridge in <A, Q>, and both <A, E> and <A, Q> are in C_1, then there exist a, b $\in$ A such that E_i is not between a and b in <A, E>, i.e. E_i is neutral in <A, E>, but (a - E_i - b) in <A, Q>. Thus there is at least one path a — b in <A, E> that does not go via E_i whenever we have (a - E_i - b) in <A, Q>. Finally, if (a - E_i - b) in <A, Q>, i.e. E_i is a bridge in <A, Q>, and there is no a — b path in <A, Q^c >, that is not in <A, Q>, then deletion of E_i from <A, E> disconnects <A, E>, i.e. E_i is a bridge in <A, E>, because every a — b path in <A, E> is in <A, Q>, and all such a — b paths go via E_i. ♦

9. Edge bases

Definition 9.1: Let <A, E> be any hypernet with $B \subseteq E$. B is called an ***edge basis*** of <A, E> iff for all $a, b \in A$ we have $a \in \Re(b)$ iff $a \in \Re_B(b)$, where $\Re_B(b)$ is the reachability function of <A, B>, and no proper subset of B has this property. ♦

Theorem 9.1: $E_i \in E$ of a hypernet <A, E> is between a and b in <A, E>, $a, b \in A$, i.e. (a - E_i - b), iff E_i belongs to every edge basis of <A, E>. ♦

Proof: If (a - E_i - b) then we can only get $a \in \Re(b)$ in <A, E> by having E_i in every edge basis of <A, E>. If E_i belongs to every edge basis of <A, E> then there must exist $a, b \in A$ such that $a \in \Re(b)$ and every path a — b goes via E_i, so (a - E_i - b). ♦

Theorem 9.2: If for $a, b \in A$ of a hypernet <A, E> there is a unique path a — b in <A, E> then $\{E_i \in E \mid a — b \text{ goes via } E_i\}$ is contained in every edge basis of <A, E>. ♦

Proof: Every E_i via which a — b goes is such that (a - E_i - b), so by theorem 9.1 each such E_i belongs to every edge basis of <A, E>. ♦

Theorem 9.3: Let <A, E> be any hypernet and let $B \subseteq E$. B is an edge basis of <A, E> iff
(1) B preserves reachability in <A, E> and
(2) for every $E_i \in B$ there exist $a, b \in A$ with (a - E_i - b). ♦

Proof: Preservation of reachability is one part of the definition of an edge basis. We must show that (2) is equivalent to minimality of B. Suppose that there is an edge $E_j \in B$ for which there exist no $a, b \in A$ with (a - E_j - b). Then we can preserve the reachabilty of a from b without E_j, so we do not need E_j in B, i.e. a proper subset $(B - \{E_j\}) \subseteq B$ will preserve reachability, so B is not an edge basis. ♦

Theorem 9.4: $B \subseteq E$ is an edge basis of a connected hypernet <A, E> iff <A, B> is a minimal connected sub-hypernet of <A, E>, i.e. there is no connected sub-hypernet <A, D> with $D \subset B$. ♦

Proof: Let B be an edge basis of <A, E>. For every $E_i \in B$ there exist $a, b \in A$ with (a - E_i - b), and since <A, E> is connected E_i is a bridge in <A, E>. So we cannot leave any $E_i \in B$ out of B because we would then be left with a disconnected hypernet <A, B - { E_i}>. Thus <A, B> is minimal and it is connected because B preserves reachability in the connected hypernet <A, E>. Conversely, if <A, B> is a connected sub-hypernet of <A, E> then B must preserve reachability in <A, E>. Since <A, B> is minimal, B is a minimal set of edges that preserves reachability in <A, E>, so B is an edge basis of <A, E>. ♦

Theorem 9.5: Let <A, E> be any hypernet. If W is a closed spanning walk of minimal length in <A, E> then $Q = \{ E_i \in E \mid W \text{ goes via } E_i\} \subseteq E$ contains an edge basis of <A, E>. ♦

Proof: If W is a closed spanning walk in <A, E> then <A, E> is connected. If W has minimal length then Q certainly preserves reachability in <A, E>, so Q must contain at least one edge basis of <A, E>.♦

Theorem 9.6: To find an edge basis for a hypernet <A, E> we may use the following constructional scheme. Let D be the set of all vertex adjacencies of all a and b, $a \neq b$ and $a, b \in A$. Each such vertex adjacency has one or more $E_i \in E$ in $l(\{a, b\})$, for each of which we have (a, E_i, b), so $\{a, b\} \subseteq E_i$.

(1) Define a bipartite graph with vertex sets V_1 and V_2 where $V_1 = \{\{a, b\} \in \wp(A) \mid$ a and b are adjacent vertices in <A, E>$\} = D$ and $V_2 = E$, and set $V = V_1 \cup V_2$ for that bipartite graph. Join each $\{a, b\} \in V_1$ to each $E_i \in V_2 = E$ for which (a, E_i, b), using an unoriented edge. These are all the vertices and edges of our bipartite graph. Let $V_2 = E = L^{(1)}$ and set $L^{(2)} = \varnothing$ and $L'^{(2)} = \varnothing$ for future use.

(2) Choose any $r \in V_1$ that has degree $d(r) = 1$ in our bipartite graph. If there are no such vertices in V_1 then proceed to (3) with $L'^{(2)} = \varnothing$. If there is such a vertex, addend that vertex $s \in V_2$ that is adjacent to r, in our bipartite graph, to $L^{(2)} \subset L^{(1)}$. Next addend the vertex $t \in V_1$ that is adjacent to s in our graph to $L'^{(2)} \subset V_1$. Now remove $L^{(2)} \cup L'^{(2)}$ from V. Repeat (2) until $V_1 - L'^{(2)} = \varnothing$, in which case we have found a set of edges in E that “covers” all the vertex adjacencies in <A, E>, and which contains at least one edge basis of <A, E>, or until no more vertices with degree 1 remain in $V_1 - L'^{(2)}$. In the latter case, proceed to (3).

(3) Choose any $r \in V_1 - L'^{(2)}$ that has $d(r) = 2$ in our graph. If there is no such vertex then proceed to (4) with $L'^{(2)}$ as it is at the end of step (2). If there is such a vertex r, choose any s adjacent to r in our graph. Addend s to $L^{(2)}$, and addend the vertex adjacent to s in our graph to $L'^{(2)}$. Remove $L^{(2)} \cup L'^{(2)}$ from V. Repeat (3) until $V_1 - L'^{(2)} = \varnothing$, in which case we have found a set of edges in E that ”covers” all the vertex adjacencies in <A, E>, and which contains at least one edge basis of <A, E>, or until no more vertices of degree 2 remain in $V_1 - L'^{(2)}$. In the latter case, proceed to (4).

(4) Repeat (3) successively with vertices $r \in V_1 - L'^{(2)}$ that have degree 3, 4, ... Eventually $V_1 = L'^{(2)}$ and at that stage $L^{(2)}$ is such that $\{E_i \in E \mid E_i \in L^{(2)}\}$ contains at least one edge basis of <A, E>, and $|L^{(2)}| \leq |E|$.

End of stage 1. ♦

Proof of stage 1: It is clear that $L^{(2)}$ contains at least one edge basis of <A, E> at this stage because $L^{(2)}$ “covers” every vertex adjacency in <A, E>. That $|L^{(2)}| \leq |E|$ follows from the fact that every $E_i \in L^{(2)}$ “covers” one “new” vertex adjacency. Further, $L^{(2)}$ is a minimal set of edges that “covers” every vertex adjacency in <A, E>, because each $E_i \in L^{(2)} \subseteq E$ covers a vertex adjacency by E_i. ♦

(5) Examine $L^{(2)}$ as follows. Find an $E_i \in L^{(2)}$ that satisfies the following condition: For all $a, b \in A$, whenever there is a path a — b via E_i in <A, E> there is also a path a — b in <A, E> that goes via members of a subset of $L^{(2)} - \{E_i\}$ only. If there is no such $E_i \in L^{(2)}$ then $\{E_i \in E \mid E_i \in L^{(2)}\}$ is an edge basis of <A, E>. If there is such an E_i, set $L^{(3)} = L^{(2)} - \{E_i\}$. Repeat the test on the members of $L^{(3)}$. Either $\{E_i \in E \mid E_i \in L^{(3)}\}$ is an edge basis for <A, E> or we define $L^{(4)} = L^{(3)} - \{E_i\}$ for some $E_i \in L^{(3)}$.

Proceeding in this way we find an $L^{(n)}$ that is one of the edge bases of <A, E> for some natural number n with $n \leq |E|$.

End of stage 2. ♦

Proof of stage 2: To show that $L^{(n)} \subseteq L^{(2)}$ is an edge basis for <A, E> we must prove that $E_i \in L^{(2)}$ necessarily belongs to an edge basis of <A, E> iff there exist a, b ∈ A such that there is at least one path a — b in <A, E> that goes via E_i and that no path a — b in <A, E> goes via any non-empty subset of $L^{(2)} – \{E_i\}$. First, if there is at least one path a — b in <A, E> that goes via E_i, and no path a — b in <A, E> goes via any non-empty subset of $L^{(2)} – \{E_i\}$, then removal of E_i from $L^{(2)}$ means that a is not reachable from b in <A, $L^{(2)} – \{ E_i\}$>, so $L^{(2)} – \{E_i\}$ does not contain an edge basis of <A, E>. But $L^{(2)}$ does contain at least one edge basis of <A, E>, so E_i must belong to every edge basis of <A, E> that is contained in $L^{(2)}$. Conversely, if for all a, b ∈ A such that there is at least one path a — b via $E_i \in L^{(2)}$ in <A, E> there is a path a — b in <A, $L^{(2)}$ - $\{E_i\}$> then $L^2 – \{E_i\}$ contains at least one edge basis of <A, E>, and so E_i does not necessarily belong to an edge basis $B \subseteq L^{(2)}$. Thus we have the correct criterion for rejecting an $E_i \in L^{(2)}$. ♦

To close this section we return to theorem 9.5.

Definition 9.2: Let <A, E> be a connected hypernet. A ***connectedness preserving set of edges*** of <A, E> is a set $Q \subseteq E$ which is such that <A, Q> is connected. ♦

How can we find a minimal connectedness preserving set $Q \subseteq E$ in <A, E>?

Theorem 9.7: Let <A, E> be a connected hypernet. W is a spanning walk of minimal length in <A, E> iff $E_W = \{ E_i \in E \mid W \text{ goes via } E_i\}$ is a minimal set of edges that preserves the connectednes of <A, E>. ♦

Proof: If W is a spanning walk of minimal length in the connected hypernet <A, E> then every E_i such that W goes via E_i is needed to preserve the connectedness of <A, E>. Conversely, if $E' \subseteq E$ is a minimal connectedness preserving set of edges for <A, E> then, since <A, E> is connected, it has at least one spanning walk, and at least one of these spanning walks will use all, and only, the members of E'. Since E' is minimal, such a spanning walk will be of minimal length $|E'|$. ♦

10. Deletion of vertices

We open this chapter with a comment in the form of a lemma. Let <A,E> be any hypernet, and let B ⊆ A. Consider the following sub-hypernets of <A,E>:

- <B, E↑B> - see definition 2.15,
- <A,E(B)> - see definition 4.1, (1),
- <A,E[B]> - see definition 4.1, (2),
- <A,E>[B] - see definition 4.8.

If B = A then all but possibly the second are precisely <A,E>. We see, from the definitions, that E ↑B ⊆ E (B) ⊆ E [B].

Lemma 10.1:
(1) <A, E (B)> ∠ <A, E [B]>.
(2) <B, E↑B> ∠ <A, E>[B] ∠ <A,E[B]>.♦

Proof:
(1) To construct <A, E [B]> from <A, E(B)> we must add zero or more edges to <A, E(B)>.
(2) First notice that the context hypernet <A, E> [B] has vertex set at least B. To construct <A, E>[B] from <B, E↑B> we must add zero or more vertices to B, and also zero or more edges to E ↑ B. Next notice that <A, E> [B] has edge set E [B], so to construct <A, E [B]> from <A, E> [B] we must add zero or more vertices. ♦

Next we recall definition 4.5 (1): If a, b, c ∈ A of a hypernet <A, E>, then b is said to be ***vertex between*** a and c, written (a - b - c), iff a and c are joined in <A, E> and b ∈ E_i ∈ E for at least one edge on every path a — c in <A, E>.

Theorem 10.1: Let a, b, c be distinct members of A in a hypernet <A, E>. Then (a - b - c) in <A, E> iff a and c are joined in <A, E> and non-joined in <A - {b}, E↑(A -{b})>.♦

Proof: If we have (a - b - c), so a and c are joined in <A, E>, and we delete b from <A, E> to produce <A - {b}, E↑(A - {b})>, then all paths a — c disappear from <A, E>, so a and c are non-joined in <A - {b}, E↑(A - {b})>. Conversely, if a and c are non-joined in <A - {b}, E↑(A -{b})> but are joined in <A, E>, then joining the context hypernet of b to <A - {b}, E↑(A - {b})> to produce <A, E> must add in a set of at least one path a — c, and b will be between a and c on all those added a — c paths, i.e. we will have (a - b - c) in <A, E>.♦

Definition 10.1: A vertex b ∈ A of a hypernet <A, E> is called a ***cut-vertex*** of <A, E> iff there exist a, c ∈ A such that (a - b - c) in <A, E>.♦

Theorem 10.2: Let <A, E> be a connected hypernet. The following statements are logically equivalent for every b ∈ A:
(1) b is a cut-vertex in <A, E>.
(2) <A - {b}, E↑(A -{b})> is disconnected .

(3) There exists a partition {A1, A2} of A - {b} such that for all a ∈ A1 and all c ∈ A2 we have (a - b - c) in <A, E>.
(4) There exist a, c ∈ A such that (a-b-c) in <A, E>.♦

Proof:
(1) ⇒ (2): If b is a cut-vertex of <A, E> then there exist a, c ∈ A such that (a - b - c) in <A, E>. But then a and c are not joined in <A - {b}, E↑(A -{b})> , so they belong to different components of <A - {b}, E↑(A -{b})>, and hence <A - {b}, E↑(A -{b})> is disconnected.
(2) ⇒ (3): <A - {b}, E↑(A -{b})> is disconnected. Let A1 ⊂ A be the vertex set of a component of <A - {b}, E↑(A -{b})> and A2 be the vertex set of any other component of this hypernet. Let a ∈ A1 and c ∈ A2. Since <A - {b}, E↑(A -{b})> is disconnected there is no path a – c in <A - {b}, E↑(A -{b})> , but since <A, E> is connected there is at least one path a – c in <A, E>, and every such path has b vertex between a and c in <A, E>, so (a - b - c) in <A, E>.
(3) ⇒ (4): Follows at once from (3).
(4) ⇒ (1): Follows at once from the definition of a cut vertex. ♦

Corollary 10.1: Vertex b ∈ A of a connected hypernet <A, E> is a cut-vertex of <A, E> iff <A - {b}, E↑(A -{b})> has more components than <A, E>.♦

Proof : Follows from part (2) of theorem 10.2. ♦

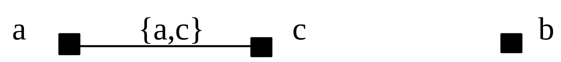
Definition 10.2: Vertex b ∈ A of a hypernet <A, E> is called an ***(x, y)- vertex*** of <A, E> iff <A, E> is in C_x and <A - {b}, E↑(A -{b})> is in C_y. b is called an ***strengthening vertex*** iff x > y, a ***neutral vertex*** iff x = y, and a ***weakening vertex*** iff x < y. ♦

Theorem 10.3: If hypernet <A, E> is in C_x and hypernet <A, E^c(a)> is in C_y, where $E^c(a) = E – E(a)$, then x ≥ y. The theorem also holds for E[a]. ♦

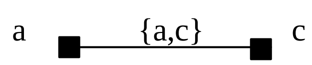
Proof: Follows at once from the fact that deleting the edges E(a) ⊆ E, i.e. the edges in the name of a, from <A, E> to produce <A, E^c(a)> cannot increase the connectedness class of <A, E> as there are no (0,1)- edges in any hypernet. Thus x ≥ y. (See theorem 8.2). ♦

Note in passing that there can exist weakening vertices, i.e. (0, 1)-vertices, in a hypernet. Consider the following simple example

a) <A, E> in C_0:



b) <A - {b}, E↑(A -{b})> in C_1:



Theorem 10.4:
(1) If b ∈ A is a (x, y)-vertex in <A, E^c(b)> then it is a (z, y)-vertex in <A, E> with z ≥ x.
(2) If b ∈ A is a (z, y)-vertex in <A, E> then it is a (x, y)-vertex in <A, E^c(b)> with z ≥ x. ♦

Proof: First notice that deleting b from <A, E^c(b)> yields <A - {b}, E^c [b]>, as does deleting b from <A, E>, and we are given that <A - {b}, E^c [b]> is in C_y.

(1) Starting with <A - {b}, E^c[b]> we get <A, E^c(b)> by adding b and all the edges of E^c(b) - E [b]. The result <A, E^c(b) > is in C_x. To get <A, E> from <A, E^c(b)> we must add all the edges of E – E^c (b), i.e. all the edges of E(b), and we get <A, E> which is in C_z. Now we cannot have $z < x$ because adding edges to a hypernet can only strengthen its connectedness or leave it the same, so z ≥ x.
(2) Starting with <A - {b}, E^c [b]>, which is in C_y, we get <A, E> by adding b and all the edges of E[b], and <A, E> is in C_z. Now to get <A, E^c(b)> from <A, E> we must delete all the edges of E(b). Let the connectedness class of <A, E^c(b)> be C_x. Then by theorem 10.3, z ≥ x. ♦

Corollary 10.2: For a hypernet <A, E> with b ∈ A, the particular cases of the theorem are:

a) b is x, y in <A, E^c(b)> ⇒ b is (z, y) in <A, E> with z ≥ x

1, 1	1, 1
1, 0	1, 0
0, 1	1, 1 or 0, 1
0, 0	1, 0 or 0, 0

b) b is (z, y) in <A, E> ⇒ b is x, y in <A, E^c(b)> with z ≥ x.

1, 1	1, 1 or 0, 1
1, 0	1, 0 or 0, 0
0, 1	0,1
0, 0	0, 0 ♦

Theorem 10.5: Let B ⊆ A be an non-empty set for a hypernet <A, E>, and let B' = A – B. Further let E(B) = (È E(b) for b ∈ B) ⊆ E and E [B] = (È E [b] for b ∈ B) ⊆ E. Then we have

(1) E^c(B) = (Ç E^c (b) for b ∈ B) and E^c [B] = (Ç E^c [b] for b ∈ B).
(2) <A-B, E↑(A-B)> =< B', E↑(B')> is a sub-hypernet of <A - {b}, E↑(A-{b})> for every b ∈ B.
(3) <A, E^c(B) > is a sub-hypernet of <A, E^c (b)> for every b ∈ B.
(4) <B', E [B′]> = ∩ <A - {b}, E [A - {b}]> for b ∈ B, so the order of the deletion of the b ∈ B ⊆ A does not affect the result. ♦

Proof:

(2) and (3) follow at once because it is less "damaging" to <A, E> to remove one b ∈ B from <A, E> than it is to delete all the members of B from <A, E>.
(4) We consider the case in which B = {a, b} ⊆ A since it is obvious if B = {a}, a ∈ A. First, <B', E [B']> = <A - {a, b}, E [A-{a, b}]>. Next we examine <A - {a}, E [A - {a}]> ∩ <A - {b}, E [A-{b}]>. Its underlying set is

(A - {a}) ∩ (A - {b}) = (A - {a, b}). Its set of edges is E [A-{a}] ∩ (E [A-{b}], i.e. all the edges in E that do not involve a ∈ A and do not involve b ∈ A, i.e E [A-{a, b}].Thus <A - B, E [A - B]> = <B', E [B'] > = ∩ <A – {b}, E [A – {b}]> in this case, and since <A - {b}, E [A - {b}], over all b ∈ B in this case, and since ∩ and ∪ are commutative, the order in which the members of B are deleted does not matter. ♦

Here follow some observations that are all relatively easy to prove. Consider a hypernet <A, E> with a, b ∈ A and a ≠ b, and the list

<A, E>, <A - {a}, E↑(A-{a})>,
<A - {b}, E↑(A - {b})>,
<A - {a, b}, E↑(A - {a, b})>,
<A, E^{c}(a)>, <A, E^{c}(b)>, <A, E^{c}({a, b}) >

of sub-hypernets of <A, E>. Then

(1) Let s ∈ A - {a, b}. d(s) in <A, E> is ≥ its value in all the other members of the list. Its value in <A, E^{c}(a)> is ≥ its value in <A - {a}, E↑(A - {a})>, in <A - b}, E↑(A - {b})>, in <A - {a, b}, E↑(A - {a, b})> and in <A, E^{c} (a)>, <A, E^{c} (b)> and <A, E^{c} ({a, b})>. Its value in <A - {a}, E↑(A - {a})> is ≥ its value in <A - {a, b}, E↑(A - {a, b})>, and its value in <A, E^{c} ({a, b})> is ≥ its value in <A - {a, b}, E↑(A - {a, b})>. Further, its values in <A, E^{c} (a)>, <A, E^{c} (b)> and <A,E^{c}({a.b})> are ≥ its values in <A, E^{c} [a]>, <A, E^{c}[b]> and <A, E ^{c}[{a, b}]> respectively.

(2) Vertex adjacency and edge adjacency in <A - {a, b}, E↑(A - {a, b})> ensures these adjacencies in all the other members of the list.

(3) For all s, t ∈ (A - {a, b}) the length of the shortest s — t path in <A - {a, b}, E↑(A-{a, b})> is ≥ the length of the shortest s — t path in each of the other members of the list.

(4) If <A, E^{c}({a, b}) > is connected then so are <A, E^{c}(a) > , <A, E^{c}(b) > and <A, E>. Every component of <A, E^{c}({a, b}) > is a sub-hypernet of a component of <A, E>.

(5) Every vertex basis of <A, E^{c}({a, b}) > contains a vertex basis of <A, E^{c}(a)>, of <A, E^{c}(b)>, and of <A, E>. ♦

11. Hypertrees

Definition 11.1: A hypernet <A, E_T> is called a ***hypertree*** iff <A, E_T> is minimally connected in the sense that deletion of any $E_i \in E_T$ will disconnect <A, E_T>.♦

As a direct consequence of the definition we see that

- Every hypertree is connected.
- A hypertree has no circuits, where, for the purposes of this chapter only, the term circuit includes closed paths of length 2.
- For every a, b ∈ A of a hypertree <A, E_T>, either $\lambda(\{a, b\}) = \emptyset$ or $\lambda(\{a, b\})$ is a singleton.
- For every a, b ∈ A of a hypertree <A, E_T>, there exists one and only one path a — b in <A, E_T>.

Theorem 11.1: The following statements are logically equivalent:
(1) T = <A, E_T> is a hypertree.
(2) T is connected and has no circuits.
(3) T is connected and has $|A|$ - 1 edges each of which labels a distinct vertex adjacency.
(4) T has no circuits, and has $|A|$ - 1 vertex adjacencies each of which has a singleton label.
(5) For all a, b ∈ A, there is precisely one path a — b in T. ♦

Proof:
(1) ⇒ (2): If T is a hypertree then it is minimally connected, so it is connected. Assume that there is a circuit in T. Then deletion of any edge in this circuit cannot disconnect T, so T is not minimally connected. It follows that T has no circuits.
(2) ⇒ (3): If T is connected then it has at least $|A|$ - 1 edges, and thus vertex adjacencies with at least a singleton label on each. If T has more than $|A|$ - 1 edges then it must have at least one circuit. It follows that T has precisely $|A|$ - 1 edges. If two of these edges label any one vertex adjacency in T then T has a circuit. Since T has no circuits by (ii), each edge in T must belong to a singleton label on a vertex adjacency.
(3) ⇒ (4): By the argument above, T can have no circuits as it is connected and has $|A|$ - 1 edges. Since each edge labels a single vertex adjacency there are $|A|$ - 1 vertex adjacencies, and each of these has a singleton label consisting of a unique edge, though we may have edges that are equal sets of course, because T has no circuits.
(4) ⇒ (5): T has no circuits, and has $|A|$ - 1 vertex adjacencies each with a singleton label. It follows that T is connected, so for a, b ∈ A there is at least one path a — b in T. Suppose there was another distinct path between a and b in T. Then T would have at least one circuit. It follows that for all a, b ∈ A there is a unique path a — b in T.
(5) ⇒ (1): T has precisely one path a — b for all a, b ∈ A, so T is connected. Deletion of any edge on such a path will disconnect T, so T is minimally connected, and hence T is a hypertree. ♦

Definition 11.2: A vertex a ∈ A of a hypertree T = <A, E_T> is called a ***pendant*** of T iff d(a) = 1. Any a ∈ A that is not a pendant has d(a) ≥ 2 and is called an ***internal vertex*** of T. ♦

Since a tree $T = \langle A, E_T \rangle$ has $|A| - 1$ vertex adjacencies, each with a singleton label, summing over all $a \in A$ yields $\Sigma\, d(a) = 2\,(|A| - 1)$, and this number is divided among the $|A|$ vertices in such a way that no $a \in A$ has $d(a) = 0$. If $|A| \geq 2$, so that the sum of the degrees is ≥ 2, then T has at least two pendants. Deletion of any internal vertex from any hypertree T will disconnect T. ♦

Theorem 11.2: An element $a \in A$ of a hypertree $T = \langle A, E_T \rangle$ is a pendant of T iff there is precisely one edge $E_i \in E_T$ with $E_i = \{a\}$ and precisely one edge $E_j \in E_T$ with $\{a\} \subset E_j$. ♦

Proof: If $a \in A$ is a pendant then we must have a single edge $E_j \in E_T$ with $\{a\} \subset E_j$, and E_j must be adjacent to some $E_i \in E_T$ by a. This means that we must have $E_i = \{a\}$ so that $d(a) = 1$ (since $E_i = \{a\}$ does not contribute an arc to $\langle A, E_T \rangle$: E_i is a dummy edge that is not counted in $|E_T|$). Further, there can be no other $E_k \in E_T$ that is adjacent to any other edge than E_i because then d(a) would not be 1 and so a would not be a pendant of T. Conversely, if we have precisely one $E_i \in E_T$ with $E_i = \{a\}$ and precisely one edge $E_j \in E_T$ with $\{a\} \subset E_j$ then it is clear that $d(a) = 1$, so a is a pendant of T. ♦

For every pendant $a \in A$ of a hypertree $T = \langle A, E_T \rangle$ we thus have a single singleton edge $E_i \in E_T$ with $E_i = \{a\}$, not counted in $|E_T|$.

Theorem 11.3: Deletion of a pendant $a \in A$ from a hypertree $T = \langle A, E_T \rangle$ will disconnect T iff there is at least one vertex adjacency (c, E_i, d), $c, d \in A$ and $E_i \in E_T$, with $a \neq c$ and $a \neq d$ and $a \in (E_i - \{c, d\})$, and $d(a) = 1$.♦

Proof: If only a pendant a is deleted from T then this will not disconnect T, so if this deletion is to disconnect T then deletion of a must delete at least one edge not incident with a from T. Conversely, if $a \in A$ and $a \in (E_i - \{c, d\})$ for some (c, E_i, d) in T then deletion of a from T will disconnect T, and since $d(a) = 1$, a is a pendant. ♦

Definition 11.3: Given any connected hypernet $\langle A, E \rangle$, $T = \langle A, E_T \rangle$ with $E_T \subseteq E$ is said to be a ***spanning hypertree*** of $\langle A, E \rangle$ iff T is a minimally connected sub-hypernet of $\langle A, E \rangle$.♦

Theorem 11.4: Every connected hypernet $\langle A, E \rangle$ has at least one spanning hypertree. ♦

Proof: $\langle A, E \rangle$ is connected. By part (ii) of theorem 11.1, if $\langle A, E \rangle$ has no circuits then it is a hypertree and is of course spanning. If $\langle A, E \rangle$ has a circuit, delete one edge on that circuit and test the result. Either it is connected and has no circuits, so it is a spanning hypertree, or it is connected and has a circuit. In the latter case, delete one edge on that circuit and test the result. Either it is connected and has no circuits, so it is a spanning hypertree, or it is connected and has a circuit. Proceeding in this manner we produce a spanning hypertree that is a sub-hypernet of $\langle A, E \rangle$.♦

Let $\langle A, E \rangle$ be a hypernet and let $T = \langle A, E_T \rangle$ be a spanning hypertree of $\langle A, E \rangle$. The $|E_T| = |A| - 1$ edges, not counting the singleton dummy pendant edges, are called ***branches*** of $\langle A, E \rangle$ with respect to T, and the remaining $|E - E_T|$ edges of $\langle A, E \rangle$ are called ***chords*** of <A, E> with respect to T. Since any hypernet <A, E> is such that A is partitioned by the components of <A, E>, and since each of these components has at least one spanning hypertree , <A, E> can be spanned by a forest of k spanning hypertrees where k is the number of components of <A, E>, and of course k = 1 iff <A, E> is connected.

Consider a connected hypernet <A, E> and a spanning hypertree T = <A, E_T> of <A, E>. Now there may be another spanning hypertree T´ = <A, E'_T> of <A, E> that differs from T only inasmuch as for at least one vertex adjacency (a, E_i , b) in T, T´ has in it the vertex adjacency (a, E_j , b) with a, b $\in$ A and E_i, $E_j \in$ E, $E_i \in E_T$, $E_j \in E'_T$, and $E_i \neq E_j$. This leads to the following definition.

Definition 11.4: Let T = <A, E_T> be a spanning hypertree of a connected hypernet <A, E>. The join of all the spanning hypertrees of <A, E> that have precisely the same vertex adjacencies {a, b}, a, b $\in$ A, as T but are pairwise different in at least one vertex adjacency by virtue of containing that vertex adjacency by an edge $E_j \in$ E different from the edge $E_i \in E_T$ by which the same two vertices are adjacent in T, is called a ***spinney*** of <A, E>. ♦

A spinney has no circuits.

Theorem 11.5: Let <A, E> be a connected hypernet. A sub-hypernet <A, E_T>, $E_T \subseteq$ E, of <A, E> is a spanning hypertree of <A, E> iff, for all a, b $\in$ A, transfering any $E_i \in (\lambda(\{a, b\}) - \lambda_T(\{a, b\}))$ to $\lambda_T(\{a, b\})$, where λ_T is the labelling function of T, yields a connected spanning sub-hypernet <A, ($E_T \cup \{E_i\}$)> of <A, E> such that <A, ($E_T \cup \{E_i\}$)> has precisely one closed path of length 2. ♦

Proof: If transferring any edge from $(\lambda(\{a, b\}) - \lambda_T(\{a, b\}))$ to $\lambda_T(\{a, b\})$ yields a spanning sub-hypernet of <A, E> that has precisely one closed path of length 2 then <A, E_T> is minimally connected and must be a spanning hypertree of <A, E>. Conversely, if <A, E_T> is a spanning hypertree of <A, E> then transferring precisely one edge E_i from $(\lambda(\{a, b\}) - \lambda_T(\{a, b\}))$ to $\lambda_T(\{a, b\})$ for any a, b $\in$ A that are vertex adjacent in <A, E_T> will yield at least one closed path, with vertices a and b in <A, ($E_T \cup \{E_i\}$)>, since <A, E_T> is minimally connected. The transfer cannot yield more than one such closed path unless $|\lambda_T(\{a, b\})| > 1$ before the transfer, which is impossible since <A, E_T> is a hypertree and thus $|\lambda_T(\{a, b\})| = 1$. ♦

Definition 11.5: Let <A, E> be a connected hypernet and let T = <A, E_T> be a spanning hypertree of <A, E>. A closed path formed by transferring precisely one edge E_i from ($E - E_T$) to E_T to produce <A, ($E_T \cup \{E_i\}$)> is called a ***fundamental circuit*** of <A, E> with respect to T. The number of chords, and hence the number of fundamental circuits, of a connected hypernet <A, E> is the same with respect to every spanning hypertree <A, E_T> of <A, E>. This number is called the ***cyclomatic number*** ν(<A, E>) of <A, E>, and is given by $\nu(\langle A, E\rangle) = |E| - (|A| - 1) = |(E - E_T)| = |E| - |E_T|$.♦

We will not pursue a theory of circuits in this report.

It is clear that closed paths of length 2, not regarded as circuits in hypernets in general, are a source of some embarrassment when dealing with circuits in a hypernet. We will see, in later sections, that in certain hypernets the problem effectively disappears.

12. Connectivity and cut-sets

Definition 12.1: Let <A, E> be a connected hypernet. R $\subseteq$ E is an ***edge cut-set*** of <A, E> iff <A, (E – R)> is a disconnected sub-hypernet of <A, E> and no proper subset of R has this property. V $\subseteq$ A is a ***vertex cut-set*** of <A, E> iff <A – V, E$\uparrow$(A-V)> = <V^c, E$\uparrow$ V^c > is a disconnected sub-hypernet of <A, E> and no proper subset of V has this property. ♦

Observations: Let <A, E> be a connected hypernet.

(1) {a} $\subseteq$ A is a vertex cut-set of <A, E> iff a is a cut-vertex in <A, E>.

(2) If R $\subseteq$ E is an edge cut-set of <A, E> and every $E_i \in$ R is such that a $\in E_i$, a $\in$ A, but is not adjacent with any vertex by E_i , then a is a cut-vertex in <A, E>.

(3) If we partition A into two sets A_1 and A_2 then any minimal set of edges of <A, E> the deletion of which cuts all the paths a_1 — a_2 with $a_1 \in A_1$ and $a_2 \in A_2$ is an edge cut-set of <A, E>. Any minimal set of vertices of <A, E> with the same property is a vertex cut-set of <A, E>.

(4) T = <A, E_T> is a hypertree iff every $E_i \in E_T$ constitutes an edge cut-set {E_i} of T. Further, {c} $\subseteq$ A is a vertex cut-set of <A, E_T>, i.e. c is a cut-vertex of T, iff c is an internal vertex of T or c is such that c $\in E_i$ – {a, b} for at least one vertex adjacency (a, E_i , b) in T with a, b $\in$ A and c $\neq$ a and c $\neq$ b and $E_i \in E_T$. ♦

Definition 12.2: Let <A, E> be a connected hypernet. The smallest number of vertices that must be deleted from <A, E> to disconnect it is called the ***vertex connectivity*** vc <A, E> of <A, E>, and the smallest number of edges that must be deleted to disconnect <A, E> is called the ***edge connectivity*** ec <A, E> of <A, E>. ♦

Recall that deleting a vertex adjacency (a, E_i , b) from a hypernet <A, E> means to delete E_i from λ({a, b}), and that this does not delete the arc between a and b unless λ({a, b}) = {E_i}.

Theorem 12.1: Let <A, E> be a connected hypernet. Then
vc <A, E> $\leq$ ec <A, E> = minimum degree min d(a) of all the a $\in$ A in <A, E> when loops are disregarded. ♦

Proof: We can clearly disconnect <A, E> by deleting min d(a) edges from <A, E>, thereby cutting off vertex a. Deletion of these edges E_i can be achieved by deleting one vertex from each of these edges E_i other than vertices adjacent by that E_i (one of which is of course a). It follows that, since these vertices need not all be distinct for distinct edges,
vc <A, E> $\leq$ ec <A, E>. It is clear that ec <A, E> = min d(a). ♦

Theorem 12.2: R $\subseteq$ E is an edge cut-set of a spinney S = <A, E> iff there is at least one pair {a, b} $\subseteq$ A for which R = λ({a, b}). ♦

Proof: If R = λ({a, b}) then deletion of R from S will disconnect S and no proper subset of R will "cut" a from b, so R is an edge cut-set of S. If R is an edge cut-set of S then deletion of R from S must ”cut” the arc between two vertices a, b ∈ A in S. It follows that R = λ({a, b}) and no proper subset of R will ”cut” a from b. ♦

Theorem 12.3: Every edge cut-set R ⊆ E of a connected hypernet <A, E> is such that at least one edge from every spanning hypertree of <A, E> belongs to R. ♦

Proof: If deletion of R from <A, E> does not entail deletion of at least one edge from each spanning hypertree of <A, E> then there will remain in <A, E – R> at least one spanning hypertree of <A, E>. But then <A, E – R> is connected, so R cannot be an edge cut-set of <A, E>. It follows that deletion of an edge cut-set from <A, E> ”cuts” every spanning hypertree of <A, E>.♦

Theorem 12.4: Every closed path of length > 1, in a connected hypernet <A, E>, has an even number of edges in common with every edge cut-set of <A, E>.♦

Proof: Let R ⊆ E be an edge cut-set of <A, E>. Deletion of R from <A, E> will partition A into two subsets, A_1 and A_2, in <A, E – R> in such a way that for any $a_1 \in A_1$ and any $a_2 \in A_2$ there is no path a_1 — a_2 in <A, E – R> because there is at least one member of R on every such path. Consider any closed path P in <A, E>. If all the vertices that lie on this closed path belong to A_1, or if they all belong to A_2, then R has zero edges in common with that path. If some of the vertices on P belong to A_1 and others to A_2, then P must cross back and forth between A_1 and A_2. Start tracing P at $a_1 \in A_1$ for example. P must end at a_1, so, in tracing P, every time we move to A_2 with an edge on P we must move back to A_1 with another edge on P (since P is a path). Thus P shares an even number of edges with R. ♦

Definition 12.3: An edge cut-set R of a connected hypernet <A, E> is said to be a ***fundamental edge cut-s***et with respect to a spanning hypertree T = <A, E_T> of <A, E> iff one and only one edge of T belongs to R. ♦

The number of fundamental edge cut-sets of <A, E> with respect to T is (│A│-1), regardless of which spanning tree T of <A, E> is chosen. Recall that the pendants of a hypertree (or spinney) T = <A, E_T> each belong to a singleton edge, but such edges are dummy edges that allow us to have a path incident with a pendant and are not counted among the edges of E_T.

Theorem12.5: With respect to a given spanning hypertree T = <A, E_T> of a connected hypernet <A, E>, a chord edge of <A, E> that determines a fundamental circuit P of <A, E>, when transfered to T, belongs to every fundamental edge cut-set of <A, E> associated with those branches of <A, E>, i.e. edges of T, that belong to P, and that chord belongs to no other fundamental circuit, in <A, E>, with respect to T. ♦

Proof: Consider the branches of <A, E>, with respect to T, that lie in P. Associated with each of these branches there is a fundament edge cut-set of <A, E> that has the relevant branch as a member. Now the chord presents, with other branches in P, a “way round” the branch that determines this fundamental edge cut-set, so to disconnect <A, E> our chord must belong to this fundamental edge cut–set. Next, suppose that our chord belongs to both fundamental circuit P and to another distinct fundamental circuit P`, both in <A, E> and with respect to T.

Now our chord then lies on both P and P`, and all the other edges in P and P` are branches of <A, E> with respect to the same spanning hypertree T = <A, E_T>, i.e. they are members of E_T. Now we can move from one end vertex of our chord through P to the other end vertex of our chord, and then back through P` to where we started. Then we have traced a walk that is either a closed path, or which determines more than one closed path, using only edges in E_T. This contradicts the fact that T = <A, E_T> is a given hypertree in <A, E>.♦

Theorem12.6: A set R $\subseteq$ E is an edge cut-set of a connected hypernet <A, E> iff <A, E – R> = <A, R^c> is a maximal disconnected spanning sub-hypernet of <A, E> in the sense that for all R` with $R^c \subseteq$ R` $\subseteq$ E, <A, R`> is a connected hypernet. ♦

Proof: If R $\subseteq$ E is an edge cut-set of <A, E> then <A, R^c> is a disconnected sub-hypernet of <A, E>, and no $R_s \subset$ R has this property, so if R` is such that $R^c \subset$ R` then <A, R`> is connected, i.e. <A, R^c > is a maximal disconnected spanning sub-hypernet of <A, E>. Conversely, if <A, R^c > is a maximal disconnected spanning sub-hypernet of <A, E> then deletion of R from <A, E> disconnects <A, E>, and deletion of any R' $\subset$ R will not disconnect <A, E>, i.e. <A, $(R')^c$ > is connected. It follows that no proper subset of R will, when deleted, disconnect <A, E>, so R is an edge cut-set of <A, E>.♦

Constructional Scheme 12.1: Let R $\subseteq$ E be any disconnecting set of edges of a connected hypernet <A, E>. To find an edge cut-set included in R we may proceed as follows.

(1) Find any $E_k \in$ R such that E_k is a bridge in <A, E>. Then $\{E_k\} \subseteq$ R is an edge cut-set of <A, E>. If there is no such member of R, proceed to (2).
(2) Choose any $E_k \in$ R and form <A, E – $\{E_k\}$>. Find any $E_l \in$ R – $\{E_k\}$ such that E_l is a bridge in <A, E – $\{E_k\}$>. Then $\{E_k, E_l\} \subseteq$ R is an edge cut-set of <A, E>. If there is no such member of R – $\{E_k\}$, set $R^t_1 = \{E_k\}$ and proceed to (3).
(3) Choose any $E_m \in$ R - R^t_1 and set $R^t_2 = \{E_m\} \cup R^t_1$. Form <A, E - R^t_2 > (which is <A, E – $\{E_m, E_k\}$> here). Find any $E_l \in$ R - R^t_2 such that E_l is a bridge in <A, E - R^t_2 >. Then $R^t_2 \cup \{E_l\}$ is an edge cut-set of <A, E>. If there is no such member of R - R^t_2, repeat (3) defining $R^t_m = \{E_l\} \cup R^t_{m-1}$, m = 3, 4, ..., successively. Eventually we find an edge cut-set R^t_n, or we find R^t_n = R, for some n, in which case R is an edge cut-set of <A, E>.♦

The scheme works because we know that R is a disconnecting set so there must be an edge cut-set included in R, and we keep "weakening" <A, E> by taking out members of R from <A, E> successively until we find, in R, a bridge of <A, E - R^t_m> in which case $R^t_m \cup$ {bridge} is an edge cut-set of <A, E>, or we do not find a bridge in any step in which case R is an edge cut-set of <A, E>.

Theorem 12.7: B $\subseteq$ A is a vertex cut-set of a connected hypernet <A, E> iff B is a minimal set of vertices such that for every spinney S of <A, E> there is at least one internal vertex of S that belongs to B, or there is at least one vertex adjacency {a, b} in S such that a $\notin$ B and b $\notin$ B and every $E_i \in \lambda(\{a, b\})$ has $(E_i – \{a, b\}) \cap B \neq \emptyset$, or both. ♦

Proof: Suppose that B is a vertex cut-set. Then if the condition does not hold deletion of B from <A, E> will leave at least one hypertree T $\angle$ <A, E>, so <A – B, E $\uparrow$(A – B) will be

connected, contradicting the fact that B is a vertex cut-set of <A, E>. Conversely, if the condition holds then deletion of B from <A, E> disconnects every spinney S ∠ <A, E>, and thus also <A, E>. Since B is minimal, B is a vertex cut-set of <A, E>.♦

Theorem 12.8: Let <A, E> be a connected hypernet and $B \subseteq A$ be a vertex cut-set of <A, E>, and let S be any spinney in <A, E>.

(1) Suppose that <A, E – E↑B> is connected, and let T = <A, E_T> be a spanning hypertree in S. T = <A, E_T> is a spanning hypertree of <A, E – E↑B> iff every $E_i \in E_T$ is such that $E_i \cap B = \emptyset$.

(2) If T = <A, E_T> is a spanning hypertree in <A, E – E(B) > then at least one internal vertex of S belongs to B. ♦

Proof: Recall that E↑B = $\{E_i \in E \mid E_i \subseteq B \neq \emptyset\}$.

(1) If T is a spanning hypertree of <A, E – E↑B > then every $E_i \in E_T$ has $E_i \cap B$ because if this were not so then E_i would not be a member of E – E↑B but would belong to E↑B and could thus not belong to a spanning hypertree in <A, E – E↑B >. Conversely, if every $E_i \in E_T$ has $E_i \cap B = \emptyset$ then every $E_i \in E_T$ belongs to E – E↑B, so deletion of E↑B ⊆ E from <A, B> does not affect T = <A, E_T>. T is a spanning hypertree of S ∠ <A, E>, so T is a spanning hypertree of <A, E – E↑B>.

(2) Recall that E(B) ⊆ E, B ⊆A, of <A, E> is the set E(B) = $\{E_i \in E \mid (a, E_i, b), a, b \in A$ and $(E_i - \{a, b\}) \cap B \neq \emptyset\}$, i.e. the set of all edges in the name of at least one member of B. Now T = <A, E_T> is a spanning hypertree in <A, E – E(B)>, and B is a vertex cut-set of <A, E> so <A – B, E↑(A – B)> is disconnected. Thus deletion of all the edges of E(B) leaves <A, E – E(B)> connected, so <A, E – E(B)> has a spanning hypertree T, but deletion of B from <A, E> leaves <A – B, E↑(A – B)> disconnected, and this can only happen if B contains at least one internal vertex of T so that deletion of B from <A, E> will disconnect <A, E> but deletion of E(B) from <A, E> will not disconnect <A, E>.♦

Corollary 12.1: Let T be a spanning hypertree of a connected hypernet <A, E>, and let <A, E> be disconnected by deleting the vertex cut-set B from <A, E> by virtue of deletion of internal vertices of T only. Then T is a spanning hypertree of <A, E – E(B)>. ♦

Proof: follows at once from the fact that deletion of E(B), only, from <A, E> will not disconnect <A, E> but deletion of B, and thus E(B), and in fact E [B], from <A, E> will indeed disconnect <A, E> because at least one internal vertex of T belongs to B. ♦

Theorem 12.9: Let B ⊆ A be a vertex cut-set of a connected hypernet <A, E>. Then <A, E – E(B)> is disconnected iff every spinney S of <A, E> has at least one vertex adjacency {a, b}, a, b ∈ A, such that every $E_i \in \lambda_S(\{a, b\})$ has $(E_i - \{a, b\}) \cap B \neq \emptyset$, where λ_S is the labelling function of S. ♦

Proof: If <A, E – E(B)> is disconnected, by deleting only E(B) from <A, E>, then every spinney S of <A, E> is disconnected by the deletion of E(B) from <A, E>. To do this, deletion of E(B) from any spinney S must involve deletion of at least one arc in S. Thus there must be an {a, b} in S such that $\lambda_S(\{a, b\}) \subseteq E(B)$, so for each $E_i \in \lambda_S(\{a, b\})$ we must have

$(E_i - \{a, b\}) \cap B \neq \emptyset$. Conversely, if every spinney S in <A, E> has at least one vertex adjacency {a, b} such that every $E_i \in \lambda_S(\{a, b\})$ has $(E_i - \{a, b\}) \cap B \neq \emptyset$, i.e. $\lambda_S(\{a, b\}) \subseteq E(B)$, then <A, E – E(B)> is disconnected. ♦

Theorem 12.10: If $a \in A$ is a cut-vertex of a connected hypernet <A, E>, but not of <A, E – E(a)>, then

$E(a) = \{E_i \in E \mid (c, E_i, d)$ is a vertex adjacency by E_i in <A, E> and $a \in (E_i - \{c, d\})\}$ includes an edge cut-set of <A, E>.♦

Proof: Deletion of $a \in A$ from <A, E> leaves us with a disconnected hypernet <A – {a}, E↑(A – {a})>, but deletion of a from <A, E – E(a)> leaves it connected, i.e. <A – {a}, E – E(a)> is connected. Note that E – E(a) is the set of all the edges of E that are not in the name of a, while E↑(A – {a}) is the set of all edges that do not have a in them, so $E{\uparrow}(A - \{a\}) \subseteq (E - E(a))$. In theorem 10.4 on deletion of vertices we showed that if a is a cut-vertex of <A, E>, i.e. is (1, 0) in <A, E>, then it is (1, 0) or (0, 0) in <A, $(E(a))^c$> = <A, E – E(a)>. Now a is not a cut-vertex in <A, E – E(a)>, so it is not (1, 0) in <A, E – E(a)> and must thus be (0, 0). Thus <A, E – E(a)> is disconnected, so E(a) must be a disconnecting set of edges in <A, E> and hence E(a) includes an edge cut-set of <A, E>. ♦

Finally, we notice that if <A,E> is connected but <A,E – E (a)> is disconnected, then a is a cut-vertex of <A,E>. The contrapositive is: If a is not a cut-vertex of <A,E> then
<A, E – E (a) > is connected.

13. Blocks

Definition 13.1: By a ***block*** <B, G> of a hypernet <A, E> we mean a maximal connected sub-hypernet, of <A, E>, that has no cut-vertex. ♦

Any block of <A, E> is a sub-hypernet of a component of <A, E>.

Theorem 13.1: If <B, R> is a block of a hypernet <A, R> then <B, R> is a sub-hypernet of some block of a hypernet <A, E> with $R \subseteq E$. ♦

Proof: If <B, R> is a block of <A, R> then it is a sub-hypernet of <A, E>. Since <B, R> must then be a connected sub-hypernet, of <A, E>, with no cut-vertex, it is a sub-hypernet of some maximal connected sub-hypernet, of <A, E>, that has no cut-vertex, so <B, R> is a sub-hypernet of some block of <A, E>.♦

Theorem 13.2: Let <B, G> be a block of a hypernet <A, E>, with $|B| \geq 3$. Then
(1) there is no $b \in B$ such that <B, G – G(b)> or <B – {b}, G↑(B – {b})> is in C_0, and
(2) there is no bridge in <B, G>, and
(3) if every $E_i \in G$ has $|E_i| > 2$ then there is no bridge in <B, G>. ♦

Proof:
(1) <B, G> is connected. If there were some $b \in B$ such that <B, G – G(b)> or <B – {b}, G↑(B – {b})> were disconnected then b would be a cut-vertex of <B, G>, so <B, G> would not be a block.
(2) Suppose that $E_i \in G$ is a bridge in <B, G>. Then there is a vertex adjacency (a, Ei, b), $a, b \in B$, that provides the only path between a and b in <B, G>. Since <B, G> is connected, and $|B| \geq 3$, it follows that at least one of a and b is a cut-vertex of <B, G>. This contradicts the given fact that <B, G> is a block.
(3) If every $E_i \in G$ of the block <B, G> has $|E_i| > 2$, then consider a vertex adjacency (a, Ei, b), $a, b \in E_i \in G$. If E_i is a bridge in <B, G> then deletion of any $c \in (E_i - \{a, b\})$ will disconnect <B, G>, so c would be a cut-vertex of <B, G>, which is impossible. It follows that there is no bridge in <B, G>. ♦

Corollary 13.1:
(1) If a and b are distinct vertices of <B, G> then, for all $c \in B$, $c \neq a$ and $c \neq b$, there is at least one path a — b that does not go via any $E_i \in G$ for which $c \in E_i$.
(2) If $E_i \in G$ is a bridge in <B, G> then $|E_i| = 2$.
(3) For all $a \in B$, there are no two distinct vertices $b, c \in B$ such that every path b — c in <B, G> goes via some vertex adjacency (d, E_i, f) with $a \in (E_i - \{d, f\})$.♦

Proof:
(1) Follows from the fact that c is not a cut-vertex of <B, G>, so <B, G – G(c)> is connected.

(2) If $E_i \in G$ with $|E_i| > 2$ were a bridge in <B, G> then, given any vertex adjacency (a, E_i, b) by E_i in <B, G>, $a, b \in B$, each $c \in B$ with $c \in (E_i - \{a, b\})$ would be a cut-vertex of <B, G>.

(3) If there were such an $a \in B$ it would be a cut-vertex of the block <B, G>.♦

Theorem 13.3: The following assertions are logically equivalent:

(1) <B, G> is a block, of hypernet <A, E>, with $|B| \geq 3$.

(2) For all distinct $a, b, c \in B$ of a hypernet <B, G> ∠ <A, E> there exists at least one path a —— c, in <B, G>, which is such that b is not between a and c on a —— c, and <B, G> is a maximal such sub-hypernet.

(3) For all distinct $a, b, c \in B$ of a block <B, G> ∠ <A, E>, there exists a path P_1 joining a and c in <B, G> that satifies the following conditions:

a) P_1 has length ≥ 2.

b) Given any $b \in (B - \{a, b\})$ such that b is between a and c on P_1, it is always possible to find a path P_2 joining a and c in <B, G> such that b is not between a and c on P_2, and <B, G> is a maximal such sub-hypernet of <A, E>.♦

Proof:

(1) ⇒ (2): There certainly exists a path a —— c in <B, G> because <B, G> is a block with $|B| \geq 3$. Now b is not a cut-vertex of <B, G>, so we do not have (a - b - c), i.e. b is not between a and c on every path a —— c in <B, G>. It follows that there is at least one path a —— c in <B, G> such that b is not between a and c on that path. Because <B, G> is a block it is a maximal such sub-hypernet of <A, E>.

(2) ⇒ (3): There is a path joining a and c in <B, G> such that b is not between a and c on that path. Let P_1 be the path a —— b —— c, so P_1 has length ≥ 2, and P_1 exists because, from (2), every pair of vertices in B are joined in <B, G>. Further, we know from (2) that there exists a path a —— c, in <B, G>, which is such that b is not between a and c on that path. Any such path will do for P_2. Finally, maximality of <B, G> from part (2) remains valid because we have only used (2) to derive (3).

(3) ⇒ (1): We know that $|B| \geq 3$ because the length of P_1 is at least 2. Further, all distinct a and c in B are joined in <B, G>, so <B, G> is connected. Now there are no distinct $a, b, c \in B$ such that (a - b - c), for in choosing P_1 as the concatenation of paths a —— b —— c we would then not be able to find a path P_2 joining a and c such that b is not between a and c on P_2. Thus <B, G> also has no cut-vertices, and we have derived (1). ♦

Theorem 13.4: Let $<B_0, G_0>$ and $<B_1, G_1>$ be distinct blocks, of a hypernet <A, E>, for which $B_0 \cap B_1 = B_{01} \neq \emptyset$. Then $B_{01} = \{b\}$, a singleton, and given any $a \in (B_0 - B_{01})$ and any $c \in (B_1 - B_{01})$, b is between a and c on every path a —— c in $<B_0 \cup B_1, G_0 \cup G_1>$, i.e. (a - b - c) in $<B_0 \cup B_1, G_0 \cup G_1>$.♦

Proof: $<B_0 \cup B_1, G_0 \cup G_1>$ is clearly not a block in <A, E>, and $B_{01} \neq \emptyset$, which means that $<B_0 \cup B_1, G_0 \cup G_1>$ is a connected sub-hypernet of <A, E>, so there exists at least one $b \in B_0 \cup B_1$ such that b is a cut-vertex of $<B_0 \cup B_1, G_0 \cup G_1>$. Now $b \notin (B_0 - B_{01})$ for, if it were, then b would be a cut-vertex of $<B_0, G_0>$, but $<B_0, G_0>$ is a block. Similarly $b \notin (B_1 - B_{01})$, so we have $b \in B_{01}$. Let $p \in B_{01}$, with $p \neq b$. Then we can find a path a —— p in $<B_0, G_0>$ such that b is not between a and p on a —— p because b is not a cut-vertex of $<B_0, G_0>$. Similarly we can find a path p —— c in $<B_1, G_1>$ such that b is not between p and c on p —— c because b is not a cut-vertex of $<B_1, G_1>$. But then b is not between a and c on the concatenation of paths a —— p —— c, which contradicts the fact that b must be a cut-vertex of $<B_0 \cup B_1, G_0 \cup G_1>$. Thus there is no such $p \in B_{01}$, so $B_{01} = \{b\}$, and since b is a cut-vertex of $<B_0 \cup B_1, G_0 \cup G_1>$ it follows that b must be between a and c on every path a —— c in $<B_0 \cup B_1, G_0 \cup G_1>$ where $a \in B_0$ and $c \in B_1$ and $a \neq b$ and $c \neq b$. ♦

14. Second intermission

We now move to transcription of some of the theory of Concept-Relationship Knowledge Structures (CRKS's) developed in Part I of [GVS99], describing the hypernet equivalent of a CRKS and examining some of its features. We will let the vertices of such hypernets represent concept-names as for CRKS's. In a CRKS each tuple of concept-names comes from a statement of relationship between the concept-names in that tuple. Two main features arise: First the occurrences of concept-names are ordered by the relevant statement of relationship, thus giving rise to a tuple of those concept-names and hence a direction from the first concept-name in the tuple to the last, and second, a given concept-name can appear more than once in a tuple.

In the hypernet equivalent of a CRKS each tuple of the CRKS is represented by an edge that is precisely the tuple set of, i.e. the set of concept-names of, that tuple. As a result we lose all direction – arrows become arcs – and a concept-name can only occur once in the edge equivalent to the relevant tuple. Thus a given set $S \subseteq A$ of a hypernet $<A, E>$ can be associated with several different tuples all of which have the same tuple set, but there is a 1 – 1 correspondence between the set of tuples of a given CRKS and the set of edges, and therefore the set of vertex adjacencies, of the equivalent hypernet.

15. Concept-Name Relationship Hypernets

Definition 15.1: By a ***concept-name relationship hypernet***, or ***CNR-hypernet***, we mean a hypernet <A, E> in which
(1) A is a set of concept-names and
(2) each edge $E_i \in E$ can be regarded as the tuple set of a tuple of concept-names that arises from a statement of relationship among those concept-names. ♦

Definition 15.2: A CNR-hypernet <A E> is called a ***formal hyperschema*** iff
(1) for all $a \in A$, $a \in E_i \in E$ for at least one non-singleton edge E_i, so $E[a] \neq \emptyset$ when we disregard singleton edges. Thus each $a \in A$ is associated with at least one other vertex of <A, E>.
(2) <A, E> has no circuits, i.e. no closed paths of any length other than 2.
(3) There is at least one $p \in A$ at which there is a special singleton edge $E_p \in E$ with $E_p = \{p\}$, and p also belongs to at least one other $E_i \in E$. Each such p is called a ***primary*** of <A, E>.
(4) There is at least one $g \in A$ at which there is a special singleton edge $E_g \in E$ with $E_g = \{g\}$, and g also belongs to at least one other $E_j \in E$. Each such g is called a ***goal*** of <A, E>. (We will distinguish primaries from goals later.)
(5) There are no singleton edges in <A, E> other than those at primaries and goals, and no singleton edge is used on any path in <A, E>. ♦

The reason for the singleton edges is that paths in <A, E> can "start" at primaries and "terminate" at goals. We will show later how it is possible to regard all paths as having a fixed direction in certain CNR-hypernets.

Definition 15.3: A formal hyperschema <A, E> is said to be ***complete*** iff it has no isolates. ♦

Note that no formal hyperschema can have complete isolates.

Theorem 15.1: If a formal hyperschema <A, E> is connected then it is complete, but the converse is not always true. ♦

Proof: If <A, E> is connected then it has no isolates, so <A, E> is complete. To prove that the converse is not always true we exhibit the following formal hyperschema, which is complete but not connected

where λ({a, c}) = {E_i} and E_i = {a, b, c} and where λ({b, d}) = {E_j} and E_j = {b, d} for example. Notice in passing that if we delete b, for example, then we get

c ■ a ■ d ■ ♦

Definition 15.4: The ***context-hyperschema*** of a ∈ A in a formal hyperschema <A, E> is a hypernet <A, E>[a] = <A[a], E[a]> ∠ <A, E> that is defined as follows. E[a] is, as defined before, the set of all E_i ∈ E that have a ∈ E_i, and A[a] = {b ∈ A | b belongs to at least one of the E_i ∈ E[a] }. ♦

Thus we can write A[a] = {È E_i | E_i ∈ E[a]}. Since E[a] = E↑A [a], because E↑A[a] is the set of all E_i ∈ E with E_i ⊆ A[a] and each such E_i must have a ∈ E_i given that A[a] = {È E_i | E_i ∈ E[a]}, we can also write <A,E>[a] = <A[a], E↑A[a]>, the maximum sub-hypernet of <A,E> that is induced by A[a] ⊆ A. So <A,E>[a] = <A[a], E[a]> = <A[a], E↑A[a]>. (See definitons 2.15, 4.1, and 4.8)

Definition 15.5: A ***betweenness sequence*** for a path-family f(a_1 — a_n) in a formal hyperschema <A, E> is found as follows. First, for all the members of λ({a_i, a_{i+1}}), i = 1, 2, ..., n - 1, for each vertex adjacency in f(a_1 — a_n), by which a_i and a_{i+1} are adjacent in f(a_1 — a_n), we list

$$a_i, E_{i1}, E_{i2}, ..., E_{im(i)}, a_{i+1}.$$

We then chain these lists together in succession from a_1 to a_n for f(a_1 — a_n). Next we write out each E_{ix} in the sequence, i.e. we replace each E_{ix} by the members of the set {v ∈ A | v ∈ E_{ix}}, getting a sequence of members of A starting with a_1 and ending with a_n. This is a betweenness sequence for f(a_1 — a_n) in <A, E>. Such a betweenness sequence is clearly not unique. (Note that a path-family is not empty, and it may only have one member.) ♦

16. Derivability in a Formal Hyperschema

Definition 16.1:

(1) Given any formal hyperschema <A, E> and a set $X \subseteq A$, we say that $a \in A$ is ***immediately derived from hypothesis*** X iff there is at least one $x \in X$ and at least one $E_i \in E$ by which there is a vertex adjacency $(x, E_i, a_{n(i)} = a)$, with every member of $(E_i - \{x, a_{n(i)}\})$ a member of X.

(2) Given any formal hyperschema <A, E> and a set $X \subseteq A$, we say that $a \in A$ is ***derivable in terms of hypothesis*** X in <A, E> iff there is a path p — a, $p \in A$, in <A, E> such that there exists at least one betweenness sequence S for p — a with the property that for every $t \in S$ we have
a) t is a primary of <A, E> or
b) $t \in X$ or
c) t is immediately derived from a subset of S_t, where S_t is the set of all predecessors of t in S.

(3) We say that $a \in A$ is ***derivable from*** P in <A, E>, or simply ***derivable*** in <A, E>, where P is the set of all primaries of <A, E>, iff a is derivable in terms of some $X \subseteq A$, by virtue of at least one path p — a and a betweenness sequence S for p — a, with either $X = \emptyset$ or such that every $x \in X$ is derivable in terms of $\emptyset$.

(4) If $a \in A$ is derivable in <A, E>, by virtue of a path p — a , p a primary of <A, E>, then p — a is called a ***derivation path*** for a in <A, E> and each such path p — a is called a derivation path for a in <A, E>, and a is said to be a ***derived vertex*** of <A, E>.♦

Definition 16.2: A complete formal hyperschema <A, E> is called a ***Concept-Relationship Knowledge Hypernet***, or simply a ***CRKH***, iff every vertex of <A, E> is derivable in <A, E>. ♦

Consider any CRKH <A, E>. Derivability in <A, E> induces a certain sense of direction on a CRKH in the following way. Given any part of a derivation path, p — a, of length ≥ 1, a is derived in terms of some of its predecessors in a betweenness sequence S_a for p — a that starts with p and ends with a.

Now we can specify, in a (complete) formal hyperschema <A, E> that is a CRKH, how to determine which of the $a \in A$ with an $E_i \in E$ such that $E_i = \{a\}$ are primaries of <A, E> and which are goals.

Simply stated, p is a ***primary*** of <A, E> iff there is a singleton edge $E_i = \{p\} \in E$ and every vertex adjacency {p, b} by one or more $E_j \in E$ that belong to $\lambda(\{p, b\})$ is such that

(1) p has a trivial derivation by a path of length zero and a set of hypothesis $X = \emptyset$ and
(2) b is derivable by virtue of an X that is a betweenness sequence, for the vertex adjacency {p, b}, that starts with p and ends with b.

Next, g is a ***goal*** of <A, E> iff g has a singleton edge $E_k = \{g\}$at it, and g is not a primary of <A, E>, and there is no vertex adjacency {g, a} on any derivation path for any vertex $a \in A$ in

<A, E>. It is evident that, since every vertex of a CRKH <A, E> is a derived vertex, we must have the following.

(1) There is at least one primary p ∈ A of <A, E> for which there exists at least one vertex adjacency (p, E_i, b), b ∈ A, in <A, E> for which every member of (E_i – {b}) is a primary of <A, E>.

(2) There is at least one goal g ∈ A of <A, E> that is not in any betweennness sequence in any but the last position.

We are now in a position to redefine circuits in the case of a CRKH: A closed derivation path, of any length whatsoever, is called a ***circuit***, and a CRKH has no circuits of any length (since singelton edges do not generate an arc at any vertex; here the primaries and goals of a CRKH).

From this point on we can visualize a direction for every vertex adjacency {a, b}, a, b ∈ A, in any CRKH <A, E>, the direction imposed by derivation. Thus we may replace arcs with arrows in each CRKH.

Theorem 16.1: Let <A, E> be a CRKH. There is at least one path that joins each primary of <A, E> to some goal of <A, E> in <A, E>, and there is at least one path that joins each goal of <A, E> to some primary of <A, E> in <A, E>. ♦

Proof: Let p be any primary of <A, E>. There is at least one derivation path incident with p. Follow that path incident with p. <A, E> has no circuits, and thus this path must have a finite length and can only be incident with a goal on the end of the path because no derivation path can end with another primary of <A, E>. Let g be any goal of <A, E>. There is at least one derivation path incident with g. Again <A, E> has no circuits so this path, which we follow in the reverse derivation mode, must have finite length and must end with a primary on the other end because it could not end with another goal of <A, E> unless we go with a derivation path to that goal, thus mixing forward and reverse directions along that path, and thus generating a semi-path that is not a path. ♦

Theorem 16.2: Let <A, E> be a CRKH, and let a ∈ A be neither a primary nor a goal of <A, E>. Then there is at least one path p — g in <A, E>, p some primary of <A, E> and g some goal of <A, E>, such that a lies on p — g, i.e. a is a member of the vertex subsequence of p — g. ♦

Proof: Since <A, E> is a CRKH, a is a derived vertex in <A, E>. Since a is a derivable, there is a derivation path p — a, p ∈ A, in <A, E>. By theorem 16.1 this path must continue on to some goal of <A, E>.♦

We now need to say something about paths in a CRKH.

Constructional Scheme 16.1: To construct a path tree, for a CRKH <A, E>, displaying and distinguishing every path from each primary of <A, E>. We will refer to vertices and edges of <A, E> and nodes and branches of the tree.

Note that we should bear in mind that derivation imposes directionality on a CRKH. It is clear that if we follow paths in a CRKH only in the "derivation direction" we will have no circuits in any CRKH. This directional ordering on paths in a CRKH may appear just to reduce a CRKH to a CRKS, but in the case of a CRKH we have

(1) a choice of the vertex by which two edges are adjacent in general and
(2) no ordering, and no repetition, of vertices in the edges by which vertices are adjacent.

This degree of choice gives us the potential, for example, to use any teaching metalanguage when we pick an interpretation of an CRKH in the educational applications mentioned in [GVS99]. The CRKH model is more flexible than the CRKS one in applications, and we have a strong link between the two models, to which link we will add more detail at a later stage of this report.

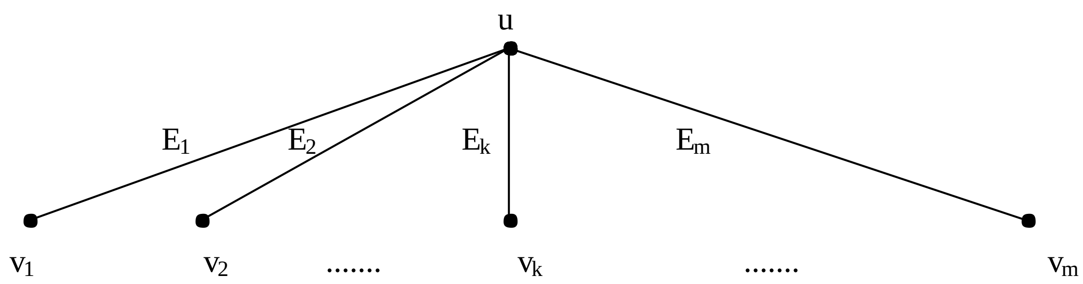
One final point before we tackle the constructional scheme: Derivability of a vertex b by virtue of a path a — b in a formal hyperscheme depends, for the induced direction of derivation onto a — b, on the existence of at least one appropriate betweenness sequence for a — b. We will see, in the following section, that there is a very specific characterization of appropriate betweenness sequences. Now for the scheme.

First we introduce an unlabelled dummy node to serve as the root of the path tree, and one only node for each primary of <A, E>. Connect each such node to the root with an unlabelled branch, and label each non-root node with the appropriate primary concept-name from A. From each node for a vertex $v \in A$ the tree now develops as follows. Find every vertex adjacency (v, E_i, w) in <A, E> for which w is derived through v and E_i, and suppose that $E_i = \{v = c_1, c_2, ..., c_k, ..., c_{n-1}, c_n\}$, and let $c_n = w$. Thus we find all such edges E_i with $E_i = \{v = c_1, c_2, ..., c_k, ..., c_{n(i)-1}, c_{n(i)}\}$ for some n(i). We now plot a new node for each such $c_{n(i)}$, and insert a branch between each node for v and every node for each of these $c_{n(i)}$. Each such branch is now labelled with the edge E_i that generates it, and each node for a given $c_{n(i)}$ is labelled with the concept-name for that $c_{n(i)}$. Repeat this for every $E_i \in E$. The resulting tree exhibits, along the paths from the root, every path from a primary to a goal in <A, E>, and distinguishes these paths. Each primary of <A, E> is represented by one only node, and every goal of <A, E> by at least one node. ♦

Constructional Scheme 16.2: Find all the paths between vertex u and vertex v in a CRKH <A, E>. Because of the derivation induced directionality in <A, E>, we can think of ourselves looking for all paths "from" a given $u \in A$ "to" a given $v \in A$.
First we should note that we can run a fast access cascade against the derivational direction in any CRKH just as easily as with this direction or without direction – see definition 4.7.

(1) Run a fast access cascade backward from $A_0 = \{v\}$ in <A, E>. Let the resulting hypernet be <A´, E´>. If $u \notin A'$ then there are no u — v paths in <A, E>.
(2) If $u \in A'$, then proceed as follows in <A´, E´>. Find all the edges that label a vertex adjacency which "starts" with u. Let these edges be $E_1, E_2, ..., E_m$, and let their "end" vertices be $v_1, v_2, ..., v_k, ..., v_{m-1}, v_m$ respectively.Each time $v_k = v$, k = 1, ..., m, we have found a path u —v of length 1. Mark each such edge and its vertex adjacency in E´ as a u — v path edge.

(3) Find all the unmarked edges in <A´, E´> that "start" with any $v_k \neq v$ among the vertex adjacencies found and marked in step (2). We now plot a tree as follows



from step (2), and then continue the development of the tree by inserting a separate branch between each $v_k \neq v$ of step (2) and the vertex $w_h \in A´$ for each edge by which v_k is adjacent with w_h.

If any of these vertices $w_h = v$ then we have now found all the u — v paths of length 2 in <A´, E´> ∠ <A, E>. Again mark all the edges and vertex adjacencies used in this step to find u — v paths of length 2, and proceed to step (4) with all the unmarked edges in E´and all those $w_h \in A´$ with $w_h \neq v$.

(4) Repeat step (3) for the next level of the tree, marking the edges and vertex adjacencies used in each stage of the generation of u — v paths of lengths 3, 4, ..., if any, until all the usable edges in E´ and their vertex adjacencies in <A´, E´> have been marked by this procedure. ♦

17. CRKH Theorems

Theorem 17.1: Any complete formal hyperschema <A, E> can be generated by a limited access cascade fom the set $B_0 \subseteq A$ of all the primaries of <A, E> iff every $a \in A$ is derivable in <A, E>, i.e. <A, E> is a CRKH. ♦

Proof: If <A, E> is generated from B_0 by a limited access cascade then, in each step of the cascade, every new vertex generated belongs to an edge $E_i \in E$ which is such that every vertex in E_i but the single new vertex, if any, is a primary or a vertex generated in a previous step. Thus for every new vertex v generated in step n of the cascade there is, at that stage, at least one path p — v, of length n, in <B_n, E_n>, and each such path has a betweenness sequence S in which every $t \in S$ is derived in terms of $X \subseteq A$ with t primary, or $t \in X$, or t immediately derived from a subset of S_t. Now if t is primary then t is trivially derivable from a set of hypotheses $X = \varnothing$ by a path of length zero. Next we notice that $X \subseteq B_{n-1}$, so if $t \in X$, and t is not primary here, then there is a path p´— t in <B_{n-1}, E_{n-1} > because <A, E> is a complete formal hyperschema, so t is not newly generated in <B_n, E_n> and this holds for all n = 2, 3, ---, so t is never generated. Thus we cannot have $t \in X$. Finally, if t is immediately derived from a subset of S_t then there is an $s \in S_t$ and an edge $E_i \in E_n \subseteq E$ such that we have, somewhere in <B_{n-1}, E_{n-1}>, a vertex adjacency (s, E_i, t) with every member of (E_i – {t}) a member of $S_t \subseteq (B_{n-1} - \{t\})$.
We have seen that every member of B_0 is a derived vertex. Suppose that every member of B_{n-1} in <B_{n-1}, E_{n-1}>, for all n = 1, 2, ..., n – 1, is a derived vertex in <A, E> and consider <B_n, E_n>. Now our set E_i – {t} is such that every one of its members is derivable by the induction hypothesis. But then, with (s, E_i, t), t is derivable in terms of hypotheses X = (E_i – {t}), and every member of X is derivable by the induction hypothesis, so t is derivable, and so every member of B_n is derivable in <B_n, E_n>. It follows that, because <B_n, E_n> = <A, E> for some n, every vertex $a \in A$ is derivable in <A, E>.
Conversely, suppose that every $a \in A$ is derivable in <A, E>. Then <A, E> can be generated by a limited access cascade from its set of primaries B_0 as follows. B_0 is the set of primaries of <A, E>, and $E_0 = \varnothing$. E_1 is the set of all edges $E_i \in E$ such that every member of E_i but one is a primary of <A, E>, i.e. a member of B_0. B_1 is the union of B_0 and all the new (non-primary) vertices generated in step 1 of the cascade. In general E_k, k = 2, 3, ..., is chosen in such a way that $E_i \in E_k \subseteq E$ iff all but possibly one member of E_i belong to B_{k-1}. B_k is B_{k-1}, in which every member is derivable in <B_{k-1}, E_{k-1}>, together with the set of all new vertices generated in step k. Eventually, for some n ∈ û, <B_n, E_n> = <A, E> because every $a \in A$ is derivable in <A, E> and the cascade generates only derivable new vertices in each step. ♦

Theorem 17.2: If $a \in A$ of a complete formal hyperschema <A, E> is derivable in terms of $X \subseteq A$, with $X = \varnothing$ or every $x \in X$ derivable in <A, E>, by virtue of a derivation path p — a, p a primary of <A, E>, and a betweenness sequence S for p — a, then every $t \in S$ is derivable in <A, E>. ♦

Proof: Since p is a primary it is derived by a derivation path of length zero with betweenness sequence $S = X = \varnothing$. Run a limited access cacade from the set B_0 of all primaries of <A, E>

in <A, E>. If p — a is a path of length n then we must "find" p — a in $<B_n, E_n>$ because a is derivable. Let an appropriate betweenness sequence for p — a, i.e. one which makes p — a a derivation path, be S and set X = S. Then, since a is derivable and S = X ≠ ∅, we see that every member of S is derivable in <A, E>. ♦

Theorem 17.3: Let <A, E> be a formal hyperschema with a ∈ A any non-primary vertex of <A, E>. If a is derivable in <A, E>, by virtue of a path p — a, then p is a primary vertex of <A, E>. ♦

Proof: We know that p — a is a derivation path. Let S be a betweenness sequence for p — a, and set X = S. Then a is derivable in terms of X, with X ≠ ∅ because a is non-primary, and every x ∈ X is derivable in <A, E>. For every t ∈ S = X, t is a primary or t ∈ X or t is immediately derived from a subset of S_t. In this case we clearly have t ∈ X trivially. Consider p. We have p is a primary in <A, E>, or p is immediately derived from a subset of $S_p = \varnothing$. Only primaries and isolates are immediately derivable from hypotheses ∅, by a trivial derivation path of length zero. Now p is certainly not an isolate, so in either case we have that p is a primary of <A, E>.♦

We now set out some corollaries of theorems 17.1, 17.2, and 17.3.

Corollary 17.1: If every t ∈ S in the proof of the theorem is derivable then every t ∈ S is immediately derived from some X ⊆ A in <A, E>. ♦

Proof: There are two cases to consider.
a) If t is on p — a then there is a vertex adjacency (x, E_i, t) on p — a, and then t is immediately derived from $X = (E_i - \{t\}) \subseteq A$.
b) If t does not lie on p — a, but is between p and a on p — a, we know that t is derivable from theorem 17.2. Thus there is at least one derivation path p´— t, p´ a primary of <A, E>, in <A, E>, and p´— t "ends" with a vertex adjacency (u, E_j, t). Let $X = E_j - \{t\}$, and we see that t is immediately derived from X. ♦

Corollary 17.2: If vertex a ∈ A of a complete formal hyperschema <A, E> is derivable in <A, E> then a is immediately derived from some X ⊆ A. ♦

Proof: If a is derivable in <A, E> then there must be some derivation path p — a for a, p primary, in <A, E>. Let the vertex adjacency with a on p — a be (x, E_i, a), x ∈ A and $E_i \in E$, and set $X = (E_i - \{a\})$. Then a is immediately derived from hypotheses X in <A, E>.♦

Corollary 17.3: A path p — a, a, p ∈ A, in a CRKH <A, E> is a derivation path for a iff p is a primary of <A, E>.♦

Proof: If p — a is a derivation path for a in <A, E> then p is a primary by theorem 17.3. Conversely, if p is a primary then any path p — a is a derivation path for a because, if S is a betweenness sequence for p — a then every member of S is derivable in <A, E> since <A, E>

is a CRKH, so we can see that a is derivable in <A, E> by p — a if we set X = S (=∅ if p = a). ♦

Corollary 17.4: If every path incident with a primary of a complete formal hyperschema <A, E> is a derivation path in <A, E>, then every a ∈ A is derivable in <A, E> and so <A, E> is a CRKH. ♦

Proof:Follows at once from the definitions of derivation path, derivable and CRKH. ♦

Corollary 17.5: Let <A, E> be a complete formal hyperschema, and let p be any primary of <A, E> and a be any non-primary of <A, E> such that there is a path p — a in <A, E>. Then p — a is a derivation path in <A, E>, i.e. a is derivable in <A, E>, iff every b ∈ A, b ≠ a, that is between p and a on p — a is derivable in <A, E>.♦

Proof: Let p — a be a derivation path with betweenness sequence S for p — a. b ∈ A, b ≠ a, is between p and a on p — a iff b ∈ S, and by theorem 17.2 every b ∈ S is derivable in <A, E>. Conversely, let every b ≠ a that is between p and a on p — a be derivable in <A, E>. Then b ∈ S, and if every member of S is derivable in <A, E> then a is derivable. But this means that a is derivable in terms of at least one X ⊆ A with X = ∅ or every member of X derivable in <A, E>, and at least one path from a primary to a must be a derivation path for a in <A, E>. Choose X = S ≠ ∅ for our path p — a and it follows that p — a is a derivation path for a in <A, E>.♦

Corollary 17.6: Let <A, E> be a complete formal hyperschema. Every a ∈ A is derivable in <A, E> iff every path p — a, p primary and a ∈ A, in <A, E> is a derivation path. ♦

Proof: The reverse implication is corollary 17.4. If every a ∈ A is derivable then there exists, by definition of the term derivable (from the set P of all primaries of <A, E>), at least one derivation path p — a, p primary, in <A, E>.♦

Corollary 17.7: A complete formal hyperschema <A, E> can be generated by a limited access cascade from the set of all its primaries iff every path incident with a primary of <A, E> is a derivation path. ♦

Proof: Follows at once from therem 17.1 and Corollary 17.6. ♦

Corollary 17.8: Let <A, E> be a complete formal hyperschema. Every a ∈ A is derivable in <A, E>, i.e. <A, E> is a CRKH, iff every a ∈ A is immediately derived from some set X_a of hypotheses which is such that every x ∈ X_a is a derived vertex in <A, E>.♦

Proof: If every a ∈ A is derivable then there is at least one derivation path p — a for a in <A, E>. Let (x, E_i, a) be the vertex adjacency with a that lies on such a path p — a, E_i ∈ E. Then a is immediately derived from $X_a = (E_i - \{a\})$. Conversely, let every a ∈ A be immediately derived from some set X_a of hypotheses such that every x ∈ X_0 is a derived vertex in <A, E>. Then there exists at least one vertex adjacency (x, E_j, a), E_j ∈ E, with

$(E_j - \{a\}) \subseteq X_a$. Now x is a derived vertex, as is every other member of X_a. Thus there is at least one derivation path p — x for some primary p, and we can concatenate p — x and (x, Ej, a) to make up a path p — a. Since every member of $E_j - \{a\}$ is derivable in <A, E>, we see by Corollary 17.5 that every b ≠ a in p — a is derivable.
Let S be an appropriate betwenness sequence for p — a, and set $X = S_a$. Then a is derivable in terms of X, i.e. derivable, because X ≠ ∅ but every x ∈ X is derivable in <A, E>.♦

Collecting some of the results of this section together, we have proved the following.

Theorem 17.4: Let <A, E> be a complete formal hyperschema. Then precisely the whole of <A, E> can be generated by a limited access cascade from the set B_0 of all the primaries of <A, E>

(1) iff every a ∈ A is derivable in <A, E>, which is true
(2) iff <A, E> is a CRKH, which is true
(3) iff every path p — a, p a primary and a ∈ A, is a derivation path in <A, E>, which is true
(4) iff every a ∈ A is immediately derived from some set $X_a \subseteq A$ of hypotheses which is such that every $x \in X_a$ is a derived vertex in <A, E>, which is true
(5) iff every b ≠ a that is between p and a, p a primary and a ∈ A, on every path p — a in <A, E> is derivable in <A, E>.♦

Running a limited access cascade from the set of all primaries in a complete formal hyperschema <A, E> provides an automated method of testing <A, E> for CRKH status.

Theorem 17.5: Let <A, E> be any hypernet, and let I [<A, E>] = <A, T>. <A, E> is a CRKH iff <A, T> is a CRKS. ♦

Proof: Let <A, E> be a CRKH, and consider a specific interpretation I [<A, E>] = <A, T>. Since I preserves vertex adjacencies, I will preserve all paths in <A, E>, mapping each path in <A, E> to a semi-path in <A, T>. Thus I preserves all derivation paths p — a, p a primary and a ∈ A, and each derivation path p — a is mapped to precisely one derivation path p → a in <A, T>. It follows that <A, T> is a CRKS. Conversely, let <A, T> be a CRKS with <A, T> = I [<A, E>] for some hypernet <A, E>. Let M be the inverse of I, so that M [<A, T>] = <A, E>. Then, since M preserves all vertex adjacencies in <A, T>, it preserves all semi-paths, mapping each semi-path in <A, T> to precisely one path in <A, E>. It follows that every derivation path p → a, p a primary and a ∈ A in <A, T> is mapped to precisely one derivation path p — a in <A, E>. It follows that <A, E> is a CRKH. ♦

The theorem is essential to a generalization of Part I of [GVS99].

Definition 17.1: By a ***derivation adjacency*** in a formal hyperschema <A, E> we mean a vertex adjacency (a, E_i, b), a, b ∈ A and $E_i \in E$, that lies on a derivation path for b in <A, E> and is such that every $x \in (E_i - \{b\})$ is either a primary of <A, E> or belongs to a derivation adjacency (y, E_j, x) that lies on a derivation path for x in <A, E>, i.e. every $x \in (E_i - \{b\})$ is derivable in <A, E>.♦

Theorem 17.6: Let <A, E> be a CRKH. Then every vertex adjacency (a, E_i, b), a, b ∈ A and E_i ∈ E, in <A, E> is a derivation adjacency of <A, E>. ♦

Proof: Consider an arbitrary vertex adjacency (a, E_i, b) in <A, E>. Since <A, E> is a CRKH both a and b are derivable in <A, E>. Then either (a, E_i, b) is on a derivation path for a in <A, E>, or it is on a derivation path for b in <A, E>. Suppose, without loss of generality, that (a, E_i, b) lies on a derivation path for b. Then (a, E_i, b) is a derivation adjacency because every x ∈ (E_i – {b}) is derivable in <A, E>.♦

We now begin to turn our attention to the sort of uses of CRKH's outlined for CRKS's in [GVS99].

Definition 17.2: Given a CRKH <A, E> and any non-primary a ∈ A, we define a ***derivation path hyperschema*** D(p — a) for a derivation path p — a in <A, E> to be a sub-hypernet of <A, E> that

(1) contains p – a and
(2) is a formal hyperschema in which the only primaries and isolates are all primaries of <A, E> and in which every non-isolate is derivable, and
(3) is minimal in the sense that p — a is not a derivation path in any sub-hypernet produced from D(p — a) by deleting from it any vertex or any edge. ♦

The primaries and isolates of D(p — a) are all regarded as singleton edges in D(p — a). We should notice that a derivation path hyperschema for a ∈ A in <A, E> is not generally unique because there may be several derivation paths for a in <A, E>.

Definition 17.3: Given a CRKH <A, E> with a ∈ A, we define the ***predecessor hyperschema*** P(a) of a in <A, E> to be that sub-hypernet of <A, E> that is generated by running a fast access cascade in the reverse of the direction of derivation from B_0 = {a} in <A, E> as follows: $E_0 = \varnothing$. <B_1, E_1> contains all the derivation adjacencies, incident with a, through which a is derived, i.e. that lie on any derivation path for a in <A, E>. This fixes E_1, and B_1 is a together with the set of all the vertices in all the members of E_1. <B_2, E_2> contains all the derivation adjacencies incident with each b ∈ B_1 and through which b is derived in <A, E>, which specifies E_2, and B_2 is B_1 together with the set of all vertices in all the members of E_2, and so on. The cascade will stop with a primary, or primaries, of <A, E>. It is clear that P(a) is a CRKH with goal a and set of primaries a subset of the set of primaries of <A, E>.♦

It is easy to show that the next theorem follows from the definitions above.

Theorem 17.7: Given a CRKH <A, E> with a ∈ A, the join of all the D(p — a) in <A, E>, p some primary of <A, E>, is a sub-hyperschema of P(a). ♦

The converse of the theorem is not generally true, as can be shown by simple counter examples – see [GVS99].

Definition 17.4: Let <A, E> be a CRKH and E_i ∈ E an edge of <A, E>. By a ***hypercluster*** for E_i we mean any minimal sub-CRKH, of <A, E>, that has E_i as one of its edges, where by minimal we mean that if we delete any vertex or edge from a hypercluster then the resulting hypernet does not have E_i in it. ♦

A hypercluster for a given $E_i \in E$ in a CRKH <A, E> is not generally unique.

Constructional schemes to find the D(p — a), and P(a), in a CRKH <A, E> are easily adapted from [GVS99]. Definitions 17.2, 17.3 and 17.4 are important in the modelling of study material, as can be seen from [GVS99]. In this case, the case of hypernets, their application potential is broader than for the CRKS's of [GVS99].

Theorem 17.8: C is a cluster for $T_i \in T$ in a CRKS <A, T> iff D is a hypercluster for $E_i = I [T_i]$ in a CRKH <A, E>, where <A, T> = I [<A, E>] and C = I [D] for some interpretation I. ♦

Proof: Follows easily from the definition of an interpretation and its inverse. ♦

18. Gauges of complexity

In this section we present some ways to gauge the complexity of a CRKH.

Definition 18.1: The ***vertex context number*** of $a \in A$ in a CRKH <A, E> is given by $Vc(a) = |A[a]|$ and the ***edge context number*** of a is given by $Ec(a) = |E[a]|$, where <A[a], E[a]> = <A,E>[a] is the context hyperschema of a in <A, E>.♦

Definition 18.2: By the ***degree*** d(a) of $a \in A$ in a CRKH <A, E> we mean the sum of all the $|\lambda(\{a, b\})|$ over all $b \in A$ for which $\lambda(\{a, b\}) \neq \emptyset$. By the ***in-degree*** id(a) of a we mean the sum of all the $|\lambda(\{a, b\})|$ over all $b \in A$ for which $\lambda(\{a, b\}) \neq \emptyset$ and (a, E_i, b), E_i some edge of <A, E> which is such that (a, E_i, b) lies on a derivation path for a in <A, E>. By the ***out-degree*** od(a) of a we mean the difference od(a) = d(a) – id(a). ♦

Definition 18.3: By the ***flow*** at $a \in A$ in a CRKH <A, E> we mean the number $f(a) = \min\{id(a), od(a)\}$. ♦

Definition 18.4: By the ***path-multiplicity*** at $a \in A$ in a CRKH <A, E> we mean the number $p(a) = id(a) * od(a)$. ♦

Definition 18.5: By the ***local context number*** of $a \in A$ in a CRKH <A, E> we mean $|\cup (E_i - \{a\})|$ where the union is taken over all $E_i \in E$ with $E_i \in \lambda(\{a, b\})$ and $b \in A$. ♦

So far all our gauges should have relatively high values in any CRKH model of a "real world" situation. Relatively low values will indicate a weakness of association among vertices.

Definition 18.6: Let <A, E> be a CRKH, and let $S \subseteq A$ with $S \neq \emptyset$. The ***rank of S***, r(S), in <A, E> is defined by $r(S) = \max |S \cap E_i|$ over all the $E_i \in E$. The number r(A) is called the ***rank of <A, E>***. ♦

Definition 18.7: Let <A, E> be a CRKH. A sub-family $E_M \subseteq E$ is called a ***matching*** if the edges of E_M are pairwise disjoint. ♦

Definition 18.8: A ***transversal*** of a CRKH <A, E> is a set $T \subseteq A$ such that $T \cap E_i \neq \emptyset$ for all $E_i \in E$. The ***transversal number*** of <A, E> is the minimum number of vertices in any transversal of <A, E>.♦

Of interest for CRKH's are maximum matchings, which tell us something about "essential" edges in the case in which "knowledge" is being modelled and we have $\cup E_i = A$ where the union is taken over the edges of E_M, and the transversal number which tells us how many "essential" vertices belong to A.

Definition 18.9: Let <A, E> be a CRKH, and consider a limited access cascade from the set of all primaries of <A, E>. The ***deductive distance dd(a) of a $\in$ A from the primaries*** of

<A, E> is n iff a is first found in <B_n, E_n>, i.e. in the (n+1) ‘ th step of the cascade, i.e. $a \notin B_{n-1}$. By an ***n-slice*** of <A, E> we mean the set of all $a \in A$ that are first found in <B_n, E_n>, i.e. in the (n+1) ‘ th step of the cascade, i.e. $a \in (B_n - B_{n-1})$. Let $N_n \subseteq A$ be an n-slice of <A, E>, and let $a \in N_n$. Then the ***weighted deductive distance***, wd(a), of a from the primaries of <A, E> is defined by wdd(a) = È N_i where the union is taken over $i \in \{0, 1, ..., n - 1\} = n \in$ û. ♦

We would, in most applications, not want dd(a) or wdd(a) to be relatively large compared to their values for other vertices of <A, E>.

Definition 18.10: Let $a \in A$ of a CRKH <A, E> belong to an n-slice N_n in <A, E> for some $n \in$ û. Then $|N_n|$ is called the ***width*** W(a) of <A, E> at a. ♦

Associated with the rank of a set $S \subseteq A$ of a CRKH <A, E> is the following.

Definition 18.11: Let <A, E> be a CRKH, and let $P \subseteq A$ be the set of primaries of <A, E>. By the ***scope*** of a set $B \subseteq A$ in <A, E> we mean the set $S(B) \subseteq E$ defined by $S(B) = \{E_i \in E \mid S \cap E_i \neq \emptyset\}$. By the ***scope number*** of $B \subseteq A$ in <A, E> we mean $|S(B)|$. S(P) is called the ***primary scope*** of <A, E>, and $|S(P)|$ the ***primary scope number***. ♦

We would like the primary scope number to be relatively high – it is at least $|P|$ -, and if S(B) is relatively low then B is relatively weakly associated with other members of A. If B = {a} then S({a}) = E[a].

Definition 18.12: Let <A, E> be a CRKH with $E_i \in E$ and $S \subseteq A$. The ***edge rank*** Er(S) of E_i with respect to S is defined by $r(S, E_i) = |S \cap E_i|$. ♦

Definition 18.13: By a ***vertex covering*** C of a CRKH <A, E> we mean a sub-family $C \subseteq E$ such that the union of all the edges in C is A. ♦

We would be interested in minimal vertex coverings, again a measure of “essential” vertices in <A, E>.

Minimum traversals and maximum matchings are fairly closely related – see [Ber73].

Next we turn to analysis of a CRKH <A, E> by means of edge ranks in order to illustrate one use of some of our gauges. Run a limited access cascade from the set B_0 of all the primaries of <A, E>, setting $E_0 = \emptyset$ as usual. Suppose we have completed step n of the cascade, i.e. we have <B_n, E_n> ∠ <A, E>. $(B_n - B_{n-1})$ is an n-slice, of <A, E>, with width $|B_n - B_{n-1}|$. Now complete step n+1 of the cascade, producing <B_{n+1}, E_{n+1}>, and consider $(E_{n+1} - E_n)$. Let $E_i \in (E_{n+1} - E_n)$ and let edge rank 1 be given by $r_1((B_{n+1} - B_n), E_i) = |(B_{n+1} - B_n) \cap E_i|$. This is the number of “new” vertices found in step (n+1) that belong to E_i, a “new” edge found in step (n+1). Let the equivalence class of E_i in $(E_{n+1} - E_n)$ induced by the rank 1 value of E_i be denoted by $r_1[(B_{n+1} - B_n), E_i]$. We now partially order these equivalence classes, from the smallest to the largest, by r_1 value. Call the r_1 value of each class the ***r_1-difficulty*** of that class.

Next consider any one of these classes. Inside $r_1[(B_{n+1} - B_n), E_i]$ we define another equivalence relation on this set of edges, all of which have the same edge rank 1 value, as follows, looking now at the "dependence" of these edges on the vertices in $(B_n – B_0)$. Let edge rank 2 be $r_2((B_n – B_0), E_j)$, where $E_j \in r_1[(B_{n+1} - B_n), E_i]$. The r_2 values specify equivalence classes $r_2[(B_n – B_0), E_j] \subseteq r_1[(B_{n+1} - B_n), E_i]$. Every member of any of these equivalence classes has the same r_2 value, and we partially order these r_2 equivalence classes, inside $r_1[(B_{n+1} - B_n), E_i]$, from smallest to largest r_2 value, the relevant r_2 value being called the ***r_2-difficulty*** of the associated equivalence class.

Next consider an $r_2[(B_n – B_0), E_j]$. Inside this eqivalence class we define a third equivalence relation as follows. Let edge rank 3 be defined by $r_3(B_0, E_k)$, with $E_k \in r_2[(B_n – B_0), E_j]$. This specifies eqivalence classes $r_3[B_0, E_k] \subseteq r_2 [(B_n – B_0), E_j]$. Again of course every member of $r_3[B_0, E_k]$ has the same r_3 value, and again we partially order these edge rank 3 eqivalence classes from smallest to largest r_3 value. This r_3 value is called the ***r_3-difficulty*** of the relevant class.

Now we can choose an equivalence class of minimal r_1 value, then one, inside that class, of minimal r_2 value, and then one, in that r_2 class, of minimal r_3 value. This allows us to choose those $E_i \in E_{n+1}$ of ***minimal difficulty*** (to learn – see [GVS99]) and work through each r_1 eqivalence class from minimal to maximal difficulty in <B_{n+1}, E_{n+1}>.

Finally, consider any given interpretation I [<A, E>] = <A, T> of the CRKH <A, E>. Clearly <A, T> is a CRKS (from the definition of interpretation). Now consider I $(E_i) = T_i$, $E_i \in$ E and $T_i \in$ T. The number of entries in T_i, call it the ***length*** of T_i, is at least $|E_i|$. We partially order the edges of each $r_3[B_0, E_k]$ from smallest to largest tuple length of the I $[E_{\oplus}]$, $E_{\oplus} \in r_3[B_0, E_k]$, regarding those edges corresponding with minimal length tuples to be the least difficult in $r_3[B_0, E_k]$. This defines eqivalence classes in each $r_3[B_0, E_k]$, each being characterized by a tuple length value called the ***r_4-difficulty*** of the class. We do the same in each $r_2[(B_n – B_0), E_j] \supseteq r_3[B_0, E_k]$, and then in each $r_1[(B_{n+1} - B_n), E_i] \supseteq r_2[(B_n – B_0), E_j]$, using r_4-difficulty to partially order edges in each equivalence class at each r_3, r_2 and r_1 level in turn. We can use the values of all four gauges, r_1, r_2, r_3 and r_4, to partially order all the tuples in any CRKS <A, T> = I [<A, E>] from "least difficult" subset of T to "most difficult" subset of T, providing us with a tuple-ordering strategy in presenting <A, T> - see [GVS99].

19. Structural analysis of a CRKH

We now turn to structural characteristics of CRKH's. These are similar to those exposed in the chapter on presentation strategies in [GVS99].

(1) The most basic structural characteristics of a CRKH are its vertex basis and its edge bases. The set of primaries of a CRKH is its unique vertex basis, and, in the terminology of graph theory, the set of goals of a CRKH is its unique vertex contrabasis.

(2) Application of Menger's Theorem in a CRKH yields two interesting insights into the structure of a CRKH. Let K = <A, E> be a CRKH with set of primaries P and set of goals G. Convert K to a CRKH Z as follows: Delete from K all egdes that consist of only a primary and a goal or that consist of only primaries and a goal. Next add dummy vertices π and γ to K, and add new dummy edges $\{\pi, p\}$ for each $p \in P$ and $\{\gamma, g\}$ for each $g \in G$. This completes the construction of $Z = Z_0$. The set of all π — γ paths, in Z, that have a given vertex v_0 of K between π and γ is called a ***bundle*** of π — γ paths and is denoted by S_0. Every member of S_0 is cut by deletion of v_0 from K. Consider a minimal separation $B(\pi — \gamma)$ for π and γ in K and let $B(\pi — \gamma) = \{v_0, v_1, ..., v_n\}$. Deleting the context-hyperschema of v_0 from K deletes all the members of bundle S_0, deleting that of v_1 deletes the set S_1 of all π — γ paths in what remains of K from that remaining hypernet, i.e. all the π — γ paths in $\langle A – \{v_0\}, E\uparrow(A - \{v_0\})\rangle$ that have v_1 between π and γ in K, and so on, producing a partition of all the π — γ paths in Z into n bundles. Two π — γ paths P_r and P_t are said to be ***quasi-disjoint*** iff they belong to two distinct bundles. Then Menger's Theorem states that the maximum number of quasi-disjoint π — γ paths in Z is equal to min $|B(\pi — \gamma)|$ - see the chapter on Menger's Theorem in this report, and the chapter on presentation strategies and section 12.5 in [GVS99].
The paths deleted from K in constructing Z are all of length 1 and are easy to deal with separately. Since two quasi-disjoint π — γ paths can share a vertex v of K, i.e. some v may be between π and γ on both paths, we introduce the following. Two π — γ paths are said to be ***independent*** iff (i) they are quasi-disjoint and (ii) no vertex v of K is between π and γ on both paths. It is easy to see that if the two paths are independent then they are quasi-disjoint, but a simple counter example will show that the converse is not generally true. ♦

Definition 19.1: A set of pairwise independent π — γ paths in Z is called a ***flow***, and the ***measure*** of a flow is defined to be the number of paths of the flow. ♦

Theorem 19.1: The measure of a maximum flow for π and γ through Z is less than or equal to min $|B(\pi — \gamma)|$. ♦

Proof: Follows from Menger's Theorem for Z and the fact that independent paths are quasi-disjoint, but the converse is not necessarily true, so there cannot be more paths in a flow than there are pairwise quasi-disjoint π — γ paths in Z. ♦

.

The members of a minimal vertex separation B(π— γ) in Z are critical in K, as are the paths in a maximim flow, in some applications. Dealing with the paths of length 1 that were deleted from K to produce Z, if any, is easy after applying the theorem.

Menger's Theorem also applies in edge form, as briefly outlined below. By an ***edge separation*** E (π—γ) for π and γ in Z we mean a set of edges of K which, if deleted from Z, will leave no π— γ paths in Z. By an ***edge-bundle*** in Z we mean the set of all π — γ paths that use a particular edge of K. Pick an edge e_0 of K. Let edge-bundle S_0 be the set of all π— γ paths in Z that use e_0. Delete from Z the common edge, e_0, of each of the members of S_0. Repeat this process in what remains of Z, defining bundle S_1 for edge e_1. Continue until no more π — γ paths remain. Two π — γ paths are said to be ***quasi-edge-disjoint*** iff they belong to two distinct edge-bundles. Now Menger's Theorem states that the maximum number of pairwise quasi-edge-disjoint π — γ paths in Z is equal to the minimum number of members in an edge separation E(π—γ) in Z, i.e. min$|$E(π — γ)$|$.

Since two quasi-edge-disjoint paths can share an egde of K, we define the following notion. Two π — γ paths in K are said to be ***edge-independent*** iff

(1) they are quasi-edge-disjoint and
(2) no edge of K lies on both π — γ paths. If two π — γ paths are edge-independent then they are quasi-edge-disjoint, but the converse is not generally true.

Definition 19.2: A set of pairwise edge-independent π — γ paths in Z is called an ***edge-flow***, and the ***measure*** of an edge-flow is defined to be the number of π — γ paths in the edge-flow. ♦

Theorem 19.2: The measure of a maximum edge-flow for π and γ through K is less than or equal to min $|$E(π—γ)$|$. ♦

Proof: Follows from the edge version of Menger's Theorem for Z and the fact that edge-independent π — γ paths are quasi-edge-disjoint but the converse is not necessarily true, so there cannot be more paths in an edge-flow than there are pairwise quasi-disjoint π — γ paths in Z. ♦

Can we get closer to the measure of a flow? Consider Z, and partition the set of all π — γ paths in Z as follows. Delete any vertex v_0 of K from Z, and let S_0 be the set of all π — γ paths in Z that are cut by that deletion. Let <B_0, E_0> ∠ K be the hypernet that is defined to be the context hyperschema of all the vertices of K that are between π and γ on any π — γ path in S_0, i.e. <B_0, E_0> is the join of all the context hyperschemas of each vertex of K that is between π and γ on any π — γ path in S_0. Delete <B_0, E_0> from Z, and let <B_1, E_1> be the sub-hypernet of Z that remains after this deletion. Choose any $v_1 \in (B_1 - \{\pi, \gamma\})$, delete v_1 from <B_1, E_1>, and let S_1 be the set of all π — γ paths in <B_1, E_1> that are cut by that deletion. Now delete from <B_1, E_1> the context-hyperschema of all the vertices of <B_1, E_1> that are between π and γ on any π — γ path in S_1. Continue in this way, defining S_r for r = 0, 1, ..., t, until S_{t+1} is empty. Then a flow of measure (t + 1) can be found by choosing precisely one

π — γ path from each S_r. The set of vertices v_r, r = 0, 1, ..., t is an example of what is said to constitute a ***flow-separation*** F(π — γ) for π and γ in Z, and we clearly have:

Theorem 19.3: The measure of a maximum flow for π and γ through K is equal to min $|F(\pi — \gamma)|$. ♦

Can we do a similar thing for edge-flows? We can indeed. Delete every edge of every member of S'_0, where S'_0 is the set of all π — γ paths of Z that are cut by the deletion of edge e_0 from K. Next choose any edge e_1 of K that remains after the deletion of all edges of all the paths in S'_0. Let S'_1 be the set of all π — γ paths, in what remains of Z, if any, that are cut by the deletion of e_1 from the remaining hypernet, and then delete from that remaining hypernet all the edges of every member of S'_1. Continuing in this way we partition all the π — γ paths in Z into sets $S'_0, S'_1, ..., S'_n$. Now two π — γ paths are edge-independent iff they belong to two distinct S'_i, because the two paths are certainly quasi-edge-disjoint and they can share no edge of K.

Thus we have

Theorem 19.4: The meaure of a maximum edge-flow for π and γ through Z is equal to min $|G(\pi — \gamma)|$, where G(π — γ) is an edge-flow-separation for π and γ in Z, i.e. G(π — γ) is a set of edges such as $e_0, e_1, ..., e_n$ that generate a partition of π — γ paths such as $S'_0, S'_1, ..., S'_n$ respectively. ♦

Since deletion of vertices of K is more destructive than deletion of edges from K in general, because of strong vulnerability, we have the following.

Theorem 19.5: If two π — γ paths P_1 and P_2 in Z are independent then they are edge-independent, but the converse is not generally true. ♦

Proof: Since P_1 is independent of P_2, P_1 and P_2 are quasi-disjoint, and P_1 and P_2 share no vertex of K, i.e. no vertex of K is between π and γ on both P_1 and P_2. Since P_1 and P_2 are then vertex-disjoint, they must clearly be edge-disjoint, so they are edge-independent because they belong to different edge bundles: Edge-disjoint implies quasi-edge-disjoint, but the converse is not true in general. If P_1 and P_2 are edge-independent then they may clearly share a vertex of K, so they are not, in general, independent π — γ paths. ♦

Corollary 19.1: min $|F(\pi — \gamma)| \leq |G(\pi — \gamma)|$ in Z. ♦

Proof: Follows at once from Theorem 19.5. ♦

Since deleting the context-hyperschema of all vertices in all the π — γ paths on which some vertex v lies is more destructive than deleting only the context-hyperschema of v, we have:

Theorem 19.6: min $|F(\pi — \gamma)| \leq |B(\pi — \gamma)|$. ♦

Since deleting all the edges of S'_i is more destructive than deleting just the generating edge e_i, we have:

Theorem 19.7: min $|G(\pi — \gamma)| \leq |E(\pi — \gamma)|$. ♦

Finally, for the same reason, we have

Theorem 19.8: min $|B(\pi — \gamma)| \leq$ min $|E(\pi — \gamma)|$. ♦

Thus we have

Corollary 19.2:

min $|F(\pi — \gamma)| \leq$ min $|G(\pi — \gamma)| \leq$ min $|E(\pi — \gamma)|$

and

min $|F(\pi — \gamma)| \leq$ min $|B(\pi — \gamma)| \leq$ min $|E(\pi — \gamma)|$. ♦

Facets of Menger's Theorem will be useful in some applications inasmuch as they separate out certain vertices, edges and derivation paths for special attention.

(3) **Matchings** and **Coverings** re-visited. In Chapter 5 of [GVS99] we discussed a variety of presentation strategies, and this section of the report picks up some of that work, but with a different emphasis. Before continuing with this section, we look again at matchings and coverings as both are important facets of the structure of a CRKH. One of the key approaches to finding matchings is the construction of a bipartite graph G from a CRKH <A, E> as follows. Order the edges of <A, E> in any way, and plot them as vertices of G in two columns $E_1 = E$ and $E_2 = E$, each in the defined order. Join two distinct vertices of G, $v_1 \in E_1$ and $v_2 \in E_2$, that are adjacent by at least one vertex $a \in A$ in <A, E>. From this graph G one can write an algorithm to find a matching in <A, E>, where we recall that a matching is defined as follows.

Definition 19.3: A ***matching*** $M \subseteq E$ in a CRKH <A, E> is a set of edges of <A, E> that are pairwise (potentially) non-adjacent. M is a ***maximal matching*** iff we can add no edge of <A, E> to M without destroying the matching property. ♦

It is easy to find a maximal matching, in <A, E>, using G – see [GVS99] p. 74 for example. The members of a maximal matching are pairwise "independent" edges inasmuch as no two of them are adjacent edges in <A, E>. A relatively large value of $|M|$ compared with $|E|$ will indicate a certain poverty of derivation paths, so maximal matching can be important in analysing the structure of <A, E>. Now recall vertex covering.

Definition 19.4: A ***vertex cover*** of a CRKH <A, E> is a set of edges $E_c \subseteq E$ which is such that È E_i, $E_i \in E_c$, is equal to A. A ***minimal vertex cover*** of <A, E> is a set of edges that, together, involve each $a \in A$ at least once, and from which we may delete no edge without destroying the covering property. ♦

If we find a maximal matching in <A, E> then we can convert it to a minimal vertex cover – see [Ber89]. A minimum cover will tell us the minimum number of edges that "say something" about each $a \in A$ in <A, E>, and presents us with a set of edges that actually does this. Constructional Scheme 5.4 in [GVS99] can easily be re-written to find a minimal vertex cover for <A, E>.

(4) Next we turn to the CRKH equivalent of a **tuple oriented partial presentation strategy**, not dealt with in [GVS99] but sometimes relevant for structural analysis of a CRKH. Let <A, E> be any CRKH.

Definition 19.5: By a ***primary edge*** of <A, E> we mean an $E_i \in E$ such that every member of E_i, but precisely one, is primary in <A, E>, and that one other vertex is non-primary in <A, E>.♦

(1) Let L_0 be the set of all primary edges of <A, E>, and there must of course be at least one. Now we start to describe a procedure in terms of our bi-partite graph G. Mark the members of L_0, in E_1 and in E_2, in G, and then delete all edges of G that link members of L_0, i.e. represent adjacencies of members of L_0.
(2) Define $L_1 \subseteq E$ as follows. A vertex $E_i \in E_1$ (and of E_2) in G belongs to L_1 iff it is adjacent with at least one member of L_0 in G. Delete all edges of G that link members of L_1, i.e. represent adjacencies of members of L_1. Now partially order the members of L_1 as follows.Let the ***order*** of each $l_1 \in L_1$ be $|l_1|$, and arrange the members of L_1 in partial order of decreasing order, those with maximum order being said to be ***closest*** to L_0 because they are, among the members of L_1, most closely associated with the vertices involved in the members of L_0.
(3) Repeat step 2 with L_0 replaced by L_1 and L_1 replaced by L_2, then with L_2 and L_3, and so on until L_k has been defined and we then find $L_{k+1} = \emptyset$. We have then dealt with some of the edges of <A, E> in a partial order that consists of succesive steps with a partial ordering of edges in each step.
(4) Finding the "strongest" associations of edges, in each step, with edges in all the previous steps can be another indication of the strength of association in a CRKH. It is clear that one can define a partial presentation strategy, i.e. a hierarchy of nested sub-hypernets of <A, E>, along these lines. In practice È $L_i \subseteq E$ may constitute only a very small subset of E, but we can condsider it as displaying "core associations" among (some of) the vertices of <A, E>.

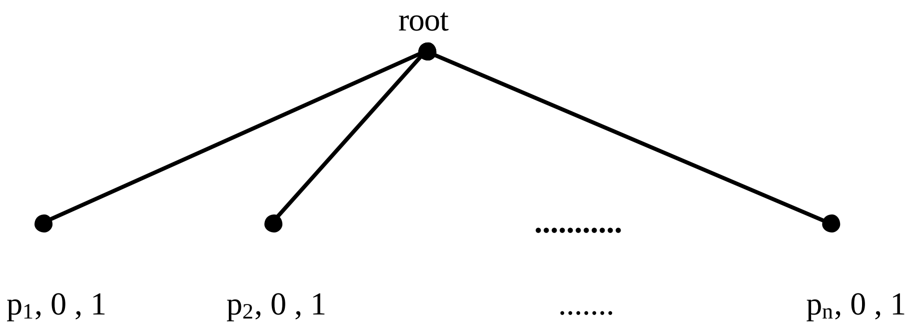
Another indication of the kind of association that should be examined in a CRKH <A, E> is the case of ***spiralling*** – see [GVS99]. Here we can regard this as a way of sorting knowledge about $a \in A$ if spiralling occurs for a (as it often does). Suppose that we have, in the predecessor hyperschema P(a) of $a \in A$, a sub-hyperschema that contains at least one derivation path for a that does not use a, i.e. a is not between the relevant primary and a on this path other than as the "end" vertex of that path, and at least one derivation path for a that does use a "on the way to a". The minimum sub-hyperschema of P(a) that contains the join of the derivation path hyperschemas of all such paths in P(a) is then said to constitute a recursive, or bootstrap, approach to a in P(a), and thus in <A, E>. It is called the ***recursive sub-hyperschema*** of a in <A, E>, and it contains at least one derivation path hyperschema, for a, that does not use a, and at least one that does. Knowledge about a $\in$ A in <A, E> is first to be found in the recursive sub-hyperschema for a in <A, E>, if one exists, starting with those derivation paths that terminate at a but do not use a anywhere else in them, thus establishing preliminary knowledge of a in <A, E>. Then the other derivation paths in the recursive sub-hyperschema can be dealt with, and then P(a), and then finally the context hyperschema of a in <A, E>. This provides us with a graded approach to finding all the knowledge about a in <A, E>. Constructional Scheme 5.5 in [GVS99] can easily be transcribed to provide a way of finding the recursive sub-hyperschema of a $\in$ A in <A, E>. A recursive sub-hyperschema is unique.

Deductive Complexity of a CRKS – see [GVS99] – can be usefully transcribed to a CRKH. It is clear that a limited access cascade from the primaries of a CRKH <A, E> generates a hierarchy, in <A, E>, in the form of a nested sequence of sub-hypernets of <A, E>. We will be concerned with that hierarchy and the notion of deductive distances in <A, E>, which we recall here.

Definition 19.6: The ***deductive distance*** from the primaries of a CRKH <A, E> of a $\in$ A is defined by dd(a) is the level of a in <A, E>, where that level is the step number in a limited access cascade from the primaries of <A, E>, in <A, E>, in which a is first encountered in that cascade. ♦

The primaries of <A, E> constitute B_0, so they are in level zero of the cascade, so dd(p) = 0 for every primary of <A, E>. Next we recall constructional scheme 16.1. In it we showed how to construct a tree that displays every path from each primary of <A, E> as a unique path in that tree. Now we label that tree, as we construct it, by marking all its branches and nodes in a way that allows us to compute what we call the ***deductive complexities*** DCOM. Again we refer to vertices and edges of <A, E>, and to nodes and branches of the ***path tree***.

First we introduce an unlabelled dummy node to serve as the root of the tree, and one only node for each primary of <A, E>. Each such node is joined to the root by an unlabelled branch. Every node, other than the root, is labelled with (concept-name, deductive distance of the vertex represented by that node, deductive complexity DCOM of that node). So far we have



for the n primaries of <A, E>, where $dd(p_i) = 0$ for every primary and we set DCOM $(p_i) = 1$ for every primary. For each node for a vertex $u \in A$, the path tree now develops as follows. Find every edge E_i by which there is a vertex adjacency (u, E_i, v) where $E_i = \{u = c_1, c_2, ..., c_m = v\}$. We plot a new node for vertex v for each edge $E_j \in E$ by which there is a vertex adjacency (u, E_j, v) for this u and v, and insert a branch from each node for u to every node for v. Each such branch is labelled with the index k of the edge E_k that generates it, together with all the members of E_k other than the two vertices which are adjacent by E_k in <A, E>. Thus, for our example E_i above, we would get a branch from each node for u to every node for v in the path tree, and that branch would have label i; $c_2, c_3, ..., c_{m-1}$, where any order of the c_s will do. Each new node for v is labelled with its concept-name, its deductive distance from the primaries of <A, E>, and the node value of DCOM. The node value of DCOM is computed from the edge that generates the particular, unique, branch to that node by setting DCOM = DCOM for the "beginning" node of that branch + Σ (DCOM of the node for c_s) from s = 2 to m-1 over all the c_s written along that branch in the branch label. We set $dcom(c_s)$ equal to any minimal value of DCOM of a node for the vertex c_s. In the case of an edge {u, v}, the branches between u and v for this edge are all labelled with the index of this edge and the set $\emptyset$ of vertices, and for such a branch we set DCOM for the end node of the branch, i.e. the one furthest from the root, to DCOM for the beginning node of that branch +1.

Next we number the nodes of the path tree. Number the root zero, and then number all sons from left to right. Now we assign a value of dcom for each concept-name that appears in any branch label as follows. Fill in DCOM for each node that has dd = 1. Certainly this is possible because all the primaries have dd = 0 and every node at dd = 1 represents a vertex that was derived in terms of primaries only. Next, proceed to nodes for vertices at dd = 2, then at dd = 3, and so on in turn, using the following method. For each concept-name v in a branch label, look in the path tree for any node for v that has a minimal value of DCOM among those nodes. Suppose that we choose node number n for v: Then dcom(v) = DCOM(n), and wherever v occurs in any branch label we enter dcom(v) and (n) next to v in that label. To see that this assignment of values of DCOM is possible for all the non-root nodes of the path tree, consider the following informal argument. In level 0 we have all the primaries, and each primary has a node for which DCOM = 1. Since each primary is trivially derived by a derivation path of length zero, we must set dcom = DCOM = 1 for each node for a primary. This takes care of the first stage of filling in DCOM and dcom. We now temporarily define a ***first derivation path*** for any non-primary vertex v of <A, E>, in <A, E>, as follows. Suppose that v is in level n, $n \geq 1$, in <A, E>. A first derivation path for v is any derivation path for v, in <A, E>, for which every vertex u used on that derivation, i.e. in an edge of that derivation path, is in a level $m < n$.

Let v be any vertex, of <A, E>, that lies in level 1, and let D(v) be any first derivation path for v in <A, E>. Then the only vertices of <A, E> that are used in reaching v by means of D(v) are primaries of <A, E>, and this includes the case of $\emptyset$ labels. It follows that we can assign a value of DCOM to that node copy of v that lies at the "end" of the unique path, in the path tree for <A, E>, which corresponds with this first derivation path D(v) for v. Notice that there must be at least one first derivation path in <A, E> for every $v \in A$ in any given level, because <A, E> can be precisely generated by a limited access cascade from its primaries. We now assign a value of DCOM to the relevant node copy of v for every first derivation path for v. Any minimal value of DCOM assigned to a node copy of v in the path tree using this procedure for v ∈ A can be chosen to be the value of dcom for v, and this value is now fixed for v so we fill it in, together with the number of the chosen node copy of v, at every occurrence of v in a label in the path tree. We do this for every v ∈ A that lies in level 1, and this is possible because each such vertex has at least one first derivation path, in <A, E>, that involves only primaries, possibly with a ∅ label, in reaching that vertex.

Next suppose that we are done with all level n vertices of <A, E> for some n ≥ 1. Thus every vertex of <A, E> that lies in level m ≤ n has been associated with at least one value of DCOM and with a single value of dcom. Let v now be any vertex of <A, E> that lies in level (n+1) in <A, E>, and let D(v) be any first derivation path for v in <A, E>. The only vertices of <A, E> that are used in reaching v by means of D(v) are vertices u in levels m ≤ n, so each such vertex u is associated with some node copies for each of which we have a value of DCOM, and all those copies have the same previously chosen value of dcom. It follows that we can now compute a value of DCOM for that node copy of v which lies at the "end" of the unique path, in the path tree of <A, E>, that corresponds with this first derivation path D(v) for v. We do this for each first derivation path for v. Any minimal value of DCOM associated with some node copy of v in the path tree using this first derivation path procedure for v can be chosen to be the value of dcom for v and attached to every occurrence of v in a branch label of the path tree, together with the number of the node copy of v which was chosen in assigning the value of dcom to v. We repeat this for every vertex of <A, E> that lies in level (n+1): This is possible because each such vertex has at least one first derivation path that involves only vertices in levels m ≤ n, and possibly ∅ labels, in reaching that vertex, and at least one such path must exist because <A, E> can be precisely generated by a limited access cascade from its primaries. Since <A, E> and its path tree are finite, it follows that the assignment of DCOM and dcom values for every node in that path tree can be achieved: DCOM(n) can be computed for every node n in the path tree of <A, E>.

Using the path tree of a CRKH <A, E> we can, by combining the DCOM and deductive distance values for each leaf (pendant) of the path tree, where each leaf is a copy of some goal of <A, E>, assign a complexity value to each derivation path in <A, E>, thereby establishing a partial order of the derivation paths in <A, E> from the least complex to the most complex. This leads to a presentation strategy – see [GVS99]. We make some brief comments on this situation in our third intermission.

20. Third intermission

.

This intermission is very speculative, and is partly for the amusement of frustrated theoretical physicists like one of the authors (HO van Rooyen), but also contains some serious suggestions about the use of CRKS's and CRKH's in the representation of study material – see Parts I and II of [GVS99]. In such a representation, the notion of a CRKH frees the designer from specific statements of relationship in a particular teaching metalanguage, opening the way to "language free" design.

Hamilton's Principle of Least Action appears to be a potential unifying principle for the theories of relativity, electromagnetic theory and quantum mechanics: All we need seems to be the appropriate definition of "action" in each case. In this author's view, it is unfortunate that the principle is formulated in a number continuum that is a wholly human invention and has little to do with the "real world", thus constituting a fundamental a priori flaw in the models used in these fields of theoretical physics. Discrete models would be more suitable, both for (partial) representation and for computation, and also for simulation, but are slow to appear, partly probably due to the overwhelming concentration on the apparent success of real and complex number modelling in Physics in general and a concurrent and appalling neglect of basic scientific method in several fields of the physical "sciences" in recent years.

Where shall we look for potential "discrete versions" of Hamilton's Principle? Let me suggest here that we have indeed uncovered something, in the very unlikely field of education, that looks suspiciously like a discrete version of Hamilton's Principle and is also lightly attached to probability, depending of course on ones enthusiasm for that Principle, which can make one see ghosts where there may not be any!

We have seen that deductive complexity can be used to partially order the set of all derivation paths in a CRKH, and of course in a CRKS. By partially ordering the members of each of these "deductive complexity equivalence classes" by means of the deductive distances of the leaves of our path tree from the primaries, we can define an overall ***complexity*** for each derivation path of the relevant CRKH (or CRKS). Now consider any one of these "complexity equivalence classes", and let it have n members, each of which is a derivation path with the same complexity. (One might suppose that a teacher/learner would start with the class of lowest complexity!). The a priori probability that the teacher/learner will choose to start with any one of these derivation paths is 1 in n, and the moment one is chosen the probability of choice of all the rest in the equivalence class becomes 1 in (n-1). Of course the teacher/learner will often make these choices on the basis of personal familiarity with the "subsidiary" derivations involved in teaching/learning a given derivation path, but to some extent this heuristic, and subjective, influence on the order of choice of derivation paths has been built into the complexity measure of those paths. What is asking to be recognised here is a sort of "least action" principle to be adhered to by "good" teachers/learners: The teacher/learner must follow a derivation path of minimum complexity every time s(he) has a choice, and the probability of choice varies with every choice actually made, indicating a clear influence of the chooser on the whole of the remaining CRKH (CRKS) every time a choice is made. The situation is of course actually more complicated than this simple choice of derivation path indicates, because in choosing a derivation path we are in fact choosing a whole sub-hypernet (sub-net) in which at least one derivation of each vertex between p and g, p a primary and g a goal, on the chosen derivation path p — g must exist. One such sub-hypernet (sub-net) for p — g can easily be constructed from the labelling of p — g on our path tree, so all the information is available for our chosen path p — g. One can see quite plainly how this works in the example that constitutes Part II of [GVS99].

If we define the complexity of a flow to be the sum of the derivation path complexities over all the independent derivation path sections of the paths in the flow, then the same thing can be said about the choice of a particular flow: A flow of lowest complexity can be found by selecting a derivation path of lowest capacity from each of the sets of paths from which one member is chosen, from each set, to make up a flow.

Again we may have several distinct flows with the same flow complexity, so we meet the same situation as with the choice of individual derivation paths, and a similar Principle applies. One may ask, incidentally, why a teacher/learner would wish to teach flows. The answer is that it makes some good sense as the various paths belonging to any given flow each go through a “region” of the relevant CRKH (CRKS) that is unrelated to the “region” through which any other path in that flow goes.

Our principle of least complexity can be worded in a fairly evokative way. The “movement” of a teacher/learner “through” a CRKH (CRKS) *will* always be along a derivation path of minimum complexity in the current CRKH (CRKS), or, more succinctly: The Principle of Least Complexity for “Good” Teachers/Learners. Given a choice of (derivation) paths (or of flows) from one (primary) vertex to another (goal) vertex in (CRKH/CRKS) space, a path that is of least complexity under the current conditions in that space will be followed.

While this intermission is likely to evoke a few chuckles over a cocktail there is, hidden behind it, a serious appeal to give more consideration to the marvellous Principle of Hamilton and to free it, if possible, from its original home in the calculus. As discrete modelling, the natural field of the digital computer, becomes ever more important we may find much wider use of such theorems as that of Menger in its various formats, and perhaps too of general principles such as that of Hamilton’s “Least Action”.

21. An Extended View of Modelling Study Material

Before continuing with the development of CRKH theory, we will use a simple example to show how the use of the representation of study material in CRKS form is linked with, and is extended by, the notion of a CRKH. Our illustration is the partial model of CRKS theory itself, as given in Appendix A of [GVS99], in the form of a CRKS. For convenience, we repeat the statements of Appendix A here. The CRKS is rather trivial, but is an adequate illustration of the point that we wish to make in this section.

The concept-names in the statements are those printed in bold. Here are the statements, made on the basis of part of Part 1 of [GVS99].

1. The **problem** of devising a science of teaching has a potential solution in terms of **vee diagrams**.
2. The **problem** of devising a science of teaching has a potential solution in terms of **concept circle diagrams**.
3. The **problem** of devising a science of teaching has a potential solution in terms of **concept maps**.
4. The **problem** of devising a science of teaching has a potential solution in terms of **semantic networks**.
5. The **problem** of devising a science of teaching has a potential solution in terms of **conceptual graphs**.
6. The **problem** of devising a science of teaching has a potential solution in terms of **CNR-nets**.
7. **Concept maps** deal with **concept-names** and **relationships** among them, as do **CNR-nets**.
8. **Concept-names** are represented by the vertices in a **CNR-net**.
9. **Relationships** are represented by the **tuples** in a **CNR-net**.
10. **Tuples** represent **relationships** in a **CNR-net**.
11. A **CNR-net** has **subnets**.
12. The set of all **subnets** of a **CNR-net**, with meet and join defined on it, forms a **distributive lattice**.
13. A **concept-name**, in a **CNR-net**, represented by a vertex with in-degree zero and out-degree ≥ 1, is called a **primary**.
14. A **concept-name**, in a **CNR-net**, represented by a vertex with out-degree zero and in-degree ≥ 1, is called a **goal**.
15. A **primary** is a vertex with in-degree zero and out-degree ≥ 1 in a **CNR-net**.
16. A **goal** is a vertex with out-degree zero and in-degree ≥ 1 in a **CNR-net**.
17. A **CNR-net** with at least one **primary**, at least one **goal**, and no circuits, and in which each **concept-name** is **related** to at least one other **concept-name**, is called a **formal schema**.
18. A **formal schema** that consists of all the **tuples** that involve a given **concept-name** constitutes, for that **concept-name**, its **context-schema**.
19. A **formal schema** in which every vertex has degree ≥ 1 is said to be **complete**.
20. A **formal schema** may have the property that every one of its vertices is **derivable**.
21. A **complete formal schema** in which every vertex is **derivable** is called a **CRKS**.

22. **Derivability** and **completeness** of a **formal schema** characterizes a **CRKS**.
23. A **primary** in a **CRKS** is trivially **derivable**.
24. Every statement of **relationship** in a **CRKS** is treated as an inference rule: This leads to the notion of **derivability**.
25. A **formal schema** that is **complete** and in which every vertex is **derivable** is called a **CRKS**.
26. **Tuples** in a **CRKS** are preserved by **CRKS isomorphism**.
27. **Isomorphism** of **CRKS's** expresses **structural analogy**.
28. **Structural analogy** is expressed in terms of **isomorphic** (sub-) **CRKS's**.
29. **Isomorphism** is used to express **structural analogy** among (sub-) **CRKS's**.
30. **Derivability** is realized in a **CRKS** by means of **derivation paths**.
31. **Derivation paths** express **derivability** in a **CRKS**.
32. **Derivability** is realized in terms of **derivation paths** in a **CRKS**.
33. A **formal schema** can be searched for relevant **subnets** using **cascades**.
34. A **cascade** from the **primaries** of a **formal schema** can be used to test a **formal schema** for **CRKS** form.
35. In a **formal schema** we can use a **cascade** from the **primaries** to test for **CRKS** form.

These statements do not tell us much about CRKS's, but we can continue to design more statements until we "cover" CRKS theory. This is just a simple illustration after all!

The **Tuples Table** is as follows, with the tuple set for each.

1. <problem, vee diagram>	{problem, vee diagram}
2. <problem, concept circle diagram>	{problem, concept circle diagram}
3. <problem, concept map>	{problem, concept map}
4. <problem, semantic network>	{problem, semantic network}
5. <problem, conceptual graph>	{problem, conceptual graph}
6. <problem, CNR-net>	{problem, CNR-net}
7. <concept map, concept-name, relationship, CNR-net>	{concept map, concept-name, relationship, CNR-net}
8. <concept-name, CNR-net>	{concept-name, CNR-net}
9. <relationship, tuple, CNR-net>	{relationship, tuple, CNR-net}
10. <tuple, relationship, CNR-net>	{tuple, relationship, CNR-net}
11. <CNR-net, subnet>	{CNR-net, subnet}
12. <subnet, CNR-net, distributive lattice>	{subnet, CNR-net, distributive lattice}
13. <concept-name, CNR-net, primary>	{concept-name, CNR-net, primary}
14. <concept-name, CNR-net, goal>	{concept-name, CNR-net, goal}
15. <primary, CNR-net>	{primary, CNR-net}
16. <goal, CNR-net>	{goal, CNR-net}

So far the difference is that the entries in the tuples are in a strict order, but those in the edges are unordered.

17. <CNR-net, primary, goal, concept-name, relationship, concept-name, formal schema>	{CNR-net, primary, goal, concept-name, relationship, formal schema}

18. <formal schema, tuples, concept-name, concept-name, context-schema>	{formal schema, tuples, concept-name, context-schema}
19. <formal schema, complete>	{formal schema, complete}
20. <formal schema, derivable>	{formal schema, derivable}
21. <complete, formal schema, derivable, CRKS>	{complete, formal schema, derivable, CRKS}
22. <derivability, complete, formal schema, CRKS>	{derivability, complete, formal schema, CRKS}
23. <primary, CRKS, derivability>	{primary, CRKS, derivability}
24. <relationship, CRKS, derivability>	{relationship, CRKS, derivability}
25. <formal schema, complete, derivable, CRKS>	{formal schema, complete, derivable, CRKS}
26. <tuple, CRKS, CRKS, isomorphism>	{tuple, CRKS, isomorphism}
27. <isomorphism, CRKS, structural analogy>	{isomorphism, CRKS, structural analogy}
28. <structural analogy, isomorphic, CRKS>	{structural analogy, isomorphic, CRKS}
29. <isomorphism, structural analogy, CRKS>	{isomorphism, structural analogy, CRKS}
30. <derivability, CRKS, derivation path>	{derivability, CRKS, derivation path}
31. <derivation path, derivability, CRKS>	{derivation path, derivability, CRKS}
32. <derivability, derivation path, CRKS>	{derivability, derivation path, CRKS}
33. <formal schema, subnet, cascade>	{formal schema, subnet, cascade}
34. <cascade, primary, formal schema, formal schema, CRKS>	{cascade, primary, formal schema, CRKS}
35. <formal schema, cascade, primary, CRKS>	{formal schema, cascade, primary, CRKS}

In designing a CRKS we need to decide on the primaries, the goals, and the concept-names, and then write out statements and permutations (re-statements) of relationships, constructing the diagram at every step as the developing diagram often indicates what kind of statements need to be made in order to archieve derivability of every vertex. A useful hint is to run a limited access cascade from the primaries at each stage of the design, getting each step of the cascade complete before moving to the following step of the cascade. The diagrams are given in figure 21.1 (CRKS) and figure 21.2 (CRKH). In labelling the diagrams we use some obvious abbreviations of concept-names and the edges of the CRKH are labelled by the index number of the statements.

In the diagram of the CRKS we have entered tuple numbers on the arrows. Thus for example, 3 arises from <problem, concept map> and the complete label is 3; ∅. 17 arises from <CNR-net, primary, goal, concept-name, relationship, concept-name, formal schema> and the complete label has only one member, 17; <primary, goal, concept-name, relationship, concept-name>. In Figure 21.2 the corresponding complete label is 17; {CNR-net, primary, goal, concept-name, relationship, formal schema}. In the diagram of the CRKH for this knowledge about CRKS's, we have entered only the edge index numbers. Thus, for example, the label l({derivable, CRKS}) is made up of E_{22} and E_{32}, so, in full, we have l({derivable, CRKS}) = {{derivability, complete, formal schema, CRKS}, {derivability, derivation path, CRKS}}.

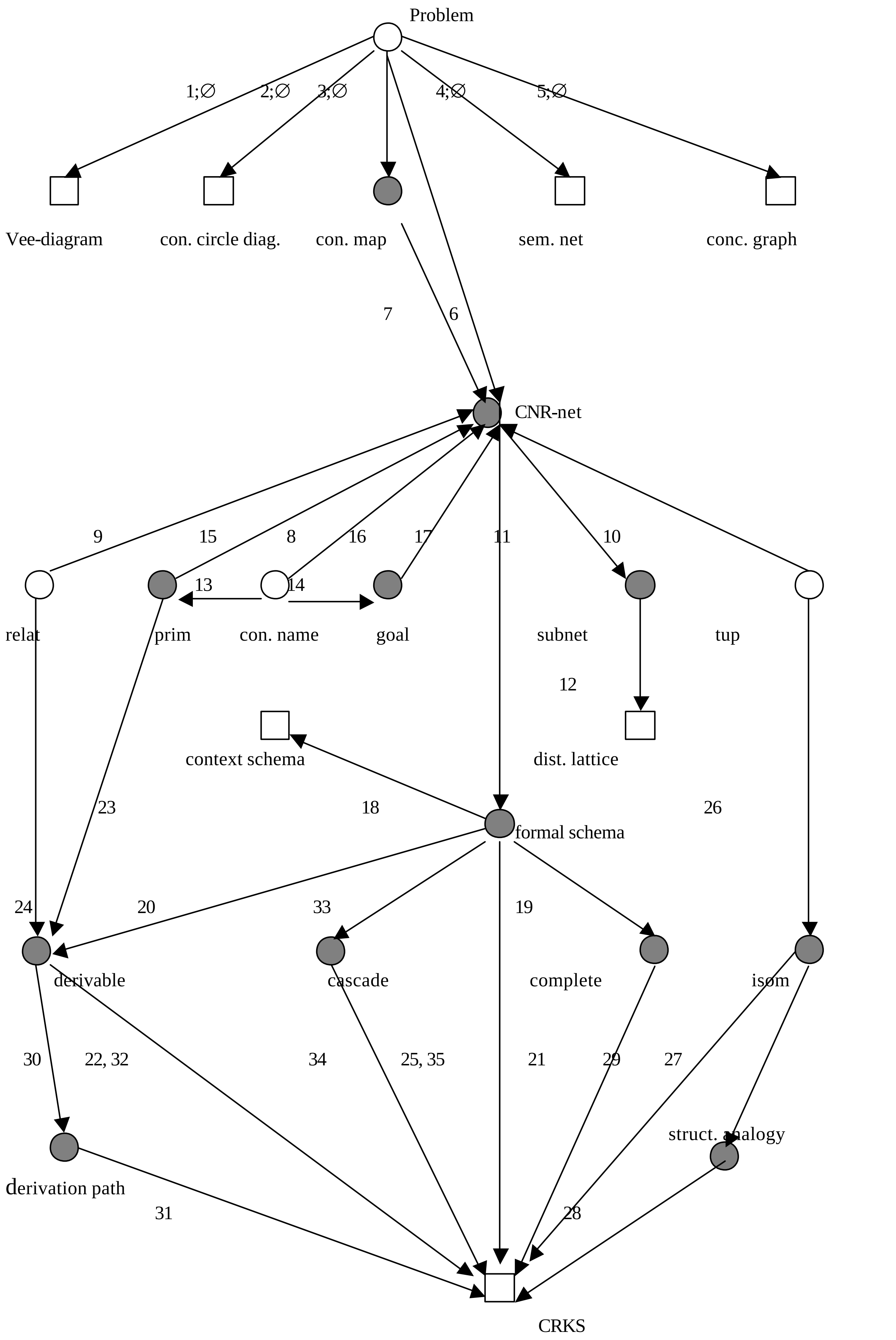


Figure 21.1: CRKS where a circle indicates a primary and a square indicates a goal

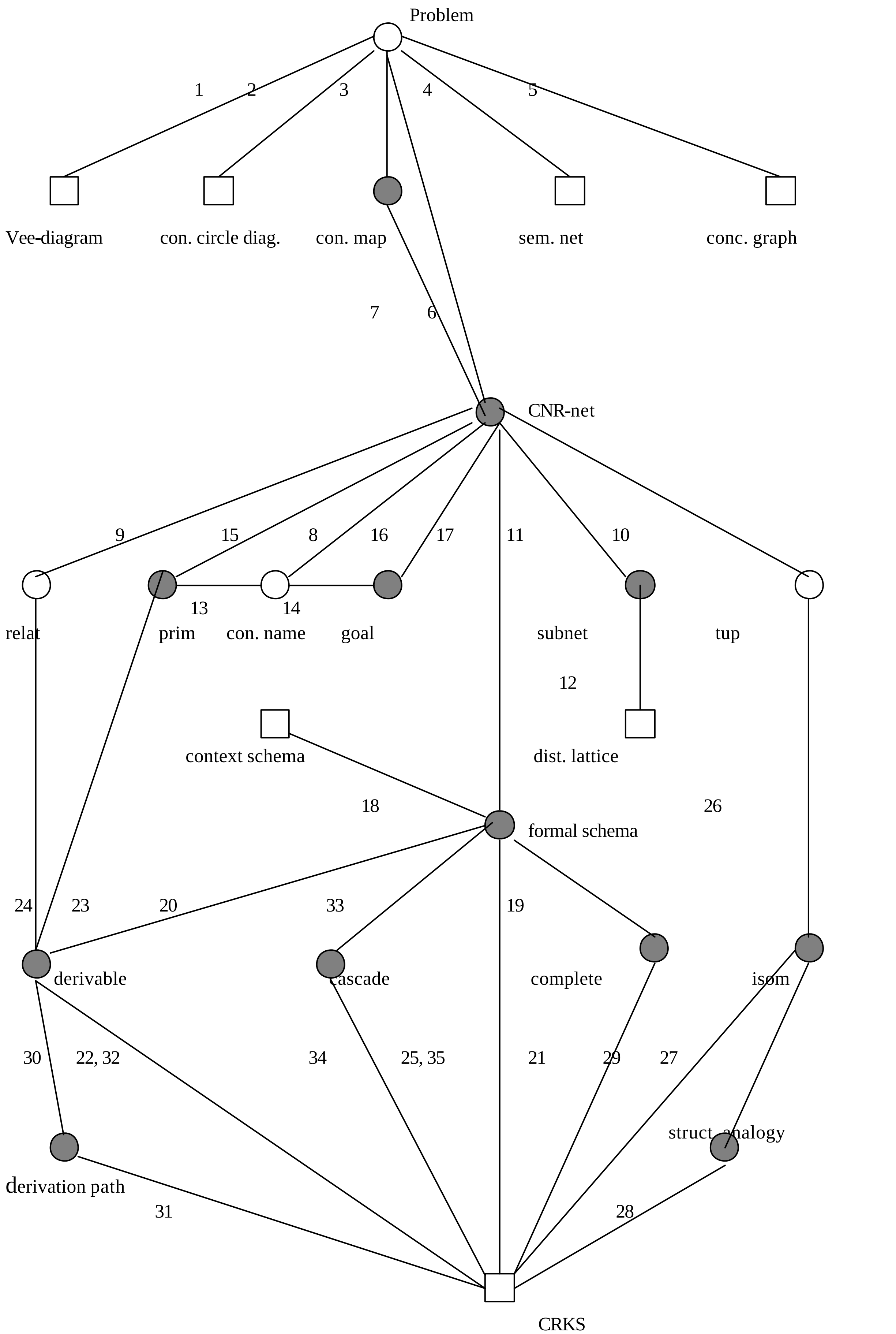


Figure 21.2: CRKH where a circle indicates a primary and a square indicates a goal

Starting with a CRKS $<A_1, T_1>$ with the statements of relationship among its concept-names in a teaching metalanguage L_1, we can easily abstract the unique corresponding CRKH $<A_1, E_1> = M [<A_1, T_1>]$. Now we can translate the concept-names to another teaching metalanguage L_2, bearing in mind that a concept-name can even be a phrase in L_1 and/or L_2, producing a 1-1 correspondence between A_1 and the set of translated names A_2. Next we construct a CRKH $<A_2, E_2>$ that is isomorphic with $<A_1, E_1>$. Then we write statements of relationship in L_2 using, for each edge $E_i \in E_1$, all and only the translations of the members of E_i as the L_2 concept-names in the relevant statement in L_2, where we can use the translated members of E_i in any order, and each can be used any number of times in the statement in L_2. The relationships, as opposed to the statements of relationship, should remain unchanged. This now defines a set T_2 of tuples, and we have $<A_2, T_2> = I [<A_2, E_2>]$ for some interpretation I of $<A_2, E_2>$.

Thus $<A_1, T_1>$ and $<A_2, T_2>$ may be made isomorphic by appropriate choice of interpretation of $<A_2, E_2>$, but such a choice may be impractical. Indeed, we may not want to be restricted to having $<A_2, E_2>$ isomorphic with $<A_1, E_1>$ as this forces one to preserve vertex adjacencies. In that case $<A_2, T_2> = I [<A_2, E_2>]$ must be chosen in such a way that the relationships expressed by the members of T_1 are preserved by the members of T_2 in teaching language L_2. We refer the reader to Definition 2.18.

In summary, the second diagram is that of the unique CRKH which is the abstraction of the CRKS represented by the first diagram, though many distinct CRKS's can of course have this same CRKH as abstraction. The concept-names involved in the CRKS can be translated to, or constructed in, another teaching metalanguage, and from these we could build a CRKH that is isomorphic with the English language (in this case) CRKH represented by the second diagram. The new CRKH can now be interpreted as a CRKS in the "new" language in a number of ways, where we recall that Theorem 17.5 asserts that if that "new" hypernet is a CRKH then each and every interpretation of it is a CRKS. Such a CRKS can now be used to teach/learn the knowledge represented by our English language CRKS in the "new" language. Heuristically, the statements from which the tuples arise in the "new" language should be chosen, from the alternatives for each CRKH edge, in a manner that best suits the teachers/learners in that language. It may be that the vertex adjacencies forced upon the designer are inappropriate in the "new" language.

22. Accommodation and analogy

Definition 22.1: By an ***accommodation*** of a CRKH <A, E> we mean any restructuring of <A, E>, for example adding 1 to the ***weight*** of an edge $E_i \in E$ every time that E_i is used in any way, thereby emphasizing certain edges of <A, E> in the sense that the higher the weight of an edge in the current, accommodated hypernet, the greater the "user familiarity" with that edge. By a ***unit edge accommodation*** we mean adding one edge to <A, E>. By a ***unit vertex accommodation*** we mean adding one vertex to <A, E>. By a ***hypercluster accommodation*** we mean adding a hypercluster for some new edge to <A, E>.♦

Definition 22.2: In the case of unit accommodations and hypercluster accommodations of a CRKH <A, E>, we say that the accommodation is ***assimilated*** by <A, E> iff the restructured hypernet that results is itself a CRKH. ♦

It is clear that a unit edge accommodation of a CRKH <A, E> in which all the members of the new edge are elements of A is the simplest form of accommodation. A unit vertex accommodation of a vertex $v \notin A$ will of course never be assimilated: We need to add in, as well, appropriate associations with members of A, in the form of new edges, to produce a context hyperschema for v that is assimilated by <A, E> if our objective is to construct CRKH's from simple structures. If a unit edge accommodation involves an edge in which there is at least one vertex $v \notin A$ then we have a slightly less complex problem, because here we introduce both v and an edge that has v as a member.

As was indicated in [GVS99], the most "natural" kind of accommodation is (hyper) cluster accommodation, because of the key role of (hyper) clusters in teaching/learning and in finding (CRKH) CRKS isomorphisms in practical situations in which analogy modelling is used. We will return to this point in the later section on isomorphism and structural analogy for CRKH's.

Finally, let us point out that even though a hypercluster is, by definition, a (minimal) CRKH for a given edge, accommodating a hypercluster into a CRKH does not always lead to effective assimilation of that hypercluster. Certainly the join of the CRKH <A, E> and a hypercluster that is disjoint from <A, E> will yield a CRKH, so that hypercluster is assimilated by <A, E>, but this is a trivial situation of no importance: What we need to do is consider only such hyperclusters that are not disjoint from <A, E>, i.e. the meet of <A, E> and the hypercluster in question has at least one vertex, and here there may be real problems that require to be dealt with to achieve assimilation of the hypercluster by <A, E>. If we deal with the case in which the meet is $< \emptyset, \emptyset >$ then the accommodation and assimilation is useless in restructuring <A, E> in practice. What we need for effective assimilation is that we add to <A, E> and the hypercluster in question enough vertices and edges to end up with a restructured hypernet <A', E'> that is a CRKH and is such that the hypercluster introduced belongs to a component of <A', E'>.

Combining unit and hypercluster accommodations can always produce, with enough perseverance, an (effective) assimilation. Some brief comments on accommodations in the case of CRKS's are presented in [GVS99].

23. Isomorphism and Structural Analogy

To see if two given CRKS's, or two given CRKH's, are isomorphic we can use constructional scheme 6.2 of [GVS99], which easily transcribes to the CRKH case. If two CRKH's (CRKS's) are isomorphic then we say that they are ***structurally analogous***. The use of structural analogy in teaching/learning by virtue of the use of "modelling" has been discussed, in the case of CRKS's, in [GVS99], and the discussion applies to CRKH's as well. Further, an example of structural analogy is presented in Chapter 7 of [GVS99], and again that work can be transcribed to the case of CRKH's. Also covered in that chapter of [GVS99], and also smoothly transcribable to CRKH's by simply replacing tuple labels with tuple-set (edge) labels on the arcs, even leaving arc directions unchanged for ease of reading, is the section on theorem proofs.

What, then, is the reason for introducing CRKH's in this connection? Well, the central problem is that of finding, if possible, an isomorphism between two sub-CRKS's: Given <A_1, T_1> and <A_2, T_2>, how can we find and construct an isomorphism between them? In [GVS99] a rather complex constructional scheme to do this, if possible, was presented. We now wish to point out that an easier solution appears from the notions of interpretation and abstraction. Setting up the problem in the field of teaching/learning "new" knowledge by refering to given knowledge, i.e. in the sphere of teaching by the use of a "model" of new knowledge in terms of given knowledge, we visualize the following situation in which we need to construct an isomorphism, i.e. a structural analogy, to compare new, developing knowledge with given knowledge.

We start with existing knowledge in the form of a CRKS K = <A, T> and some "new" observations in the form of a cluster K' = <A', T'> for some tuple of "new" concept-names. Now in seeking a match, in K, for K', we meet the first, and greatest, problem in trying to set up an isomorphism/structural analogy between a sub-CRKS of K and the cluster K': That of relative permutations. How do we recognise a match between a tuple in K and a tuple in K' when we have to take account of all possible permutations of both tuples? Bearing in mind that the whole procedure is a trial-and-error attempt to find the "best" structural analogy – see Chapter 8 in [GVS99] – we side-step this problem while maintaining the basic approach used in [GVS99], as outlined briefly below.

First we abstract K = <A, T> and K' = <A', T'>, producing CRKH <A, E> and the hypercluster CRKH <A', E'> respectively. Now relative permutations are irrelevant. Next we look at the member or members of E', assuming that not all members of E and of E' are unordered pairs, and find a matching of <A', E'> in <A, E> by matching all the sets in E' with a collection of the same number of sets in E that form a hypercluster in <A, E>, if possible. There may be several such matchings, so it is better, but not essential, to start with a number of "new" hyperclusters and try to match them simultaneously. Even then there may be more than one possible initial matching, but continuing with the construction will show which initial matching is "best". (Of course one can also apply heuristics in deciding between several possible matchings, but our formal measure of relative success is the number of vertices and edges in the final matching.)

Next we turn the isomorphism found from <A’, E’> into a hypercluster in <A, E> round, and expand its domain in <A, E> one edge at a time, each edge having as ”large” a meet with the current domain of the growing isomorphism in <A, E> as possible. Each edge projected by the tentative expansion of the domain of our CRKH isomorphism is tested as follows. We define, at each stage of the ”prediction” from <A, E>, an interpretation of the ”predicted” CRKH, based on expanding the inverse of the abstraction of <A’, E’>, and producing for each predicted edge a tuple from that edge. What tuple? Well, combining the abstraction of <A, T> with the potential CRKH isomorphism and the developing interpretation we can identify the potential matching tuple in <A, T>, so we can construct a matching tuple in the growing new knowledge CRKS that contains <A’, E’>.

Now try to provide semantics for that predicted new tuple by trying to write an appropriate and consistent statement of relationship for that tuple, identifying the relevant ”new” concept-names in that tuple. If this effort is ”acceptable”, and judging that may require some empirical work suggested by the predicted tuple, then we accept the ”prediction”; if not then we move on to another ”prediction”. Eventually we will have found no isomorphism, or several from which to choose, and can use the matching sub-hypernet of <A, E> as a ”model” of the ”new” knowledge for use in presenting the ”new” knowledge. There is just one further stipulation: The matching relation nets must be CRKS’s, and thus the matching hypernets must be CRKH’s, in the case of teaching/learning applications, but in other applications we can broaden the approach to isomorphic matching of general hypernets. To write a constructional scheme for the procedure briefly outlined above is easy.

Finally, the section on the use of abstraction isomorphism and algorithmic isomorphism in the field of problem solving - section 8.5 in [GVS99] - is easily transcribable to CRKH representations of top-down algorithms. In fact, as pointed out in section 1 of this report, the entire treatment of problem solving in [GVS99] is best done in terms of CRKH’s because in [GVS99] we forced an arbitrary order onto the members of the edges. Either top-down direction, with a singleton vertex basis, or bottom-up direction, with a non-empty, non-singleton vertex basis, can be ”read into” the hypernet. If read top-to-bottom we have a (usually connected) hypernet; if read bottom-to-top we have derivation path ordering in a (usually connected) CRKH. In the case of connectedness, which is clearly desireable, a fairly generous slice of the theory of hypernets presented in this report is applicable in the analysis of the structure of the kind of hypernets refered to in section 1, and considerable simple computer support for such analysis can easily be made available.

As pointed out in [GVS99], the isomorphism finding procedure can also be used in other education oriented applications for example, such as in finding and analysing ”common ground” for the current study material among the CRKS’s/CRKH’s drawn up by the members of a class of learners.

24. Models of Reasoning

CRKS models of reasoning were introduced in Chapter 9 of [GVS99], and all that is said there can be transcribed to CRKH models. Models of ***intuitive*** and ***deductive*** reasoning are based upon sequences of fast access and limited access cascades respectively. ***Inductive*** reasoning is based on finding what is common among a number of CRKS's by means of abstraction isomorphisms, and then projecting this structure into (partially) similar new CRKS's by means of algorithmic isomorphism, thereby describing common inductive reasoning formally. If only two CRKS's are involved we describe one as a structural analogue of the other. We are of course assuming that all these CRKS's can have disjoint vertex sets.

Deductive reasoning may be described as ”vertical reasoning” and is geared to developing the consequences of a set of primary concept-names or, in general, certain ”basic facts”. This might also be described as ”male reasoning”, and is predominant in basic education in many fields. In contrast, inductive reasoning may be described as ”lateral reasoning” with some justification, and can also be described as analogical reasoning on the formal basis of CRKS isomorphism. We may also assert that this ”analogical association” can be described as ”female reasoning”. Though we do not of course claim that all males reason vertically and all females laterally, since many people are adept at both methods of reasoning, there seems to be cause to claim that many female learners have more difficulty than males in certain fields of education as the result of the ”male orientation” of organization and presentation of study material. We believe that much more emphasis should be placed on analogical reasoning in teaching and research if we want to achieve a balance between establishing new concepts and the development of their consequences.

In [GVS99] we introduced the notion of cluster sets, and from this the notion of cluster associations. In the CRKH approach to reasoning, this is the precise equivalent of plotting a graph in which each vertex represents the cluster set of a hypercluster, i.e. the union of the edges from which the relevant hypercluster is defined, and two vertices are joined iff the two relevant cluster sets have a non-empty intersection. Notice that we are implying that this edge is included in the vertex set of the (hyper) cluster for that edge. If necessary, permutations of the defining tuple for the (hyper) cluster can be used to construct the (hyper) cluster. Labelling each arc in this graph with the relevant intersection set produces a graph of the cluster associations involved, and following walks in this graph is our model of ***associative*** reasoning.

At the other extreme from associative reasoning, among our five CRKS models of reasoning, is constructive reasoning. This is dependent upon the associations described above. In the other three models we assume that already constructed CRKS's (or CRKH's) exist. In the association model only individual observations, each represented by a (hyper) cluster, exist. The question then is how to order at least some of those (hyper) clusters, using some or all of the associations in our association graph, into a body of knowledge in the form of a CRKS on the basis of (part of) the data displayed in that graph. How do we effectively combine clusters? The process of joining (hyper) clusters together to produce a CRKS (or CRKH) is termed ***constructive*** reasoning. Some mainly heuristic guidelines for this task are set out in Chapter 9 of [GVS99].

In the following section of this report we set out a brief example of models of reasoning. We do this in terms of CRKH's rather than CRKS's because of the flexibility of interpretation into CRKS format. We must however bear in mind that we have always to start all but the association model with specific statements of relationship, thus giving rise to CRKS's from which we abstract to CRKH's for a range of specific interpretations, one of which is that CRKS of origin.

25. An Illustration of Models of Reasoning

In this simple example, in terms of CRKH models, we start by assuming that the properties of addition of integers are discovered by induction from a number of (good and bad) examples, such as the notion of a ”number line” for instance, by the use of (partial) abstraction isomorphisms. Notice that we could opt for the ”common ground” of the ranges of these abstraction isomorphisms, or for the ”best” one.

We take ”integer” to be the only primary concept-name, and we assume that the properties of the relationship of equality are known. Equality is represented by the symbol =, and addition of integers by +. Zero is represented by the symbol 0. All concept-names about which we wish to say something are marked in the statements of relationship given. In order to demonstrate analogical reasoning, in a very small way, we distinguish between the word ”zero” and the symbol ”0” in the sense that we treat ”0” as a concept-name in the statements of relationship, but ”zero” as a non-concept-name word. This trick enables us to find a non-trivial isomorphism between two sub-CRKH's of the CRKH that we construct from our statements of relationship.

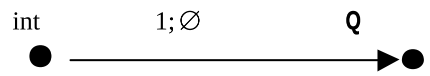
The statements that arise from our ”observed” clusters, and the diagram of each cluster, and, implicitly, the hypercluster abstracted from it, follow. We would show directions, imposed by derivation paths, in the CRKH's, these being those shown in the clusters. We attempt to build a cluster for each tuple defined by using only previously met tuples/statements with the defining tuple of that cluster. For each cluster we define a complexity measure as follows.

Definition 25.1: Given any cluster K, the ***cluster complexity*** of K is given by CCOM(K) = $S\ n_i$ where the sum is taken over all the n_i-tuples of K. Given a hypercluster M [K], the ***hypercluster complexity*** HCOM(M [K]) = $S\ |E_i|$ where the sum is taken over all the edges E_i of M [K]. ♦

It is clear that HCOM([K]) ≤ CCOM(K).

For each of the statements below, we give a cluster K which can easily be converted to the abstracted hypercluster M [K], together with the value of CCOM(K) and the value of HCOM(M [K]). These two values give us one kind of estimate of the relative difficulty of learning the cluster, and hypercluster, respectively.

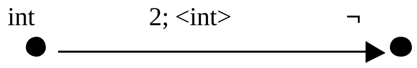
1. Addition of **integers** is represented by the symbol **Q.**
   A cluster for 1 is



CCOM = 2 and HCOM = 2.

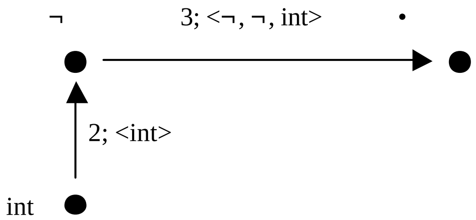
2. For every **integer** x there is a unique negation that is also an **integer** and is represented by the symbol ¬ x.

A cluster for 2 is



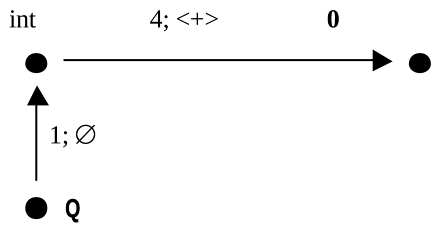
CCOM = 3 and HCOM = 2.

3. ¬ (¬ x), the negative of ¬ x, for every **integer** x, is • x.
   A cluster for 3 is



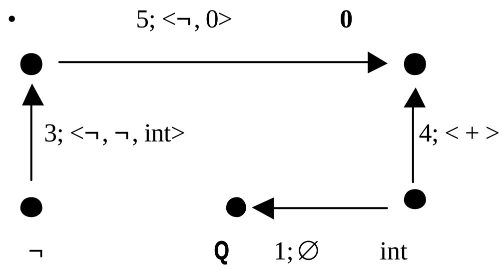
CCOM = 3 + 5 = 8 and HCOM = 2 + 3 = 5.

4. There is a special unique **integer**, for **Q,** called zero and represented by the symbol **0**.
   A cluster for 4 is



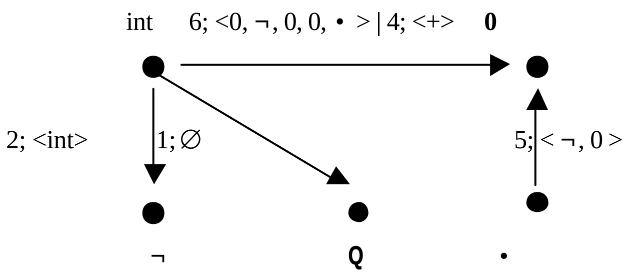
CCOM = 2 + 3 = 5 and HCOM = 2 + 3 = 5

5. • holds between ¬ **0** and **0**.
   A cluster for 5 is



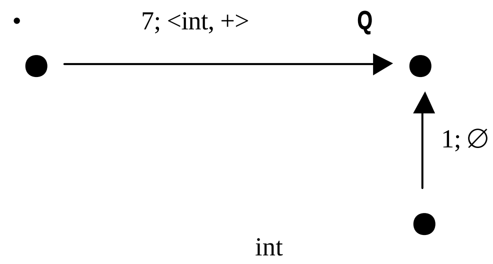
CCOM = 5 + 4 + 3 + 2 = 14 and HCOM = 3 + 3 + 3 + 2 = 11

6. The only **integer** that is its own negative is **0**, i.e. ¬ **0** • **0**.
   A cluster for 6 is



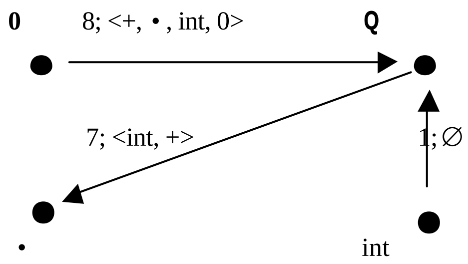
CCOM = 3 + 6 + 3 + 4 + 2 = 18 and HCOM = 2 + 4 + 3 + 3 + 2 = 14
(note: 4; <+> is necessary so as to reach 0 for use in 6).

7. • holds, for any **integers** x and y, between x **Q** y and y **Q** x.
A cluster for 7 is



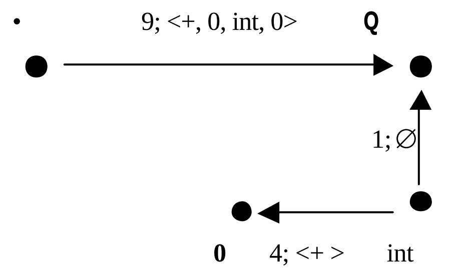
CCOM = 4 + 2 = 6 and HCOM = 3 + 2 = 5.

8. **0 Q** x • x for every **integer** x with **0** under the operation **Q.**
A cluster for 8 is



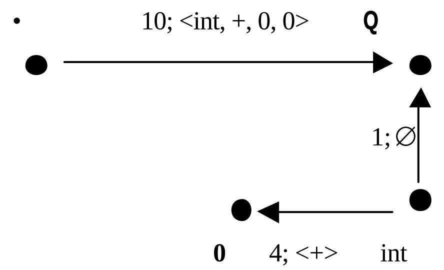
CCOM = 6 + 4 + 2 = 12 and HCOM = 4 + 3 + 2 = 9.

9. x • x **Q 0** for every **integer** x with **0** under the operation **Q.**
A cluster for 9 is



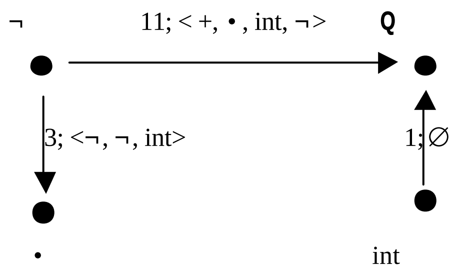
CCOM = 6 + 2 + 3 = 11 and HCOM = 4 + 2 + 3 = 9.

10. From statements 8 and 9 we have that • holds, for every **integer** x, between x **Q 0** and **0 Q** x, which conforms with statement 7.
A cluster for 10 is



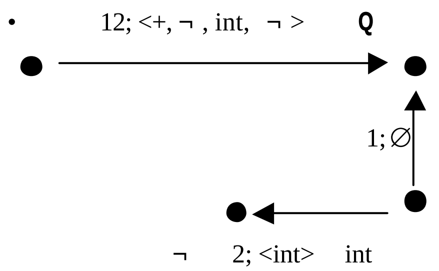
CCOM = 6 + 2 + 3 = 11 and HCOM = 4 + 2 + 3 = 9.

11. ¬ x **Q** x • zero for every **integer** x with ¬ x under the operation **Q.**
A cluster for 11 is



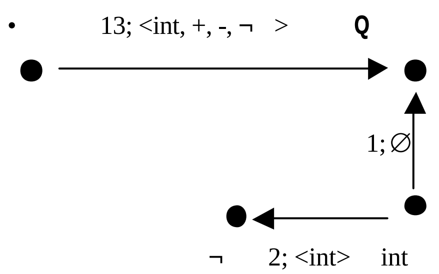
CCOM = 5 + 6 + 2 = 13 and HCOM 3 + 5 + 2 = 10

12. Zero • x **Q (**¬ x) for every **integer** x with ¬ x under the operation **Q.**
A cluster for 12 is



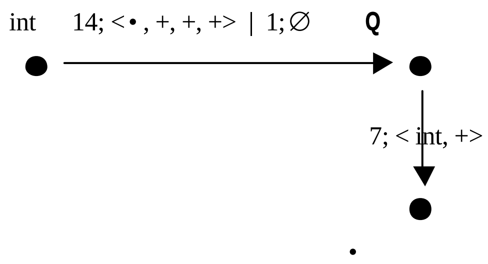
CCOM = 6 + 2 + 3 = 11 and HCOM = 4 + 2 + 2 = 8.

13. From statements 11 and 12 we have that • holds, for every **integer** x, between x **Q (**¬ x) and ¬ x **Q** x. This conforms with statement 7.
A cluster for 13 is



CCOM = 6 + 2 + 3 = 11 and HCOM = 4 + 2 + 2 = 8.

14. For any **integers** x, y and z, • holds between x **Q** (y **Q** z) and (x **Q** y) **Q** z.
A cluster for 14 is



CCOM = 6 + 2 + 4 = 12 and HCOM = 3 + 2 + 3 = 8.

Notice that we must reach **+** by means of 1; ∅ before we can use 14. It is easy to verify that each of our clusters is indeed a minimal CRKS for the tuple in question.

Even this simple example is rich in associations, so the associations graph will be only partially presented: In figure 25.1 we show only those cluster associations that involve ”0”. Each vertex of the graph is labelled with the tuple number of its cluster set, and its cluster set .

Next, we construct a CRKS / CRKH from the given clusters. Because we have simplified the construction by using only previously defined tuples in the cluster for a particular tuple, we can simply start with cluster 1 and then join it with cluster 2, 3, …, 14, in that order, with no problem. The process will not always be so straightforward! Notice that only selected associations are used in constructing the CRKS/CRKH. Some choices of association are as follows. Tuple 4 is associated, via ”0”, only with tuples 5, 6 and 8. Tuple 10 is associated with tuple 5 via ”=”, and tuples 9 and 10 are associated with tuple 8 via ”+”, where our choices are the concept-names at which we make these tuples adjacent and are among a host of such choices which can be made. The CRKS is shown in figure 25.2.

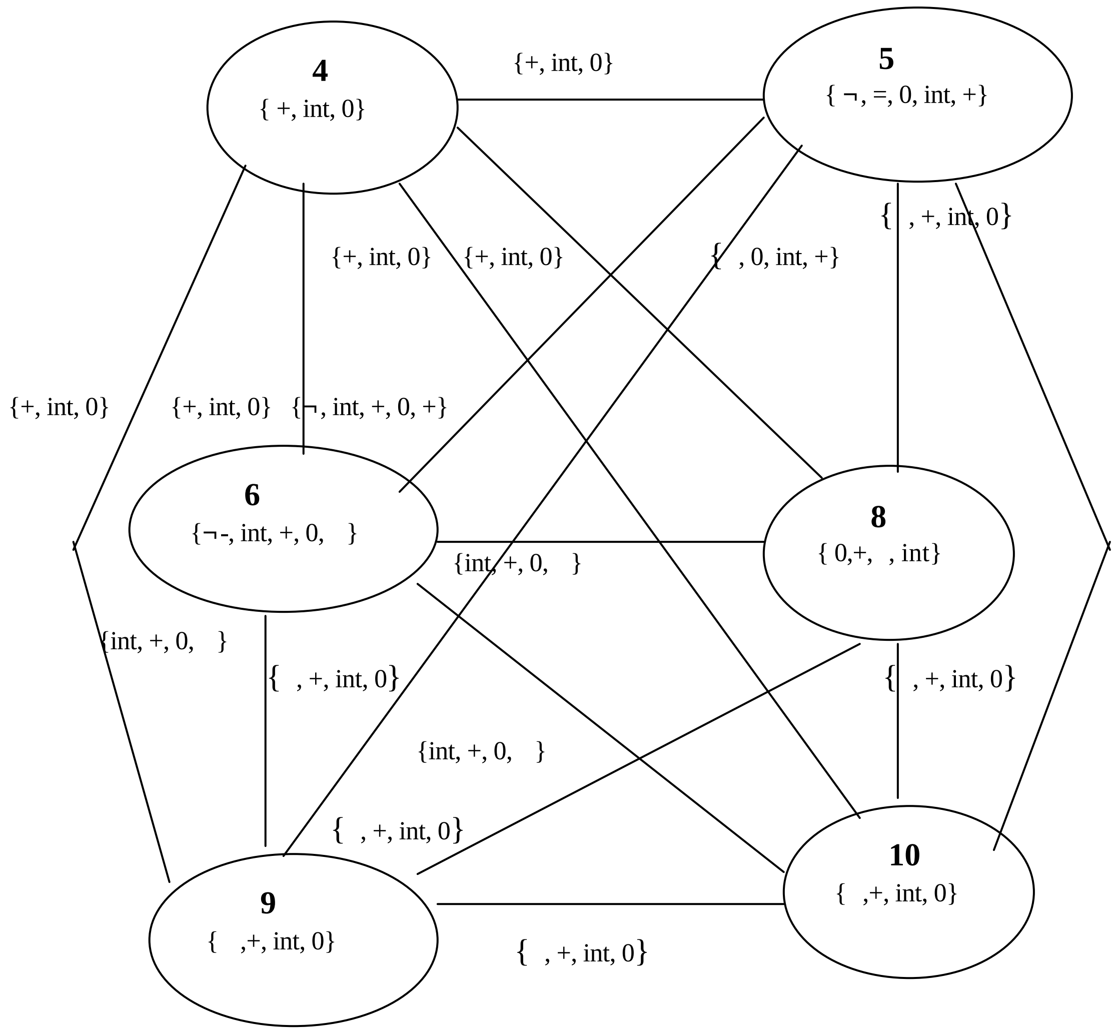


Figure 25.1: Cluster associations involving "0"

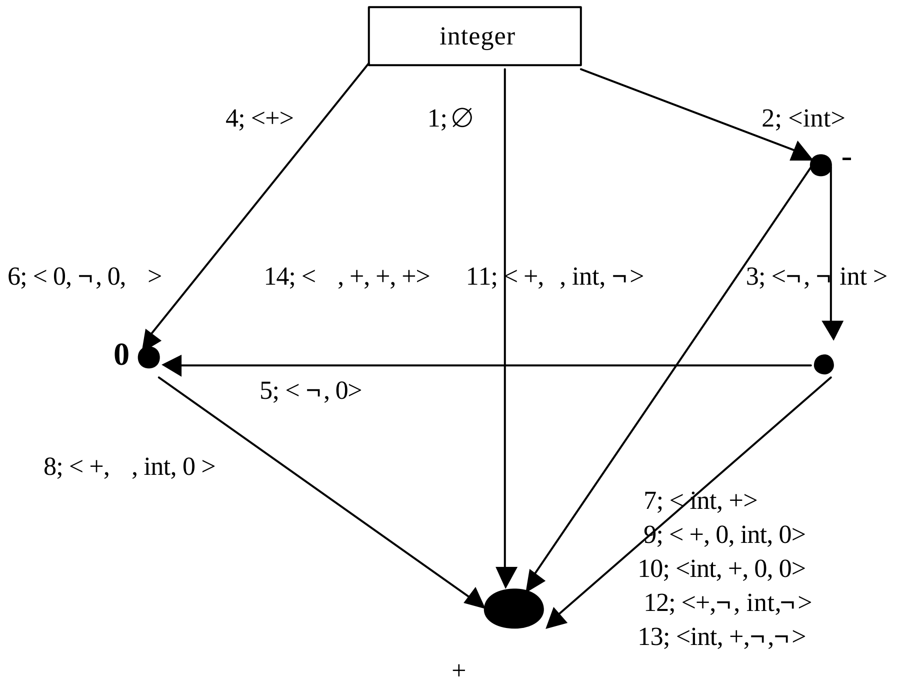
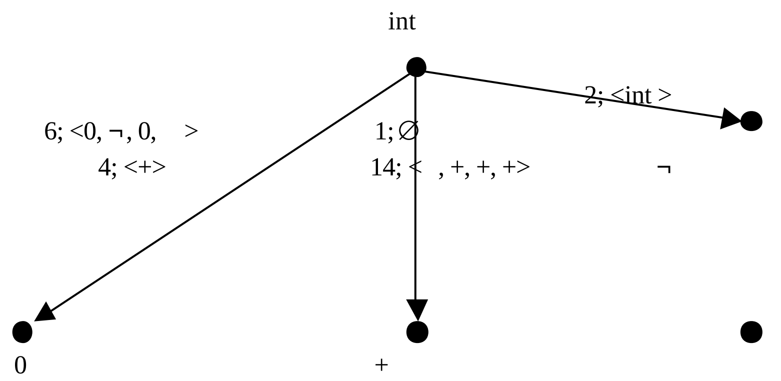


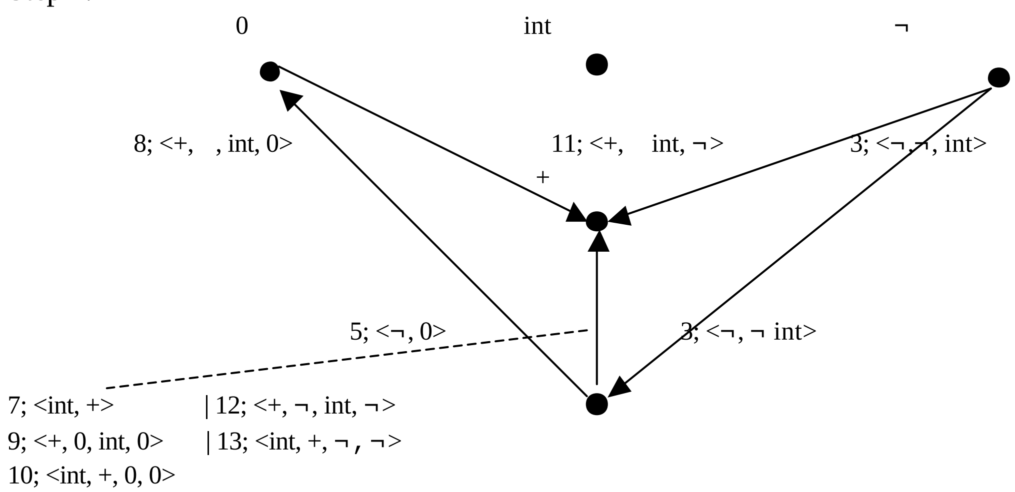
Figure 25.2: CRKS/CRKH

To illustrate our model of intuitive reasoning in this CRKS - figure 25.1 - we run a fast access cascade from B^1_0 = {int}, the only primary. At each step we show only what is newly found in that step.

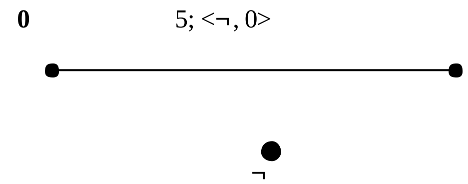
Step 1:



Step 2:

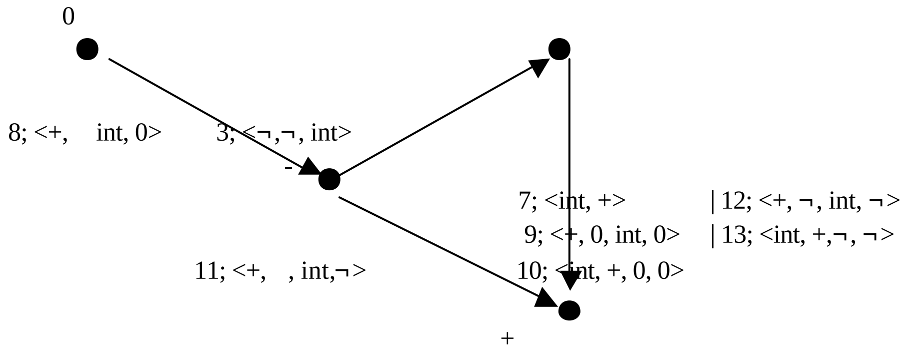


After two steps the whole CRKS has been accessed. Suppose that after step 1 we decide to explore further only the concept-name ” ”. We start a new cascade with B^2_0 = { }. T^2_0 = ∅, and for T^2_1 we have a choice of tuples that start with ” ”, i.e. tuples 5, 7, 9, 10, 12 and 13. If we choose only 5, then this step 2 yields



a formal schema. For the next cascade, let’s choose B^3_0 = {0, }, and T^3_1 = {5} again. We get, in this step 3, the newly found data

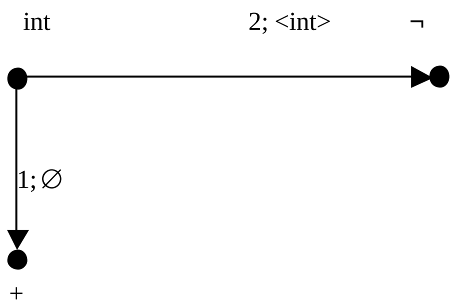
Step 3:



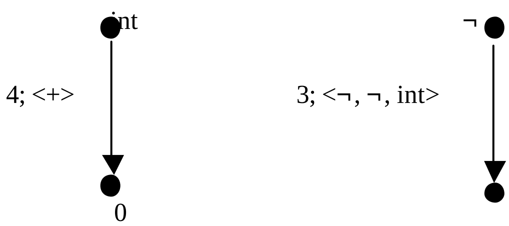
Joining these formal schemas, leaving out the previous step 2, we see that this ”controlled” chain of fast access cascades has generated the given CRKS. The power of this view of intuitive reasoning by means of a sequence of ”directed” fast access cascades will only become apparent when the given CRKS is very large.

To illustrate our model of deductive reasoning in this CRKS we run a limited access cascade from its primary, i.e. $B^1{}_0$ = {int}, in steps, showing what is newly derived in each step.

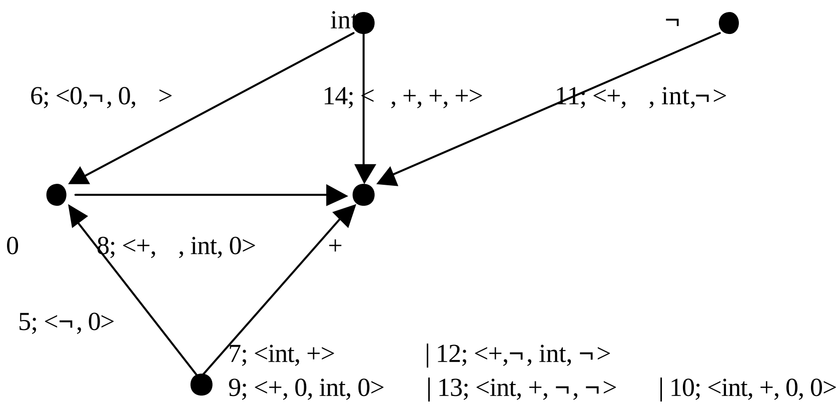
Step 1:



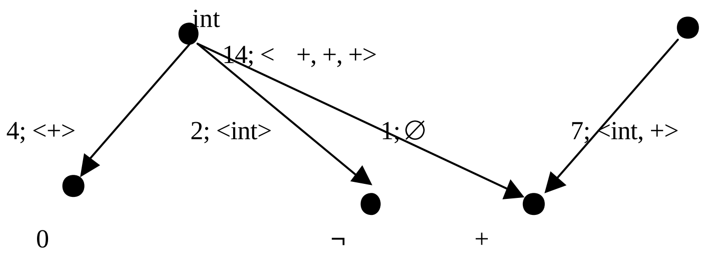
Step 2:



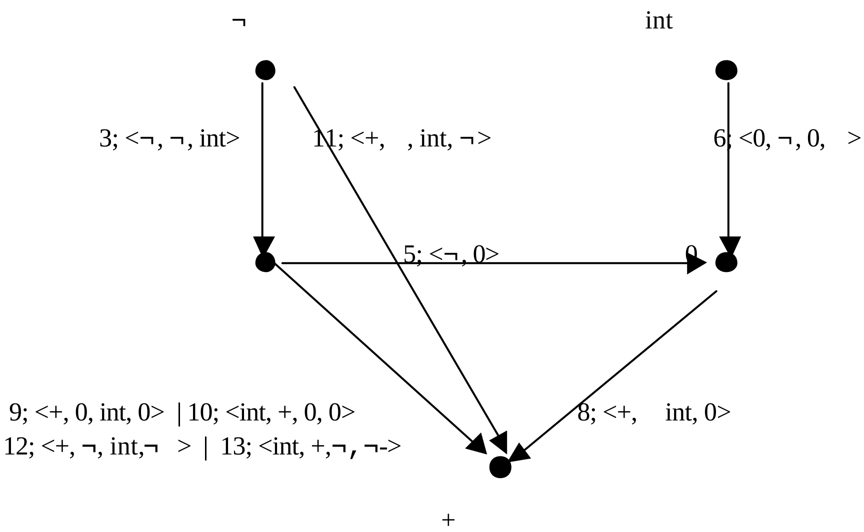
Step 3:



The join of these three formal schemas is precisely our given CRKS. Suppose that after step 2 we decide to continue with a new limited access cascade from $B^2{}_0$ = {int, +, }. In the first step of this cascade we get

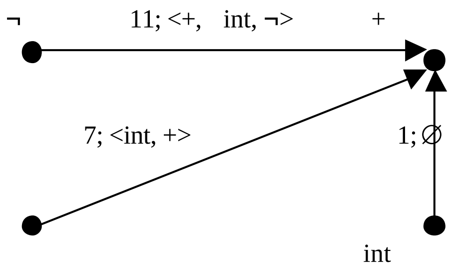


The next step, 2', of this second cascade yields



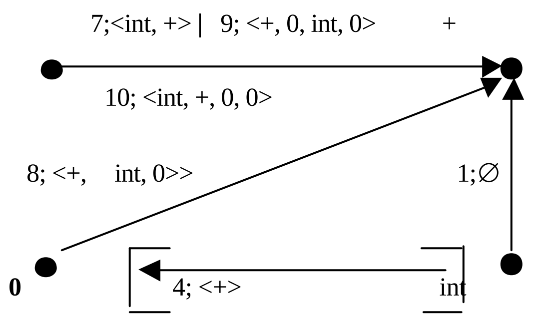
Joining < $B^1{}_1$, $T^1{}_1$>, < $B^2{}_1$, $T^2{}_1$> and step 2' above yields the entire CRKH.

Finally, we point out that it is easy to show that clusters 8 and 11 can be adjusted to be isomorphic. We change to an alternative cluster for 11, for the tuple <¬ , +, , int, ¬, +>, as shown below.

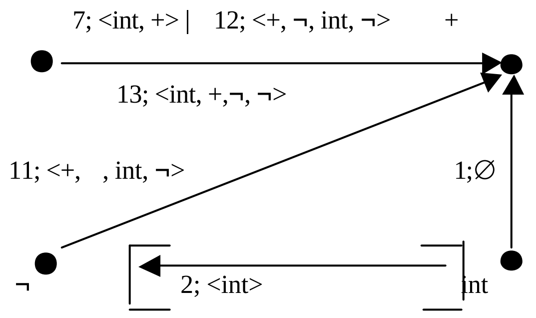


We have deleted 3 and added 7. This does not affect the construction of the CRKS from the clusters. This alternative cluster and that for statement 8 are isomorphic, where ”-” and ”0” are matched, so, for example we can use this structural analogy between the two clusters to teach/learn cluster 11 by referring to cluster 8, previously learned, as a model of cluster 11. Further, it is easy to extend this isomorphism by joining cluster 9, and then cluster 10, to cluster 8, deleting tuple 4, and isomorphically mapping this domain onto the join of cluster 12 and 13, without tuple 2, with our revised cluster for tuple 11.

Joining clusters 8, 9 and 10 yields the CRKS



Joining clusters 12 and 13 to our alternative cluster for 11 yields the CRKS



Ignoring 2 and 4, it is easy to find the isomorphism between these two CRKS's – we can go via the equivalent CRKH's. Expansion of the domain of the mapping one tuple at a time, starting with the isomorphism between clusters 8 and 11 (revised), will break down when we try to map 4; <+>. In most cases isomorphic (sub-) CRKS's/CRKH's will share no vertices.

We refer the reader to Chapters 9 and 10 of [GVS99N] for comments on structural analogy and the uses of CRKS's/CRKH's in education.

Closing comment: It is clear that the digraphs constitute a sub-class of the class of relation nets, and it appears that relation nets have, potentially, a wider domain of practical applications when used as models in such applications. It is also apparent that the graphs form a sub-class of the class of hypernets, as do the hypergraphs. Thus, in general, hypernets should have a wider domain of practical applications, when used as models in such applications, than either of these two sub-classes.

Literature

[Ber73] Berge, C.: Graphs and hypergraphs. North Holland Publishing Company, Amsterdam, 1973

[Ber89] Berge, C.: Graphs. North Holland Publishing Company, 2nd ed., Amsterdam, 1989

[GVS99] Geldenhuys, A.E., Van Rooyen H.O., and Stetter, F.: Knowledge representation and relation nets. Kluwer Academic Publishers, Boston, 1999

[Har69] Harary, F.: Graph theory. Addison-Wesley, 2nd ed., 1969

[SVR93] Stetter, F. and Van Rooyen, H.O.: Program measures based on a graph-like model. J. of Inf. Proc. and Cyb. 29, 55-76, 1993

[VR76] Van Rooyen, H.O.: Binary networks in graph theory. Ph. D. Thesis, Unisa, 1976

[Wei83] Weiermanns, D.J.: Development of a nebula database for a student advice system. MSc. dissertation, Unisa, 1983

Index