Modelling Knowledge Systems using Relation Nets and Hypernets

by

Hendrik O. van Rooyen - University of South Africa
Aletta E. Geldenhuys - University of Botswana
Franz Stetter - Universität Mannheim

Email

"HO van Rooyen" <hopete@intekom.co.za>
"A Geldenhuys" <aegelden@telkomsa.net>
"F Stetter" <fstetter@uni-mannheim.de>

Keywords

Accommodation: formal approach
Assimilation: of study material
Clusters: formal definition in teaching
Concept-Relationship Knowledge Structure (CRKS)
Curricula: design of, analysis of
Digraphs/Graphs: generalized
Discrete structures: with n-tuples for $n \geq 2$, representation/modelling of, analysis of
Hypernets: Generalization of relation nets
Isomorphism: in reasoning by analogy, in teaching via modelling
Knowledge Hypernet (KH)
Knowledge: representation of, modelling of, management of
Menger's theorem: a generalized version, uses in teaching
Problems: representation and solution of
Reasoning: a model for modes of: associative, deductive, inductive, intuitive, constructive
Relation nets
Spiralling: formal definition in teaching
Study material: representation of, design of, analysis of, presentation of
Teaching/Learning: theory of
Theorems: representation of proofs

Preface

In the book Knowledge Representation and Relation Nets [GVS99] we introduced a structural model called a Relation Net and, in Part 1 of [GVS99], a special relation net called a Concept – Relationship Knowledge Structure (CRKS). In this work we broaden the notion of a relation net to produce a new but associated structural model, a Hypernet. We show that the general theory of hypernets has applications in the acquisition/learning, representation, retrieval, accommodation and assimilation, management and communication/teaching of knowledge, and also in problem representation and solution and in modelling the various modes of reasoning. This report is a revised and extended version of [VSG01]. As illustration of the theory we refer, in Appendix 2, to [GVS99] for examples of how CRKS's can be used to represent, assist in the design of, analyse and present study material.

In the first chapter we present some essential background about Relation Nets and Concept – Relationship Knowledge Structures (CRKS's) – see also [GVS99]. We then introduce the notion of a Hypernet. Chapter 2 deals with some of the characteristics of hypernets, following the general lines of development in [HNC65]. In chapter 3 we meet knowledge hypernets (KH's) and their potential applications, and Chapter 4 introduces a NET, called EDUNET, for use in education.

The formal theory is explained in terms of uncomplicated tables, informative diagrams, and construction schemes that can lead to formal algorithms, and hence to simple computer implementation in terms of a suite of standard programs that covers all applications. Appendix 1 lists the constructional schemes.

The entire approach is through structural modelling – see also [HNC65], [LEN80] and section 1.4 of [GVS99] – and one can trace a line of development of such modelling from concept maps through digraphs and relation nets to hypernets. This work is of interest for researchers in the field of knowledge representation and related applications.

Acknowledgements

The authors would like to thank:

- Our families for their patience and encouragement.
- Universität Mannheim for support and financial assistance.
- Petro van Rooyen and Barbara Stetter for typing drafts.
- Professor Hugh Helm for many years of support and for valuable assistance in the field of science education.
- Jackie Brear for encouragement and support.

Contents

1. Background: Relation Nets, Hypergraphs, and Hypernets

By a NET we will mean a collection of stored items of various sorts such that

(1) items can be linked using relational links,

(2) items and/or links can be added to the NET,

(3) items and/or links can be deleted from the NET,

(4) items and links between them define a sub-NET, and

(5) items, and possibly links between them, can be accessed from the NET by following links from the members of any given initial collection of items, possibly with links between some of those initial items, i.e. from an initial sub-NET of the NET.

We make the following distinction, which will become clearer as the presentation progresses. At the lowest level the items are called ***data*** items. Items that consist of a structured/ordered collection of data items are called ***information*** items. Items that consist of a structured/ordered collection of information items are called ***knowledge*** items.

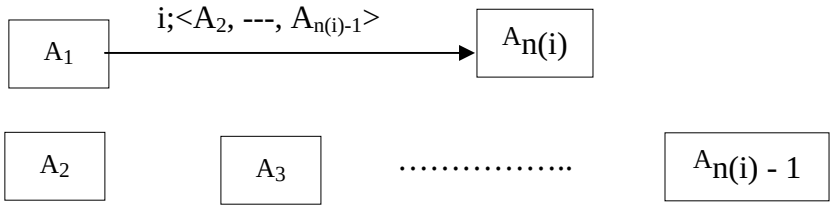
1.1 Relation Nets

In [GVS99] we introduced the notion of a relation net and showed how a special kind of relation net, called a Concept-Relationship Knowledge Structure (CRKS), can be used to order study material in a natural manner that makes that study material more easily teachable and learnable. That work approached the ideal of a "science of teaching" first suggested by Hestenes as long ago as 1979 [Hes79]. We present, in this chapter, a summary of that work as a preliminary to the introduction of hypernets. In later sections we will show how a hypernet can be used as the basis of an uncomplicated model of a NET in education. We begin with a description of a relation net. Certain definitions and theorems from [GVS99] are modified in the process.

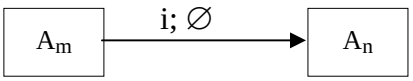
Let A be any finite set, and let T be any finite set $T = \{T_1, T_2, \ldots, T_m\}$ of tuples of members of A where each tuple T_n of T has at least two distinct members of A as entries in it. The structure composed of A and T is written <A, T> and is called a ***relation net***. Thus we will regard a relation net simply as a set A and a set of tuples over A.

A ***diagram*** of a relation net consists of a point, or ***vertex***, to represent each member of A, together with an arrow from the first entry in each tuple to the last entry of that tuple, and a label, on that arrow, which consists of the rest of the entries in that tuple in the order in which they occur in that tuple. Every tuple that starts with $A_i \in A$ and ends with $A_j \in A$ is represented by the appropriate label on the (one only) arrow from A_i to A_j - written $<A_i,A_j>$. Notice that a given member of A may occur any number of times in any of the tuples of T, and that the order of occurrence of members of A in every tuple of T is fixed.

For each $T_i = <A_1, A_2, \ldots, A_{n(i)-1}, A_{n(i)}> \in T$ of a relation net <A, T> we thus have, in the diagram of <A,T>,



If the relevant tuple is a pair $<A_m, A_n>$ then we have



where ∅ represents the empty set. An alternative label, in this case, is i; < >.

A ***Concept-Name-Relationship-Net (CNR-net)*** is a relation net in which the points/vertices/items represent concept-names and the tuples arise from statements of relationship among those concept-names.

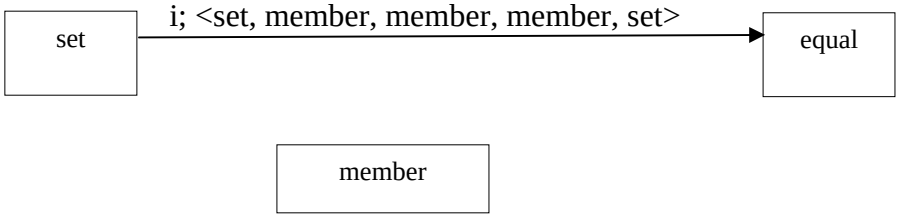
In the case of a CNR-net each tuple $T_i \in T$ arises from, and characterizes, a unique statement of relationship among the concept-names entered in that tuple. Thus, for example, if statement i is:

"If the **set** denoted by X and the **set** denoted by Y have precisely the same **members**, i.e. a is a **member** of X if, and only if, (iff) a is a **member** of Y, then we say that the symbols X and Y denote the same **set**, which we express by saying that X and Y are **equal**",

where the words in bold are those concept-names with which we are concerned in this statement, then we have the following: For T_i we have

i: <set ,set, member, member, member, set, equal>,

and, in the diagram,



Thus, in a CNR-net, the semantics consists of the list of natural language statements

i: statement

while the syntax consists of a listing of the n(i)-tuple of concept-names that arises from the statement of relationship for each i, in the form

i: $<A_1, \ldots, A_{n\,(i)}>$

where the A_k, k = 1, …, n(i), are not necessarily distinct members of A. Notice that non-concept-name words and phrases in statements of relationship for a CNR-net are assumed to be familiar. Note further that with sufficient linguistic dexterity one can re-word any statement in such a way as to achieve any permutation of the n-tuple of concept-names that arose from the original statement, without changing the relationship expressed.

How well is an $A_m \in A$ related to other members of A in a given relation net <A, T>? First we look at the "local" situation of A_m.

The ***in-degree*** of $A_m \in A$ in a relation net <A, T>, written $id(A_m)$, is the total number of labels on all the arrows to A_m. The ***out-degree*** of A_m, written $od(A_m)$, is the total number of labels on all the arrows from A_m. The ***degree*** of A_m, written $d(A_m)$, is given by $d(A_m) = id(A_m) + od(A_m)$.

Constructional Scheme 1.1.1: To determine $id(A_m)$, $od(A_m)$ and $d(A_m)$ for $A_m \in A$ in a relation net <A, T>.

(1) Count the number of tuples, in the tuples list/table T of <A, T>, that end with A_m. This $id(A_m)$.
(2) Count the number of tuples, in the tuples list/table of <A, T>, that begin with A_m. This is $od(A_m)$.
(3) $d(A_m) = id(A_m) + od\,(A_m)$. ♦

Comment: Implementation is obvious.

For $A_m \in A$ of a relation net <A, T>, we name the following sets for convenience.

(1) The set of all ***tuples in the name of*** A_m, denoted by $R(A_m)$, is the set of all those tuples in T that have A_m in them, at least once, other than as a first or last entry in the tuple, and
(2) the set of all ***tuples with*** A_m, denoted by $R[A_m]$, is the set of all those tuples in T that have A_m in them anywhere at least once.

Constructional Scheme 1.1.2: To determine $R(A_m)$, $R[A_m]$, and hence $|R(A_m)|$ and $|R(A_m)|$, for $A_m \in A$ in a relation net <A, T>, where |X| denotes the number of members of set X.

(1) List the tuples, in the tuples table T of <A, T>, that have at least one occurrence of A_m in them in any but the first and last position in the tuple. This is the list of members of $R(A_m)$. Count those members to determine $|R(A_m)|$.
(2) List the tuples, in the tuples table T of <A, T>, that have at least one occurrence of A_m in them in any position in the tuple. This is the list of members of $R[A_m]$. Count those members to determine $|R[A_m]|$. ♦

Comment: Implementation is obvious.

Next we introduce substructures of a relation net. Let <A, T> and <B, U> be relation nets. We say that <B, U> is a ***subnet*** of <A, T> if, and only if, $B \subseteq A$ and $U \subseteq T$. Thus every tuple in <B, U> is also in <A, T>, and it is evident that all the entries in the tuples of U are members of B because <B, U> is a relation net. We write <B, U> $\angle$ <A, T>.

Constructional Scheme 1.1.3: To check that <B,U> $\angle$ <A,T> for two given relation nets <A, T> and <B,U>.

(1) Check that every member of B belongs to A. If this is true then
(2) check that every tuple in U belongs to T. If this is also true, then <B,U> $\angle$ <A,T>.♦

Comment: Implementation is obvious.

With respect to "relational integratedness" of a point/vertex in a relation net, we have the following two worst cases. $A_m \in A$ is called an ***isolate*** in a relation net <A, T> if, and only if, $id(A_m) = od(A_m) = 0$ but $R(A_m) \neq \varnothing$, i.e. $|R(A_m)| \neq 0$. Thus A_m has no arrow to or from it, but it occurs in at least one tuple in <A, T>. A_m is called a ***complete isolate*** in a relation net <A, T> if, and only if, $id(A_m) = od(A_m) = |R(A_m)| = 0$, i.e. if, and only if, $R[A_m] = \varnothing$ so $|R(A_m)| = 0$. Thus A_m is not an entry anywhere in any tuple in T.

Constructional Scheme 1.1.4: To find the isolates and the complete isolates in a relation net <A, T>.

(1) From the tuples table T, find, by examining each $A_m \in A$ in turn, all those vertices that do not occur at the beginning of any tuple in T nor at the end of any tuple in T. These are the isolates of <A, T>, and some of them may be complete isolates.
(2) From the tuples table T, find, by examining each isolate found in (1), all those $A_m \in A$ that do not occur in any position in any tuple in T. These are the complete isolates of <A, T>, a subset of the set of isolates of <A, T>. ♦

Comment: Implementation is obvious.

Notice that by saying that $A_m \in A$ is in $T_j \in T$ in a relation net <A, T> we mean that A_m belongs to the set of entries in T_j. For all $T_j \in T$ in <A, T>, the set of entries of members of A in T_j is called the ***tuple set*** of T_j in <A,T>.

A subnet <B, U> of a relation net <A, T> is called a ***spanning subnet*** of <A, T> if, and only if, B = A and $U \subseteq T$.

Constructional Scheme 1.1.5: To check that relation net <B, U> is a spanning subnet of a relation net <A, T>.

(1) Check that every member of B is also a member of A, and that every member of A is also a member of B, i.e. that A = B.
(2) Check that every tuple in U is also a tuple in T, i.e. that $U \subseteq T$. ♦

Comment: Implementation is obvious.

To find A we need to read every entry in every tuple of T. We start with $A = \varnothing$ and add to A every entry that has not already been added.

A spanning subnet <A, U> $\angle$ <A, T> with U a selected subnet of T can be useful in examining the structural properties of <A, T>. Another useful subnet of a relation net <A, T> is described as follows:

Let relation net <B, U> be a subnet of a relation net <A, T>, i.e. $B \subseteq A$ and $U \subseteq T$. <B, U> is the ***maximum subnet induced by*** B in <A, T> provided that tuple $T_i \in T$ belongs to U if, and only if, every entry in T_i is a member of B. Thus <B, U> is made up of all the members of B together with precisely all the tuples that have all their entries in B, i.e. precisely all the tuples in T that have as their tuple set a subset of B. To emphasize the role of B we write <B, U> as <B, (T↑B)>, or simply <B, T↑B>, in this case, where T↑B is read "T restricted to members of B only".

Constructional Scheme 1.1.6: To find the maximum subnet <B,T↑B> of a relation net <A, T>, where $B \subseteq A$.

Read every tuple in T, and mark each tuple for which every entry is an entry of a member of B. These are the tuples of <B, T↑B>, and of course the vertex set is B. ♦

Comment: Implementation is obvious.

For all $A_r \in A$, the set of all vertices/points A_s such that there is an arrow from A_r to A_s in a relation net <A, T> is said to be ***adjacent from*** A_r in <A, T>. This set is regarded as trivially containing A_r itself.

Constructional Scheme 1.1.7: To find all the $A_n \in A$ that are adjacent from/to $A_m \in A$ in a relation net <A, T>.

(1) Find and mark all the tuples of T that begin with an entry of A_m.
(2) Mark all the members of A that occur at the end, i.e. as the last entry, of at least one of those tuples marked in (1). The marked members of A, together with A_m, are the vertices that are adjacent from A_m in <A, T>.

(3) Adjacency **to** A_m is similar. ♦

Comment: Implementation is obvious.

It is now time to turn our attention to "non-local relational integratedness" in a relation net. This hinges on the existence of "paths" in a relation net <A, T>, as paths chain tuples together to display more complex relationships than do the individual tuples.

By a ***walk*** in a relation net <A, T> we mean an alternating sequence of points/vertices and arrows, A_1, $<A_1, A_2>$, A_2, $<A_2, A_3>$, A_3, …, A_q, $<A_q, A_{q+1}>$, A_{q+1} written $A_1 \rightarrow A_{q+1}$, where for k = 1 , …, q each arrow is associated with one only label from the set of labels on the arrow $<A_k, A_{k+1}>$ from A_k to A_{k+1} in <A, T>. Neither the A_k nor the $<A_k, A_{k+1}>$ need be unique. The ***length*** of a walk is the number of arrow entries in it, in this case q. If all but possibly A_1 and A_{q+1} are distinct vertices, and all the arrows are distinct (and therefore each associated with a distinct member of its set of labels), then $A_1 \rightarrow A_{q+1}$ is called a ***path*** in <A, T>. If $A_1 = A_{q+1}$ for a path $A_1 \rightarrow A_{q+1}$ then we call $A_1 \rightarrow A_{q+1}$ a ***circuit***. Thus a circuit is a closed path. If at least one arrow in a walk is traversed in the reverse direction of that arrow then we call it a ***semi-walk***. We also refer to ***semi-paths***.

By a ***walk-family*** $f(A_r \rightarrow A_s)$ in a relation net <A, T> we mean a non-empty collection of walks from A_r to A_s, in <A, T>, the members of which all use the same points/vertices and the same arrows in the same order but are distinct by virtue of the arrow labels used on the arrows in those walks.

Let A_r, A_j and A_s be members of A in a relation net <A, T>, and let $A_r \rightarrow A_s$ be a given walk in <A, T>. Then A_j is said to be ***vertex between*** A_r and A_s on $A_r \rightarrow A_s$ if, and only if, A_j is a vertex on $A_r \rightarrow A_s$, or A_j belongs to at least one label on $A_r \rightarrow A_s$, or both. A_j is said to be ***reachable*** from A_r in <A, T> if, and only if, there is at least one path $A_r \rightarrow A_j$ in <A, T>. A walk $A_r \rightarrow A_s$ in a relation net <A, T> is said to ***go via*** the members of the set of precisely all those labels that each occur on at least one arrow of $A_r \rightarrow A_s$.

Next we describe ***context sensitivity*** in a relation net. If a set $B \subseteq A$ of a relation net <A, T> is deleted from <A, T> then every member of B is deleted from <A, T> and every tuple in which at least one member of B is an entry must also be deleted from <A, T>. As a result, certain arrows not incident with any vertex in B may disappear because all their labels disappear. This constitutes what we call context sensitivity in relation nets.

The ***meet*** <A, T> of two relation nets <B, F> and <C, G> is $<A, T> = <B \cap C, F \cap G>$. <A, T> is a unique relation net. We write <A, T> as $<B, F> \cap <C, G>$. The ***join*** <A,T> of two relation nets <B, F> and <C, G> is $<A, T> = <B \cup C, F \cup G>$. <A, T> is a unique relation net, and we write <A, T> as $<B, F> \cup <C, G>$. Thus the meet consists of the common vertices and all those common tuples with entries from the common vertices only, and the join has all the vertices and tuples of the constituent relation nets.

Constructional Scheme 1.1.8: Find all the vertices that are vertex between A_r and A_s on a given path $A_r \rightarrow A_s$ in a relation net <A, T>.

Mark all those vertices that appear at least once as an entry in at least one of the tuples used on $A_r \rightarrow A_s$. The set of vertices marked is the set we seek. ♦

Comment: Start with a set $B = \varnothing$. Read each tuple used on the path and add to B every entry

that has not already been added.

Constructional Scheme 1.1.9: Find all the vertices that are reachable from a given $A_r \in A$ in a relation net <A, T>.

(1) Start to build the set $V \subseteq A$ of all the vertices of A that are reachable from $A_r \in A$ in <A, T> by setting $V_0 = \{ A_r \}$.
(2) Find and mark every tuple in T that starts with A_r. Call this set of tuples T_1. Add to V_0 the end vertex of each member of T_1, getting V_1 with $V_0 \subseteq V_1$.
(3) Find and mark every tuple in $(T - T_1)$ that starts with a vertex in $(V_1 - V_0)$. Let this set of tuples be T_2. Add to V_1 the end vertex of each member of T_2, getting V_2 with $V_0 \subseteq V_1 \subseteq V_2$.
(4) Continue in this manner, getting to the stage at which we have defined T_n and V_n with $V_0 \subseteq V_1 \subseteq \ldots \subseteq V_n$ and $n \geq 1$. Now find and mark every tuple in $(T - T_n)$ that starts with a vertex in $(V_n - V_{n-1})$. Let this set of tuples be T_{n+1}. Add to V_n the end vertex of each member of T_{n+1}, getting V_{n+1} with $V_n \subseteq V_{n+1}$.
(5) Repeat 4, increasing n by 1 at each step, until $T_n = \varnothing$ for some n. Then $V = V_n$. ♦

Comment: Implementation is obvious as we read appropriate tuples and mark their last entries.

Notice that each vertex in $(V_n - V_{n+1})$, $n \geq 1$, can be reached from A_r by at least one path of length n in <A, T>.

Constructional Scheme 1.1.10: Find the join of two relation nets <B, F> and <C, G>.

(1) Find the vertex set $B \cup C$.
(2) Find the tuples set $F \cup G$.

The result is $\langle B, F\rangle \cup \langle C, G\rangle = \langle B \cup C, F \cup G\rangle$. ♦

Comment: Implementation is obvious, as finding the union of two sets is standard.

Constructional Scheme 1.1.11: Find the meet of two relation nets <B, F> and <C, G>.

(1) Find the vertex set $B \cap C$.
(2) Find the tuples set $F \cap G$.

The result is $\langle B, F\rangle \cap \langle C, G\rangle = \langle B \cap C, F \cap G\rangle$. ♦

Comment: Implementation is obvious, as finding the intersection of two sets is standard.

If tuple $T \in F \cap G$ then the tuple set of T is a subset of $(B \cap C)$, but the converse is not always true because T may belong to only one of F or G.

We now describe one of the central features of relation nets. Consider a relation net <A, T> and a collection of subnets, of $\langle A,T\rangle$, $\langle B_0, R_0\rangle \angle \langle B_1, R_1\rangle \angle \ldots \angle \langle B_k, R_k\rangle \angle \ldots \angle \langle B_n, R_n\rangle$. Thus, for each k = 0, 1, 2,…, n - 1 we have $\langle B_k, R_k\rangle \angle \langle B_{k+1}, R_{k+1}\rangle$, i.e. $B_k \subseteq B_{k+1}$ and $R_k \subseteq R_{k+1}$. Such a collection of subnets of <A, T> is called a ***fast access cascade*** from B_0 in <A, T> if, and only if, we have

a) $B_0 \subseteq A$ and $R_0 = \varnothing$.
b) A tuple in T belongs to R_1 if, and only if, the first entry of that tuple is that of a member of B_0.
c) B_1 is B_0 together with the union of the tuple sets of all the tuples in R_1, and, in general, for k = 2, 3 … ,
d) a tuple in T belongs to R_k if, and only if, the first entry of that tuple is of a member of B_{k-1} or that tuple belongs to R_{k-1} (so $R_{k-1} \subseteq R_k$).
e) B_k is B_{k-1} together with the union of the tuple sets of all the tuples in R_k (so $B_{k-1} \subseteq B_k$).

Such a cascade is called a ***limited access cascade*** from B_0 in <A, T> if, and only if, at each step k = 1,2,… , a tuple in T belongs to R_k if, and only if, every entry in that tuple but possibly the last entry is of a member of B_{k-1} or that tuple belongs to R_{k-1}. (That last entry may or may not be of a member of B_{k-1}.)

A cascade will stop when $<B_k, R_k> = <B_{k-1}, R_{k-1}>$ or when $<B_k, R_k> = <A, T>$. A cascade can be "run" interactively step-by-step, in which case the user can stop the cascade after completion of any step before automatic termination.

Constructional Scheme 1.1.12: To "run" a fast access cascade from a given $B_0 \subseteq A$ in a relation net <A, T>.

(1) Set $R_0 = \varnothing$.
(2) Let $R_1 \subseteq T$ be such that a tuple belongs to R_1 if, and only if, the first entry of that tuple is of a member of B_0. $R_0 \subseteq R_1$.
(3) B_1 consists of B_0 together with every $A_m \in A$ that occurs at least once in at least one member of R_1, i.e. B_0 together with the union of the tuple sets of all the tuples that belong to R_1. $B_0 \subseteq B_1$.
(4) $R_2 \subseteq T$ is chosen in such a way that a tuple belongs to R_2 if, and only if, the first entry of that tuple is of a member of B_1 or the tuple belongs to R_1. $R_0 \subseteq R_1 \subseteq R_2$.
(5) B_2 consists of B_1 together with every $A_m \in A$ that occurs at least once in at least one member of R_2, i.e. of B_1 together with the union of the tuple sets of all the tuples that belong to R_2. $B_0 \subseteq B_1 \subseteq B_2$.
(6) Assume that we continue in this manner, and have specified $<B_k, R_k>$, k > 1. $R_{k+1} \subseteq T$ is chosen in such a way that a tuple belongs to R_{k+1} if, and only if, the first entry in that tuple is of a member of B_k or the tuple belongs to R_k. $R_0 \subseteq R_1 \subseteq \ldots \subseteq R_k \subseteq R_{k+1}$.
(7) B_{k+1} consists of B_k together with every $A_m \in A$ that occurs at least once in at least one member of R_{k+1}, i.e. of B_k together with the union of the tuple sets of all the tuples that belong to R_{k+1}. $B_0 \subseteq B_1 \subseteq \ldots \subseteq B_k \subseteq B_{k+1}$.
(8) Repeat (6) and (7), increasing k by 1 at each step, until $<B_{k+1}, R_{k+1}> = <B_k, R_k>$ for some k or the option to stop for some k is chosen. ♦

Comment: Implementation is obvious as we read, and mark, vertices and tuples from the tuples table T of <A, T>.

Constructional Scheme 1.1.13: To "run" a limited access cascade from a given $B_0 \subseteq A$ in a relation net <A, T>.

(1) Set $R_0 = \varnothing$.
(2) Let $R_1 \subseteq T$ be such that a tuple belongs to R_1 if, and only if, every entry in that tuple,

but possibly the last entry, is of a member of B_0. $R_0 \subseteq R_1$.

(3) B_1 consists of B_0 together with every $A_m \in A$ that occurs at least once in at least one member of R_1, i.e. of B_0 together with the union of the tuple sets of all the tuples that belong to R_1. $B_0 \subseteq B_1$.

(4) $R_2 \subseteq T$ is chosen in such a way that a tuple belongs to R_2 if, and only if, every entry in that tuple, but possibly the last entry, is of a member of B_1 or the tuple belongs to R_1. $R_0 \subseteq R_1 \subseteq R_2$.

(5) B_2 consists of B_1 together with every $A_m \in A$ that occurs at least once in at least one member of R_2, i.e. of B_1 together with the union of all the tuple sets of all the tuples that belong to R_2. $B_0 \leq B_1 \leq B_2$.

(6) Assume that we continue in this manner, and have specified $<B_k, R_k>$, $k \geq 1$. $R_{k+1} \subseteq T$ is chosen in such a way that a tuple belongs to R_{k+1} if, and only if, every entry in that tuple, but possibly the last entry, is of a member of B_k or the tuple belongs to R_k. $R_0 \subseteq R_1 \subseteq \ldots \subseteq R_k \subseteq R_{k+1}$.

(7) B_{k+1} consists of B_k together with every $A_m \in A$ that occurs at least once in at least one member of R_{k+1}, i.e. of B_k together with the union of the tuple sets of all the tuples that belong to R_{k+1}. $B_0 \subseteq B_1 \subseteq \ldots \subseteq B_k \subseteq B_{k+1}$.

(8) Repeat (6) and (7), increasing k by 1 at each step, until $<B_{k+1}, R_{k+1}> = <B_k, R_k>$ for some k or the option to stop for some k is chosen. ♦

Comment: As for CS 1.1.12, but here we choose the vertices and tuples marked in a different way.

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

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

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

(3) $B_1 = B_0 \cup$ (the union of the tuple sets of all the members of R_1), and in general for k = 2,3,…,

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

(5) $B_k = B_{k-1} \cup$ (the union of the tuple sets of all 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 $T_i = <A_1, A_2, \ldots, A_\ell, \ldots, A_{n(i)}> \in T$ in such a way that $T_i \in R_k$ iff $\{A_m \in A \mid m = 1,2,\ldots,n(i)-1\} \subseteq B_{k-1}$, or $T_i \in R_{k-1,}$ and where $A_{n(i)} \in A$ may or may not belong to $B_{k-1.}$

Note: When “running” or “generating” a cascade we mark each label as it is used so that no label is ever repeatedly “found”.

Consider a relation net <A, T>, and any $A_k \in A$. A subnet that has as its only tuples precisely all those tuples of T that have A_k as at least one entry in them, i.e. the tuples of $R[A_k]$, together with the union of the tuple sets of all those tuples, is called the ***context-net*** of A_k in <A, T>. It is written <A, T> $[A_k]$, and it is of course a relation net: In fact it is the minimum subnet of <A, T> that is ***induced by*** $R[A_k]$,i.e. induced by the union of the tuple sets of all the tuples of $R[A_k]$, in <A, T>. <A,T>$[A_k]$ contains all the information about A_k within the context of <A, T>, and to delete A_k from <A, T> means to delete <A, T>$[A_k]$ from <A, T>.

Constructional Scheme 1.1.14: Find the context-net $<A, T>[A_k]$ in a relation net $<A, T>$.

(1) Find the set $R[A_k] \subseteq T$ of all tuples in which A_k occurs as at least one entry. This is the tuple set of $<A, T>[A_k]$.
(2) Take the union of the tuple sets of all the members of $R[A_k]$. This is the vertex set of $<A, T>[A_k]$. ♦

Comment: Implementation of both instructions of CS 1.1.14 is obvious - see CS 1.1.2 for part 1.

1.2 Concept-Relationship Knowledge Structures (CRKS's)

We deal in this work mainly with two constrained structures. The first is a special kind of CNR-net, which is itself a special kind of relation net. The second is a special kind of hypernet that is to function as a model of a NET for Education, which we will meet in chapter 4. The first must, as we will see, be dealt with before moving on to the second. They have several similarities. The CNR-net structure is fairly extensively covered, along with some of its applications in teaching and learning, in Part I of [GVS99], so we present only a summary of that work, with certain revisions.

A CNR-net <A, T> is called a ***formal schema*** if, and only if,

(1) each concept-name vertex is related to at least one other concept-name vertex in <A, T>, and
(2) <A, T> has no circuits, and
(3) there is at least one concept-name vertex that has in-degree zero and out-degree non-zero, each such vertex being called a ***primary*** of <A, T>, and
(4) there is at least one concept-name vertex that has in-degree non-zero and out-degree zero, each such vertex being called a ***goal*** of <A, T>.

Constructional Scheme 1.2.1: To find the primaries of a potential formal schema <A, T>.

Let $A_k \in A$. A_k is a primary of <A, T> if, and only if, A_k is the first entry of at least one $T_i \in T$ and is not the last entry of any $T_j \in T$. Choose any $A_k \in A$ and check it for primary status in <A, T>. Do this for each $A_k \in A$. <A, T> must have at least one primary. ♦

Comment: Implementation is obvious, checking the first and last entry of each tuple for each $A_k \in A$ and rejecting each inappropriate, i.e. non-primary, entry, while marking each primary found by reading the first and last entry of every tuple without prior rejection.

Constructional Scheme 1.2.2: To find the goals of a potential formal schema <A, T>.

Let $A_k \in A$. A_k is a goal of <A, T> if, and only if, A_k is the last entry of at least one $T_i \in T$ and is not the first entry of any $T_j \in T$. Choose any $A_k \in A$ and check it for goal status in <A, T>. Do this for each $A_k \in A$. <A, T> must have at least one goal.♦

Comment: As for CS 1.2.1 with appropriate modification.

Constructional Scheme 1.2.3: To check that each $A_k \in A$ in a potential formal schema <A, T> is related to at least one $A_j \in A$, $A_j \neq A_k$, in <A, T>.

(1) Check that each $T_i \in T$ has at least two entries of at least two distinct members of A.
(2) Choose any $A_k \in A$ and check that A_k appears in at least one $T_i \in T$ in which there is at least one entry of some $A_j \neq A_k$. Do this for each $A_k \in A$. ♦

Comment: Implementation is obvious, by reading the tuples table.

Constructional Scheme 1.2.4: To determine whether or not there are circuits of any length in a relation net <A, T>.

(1) Consider the diagram of <A, T>. Delete from it every arrow label, leaving a vertex labelled directed graph <A, P> where P is a set of ordered pairs of members of A, the arrows of the digraph. $A_k \in A$ lies on at least one circuit in <A, T> if, and only if, it lies on at least one circuit in <A, P>, because pair $<A_k, A_j>$ belongs to P if, and only if, there is an arrow from A_k to A_j in <A, P> and hence also in <A, T>.

(2) In <A, P> a fast access cascade and a limited access cascade from $B_0 \subseteq A$ will be identical: Both just follow reachability in <A, P> from $A_k \in A$ to $\Gamma(A_k)$ to $\Gamma(\Gamma(A_k))$ to $\Gamma(\Gamma(\Gamma(A_k)))$ and so on step-by-step, for each $A_k \in B_0 \subseteq A$, where Γ is the adjacency function of <A, P>. To find if $A_k \in A$ lies on at least one circuit in <A, P>, and thus also in <A, T>, we choose $B_0 = \{A_k\} \subseteq A$. $R_0 = \varnothing$. Run a cascade from B_0 in <A, P> as follows. Choose for $R_1 \subseteq P$ all the pairs $<A_k, A_j>$, $A_j \neq A_k$, that start with A_k. Run the cascade step-by-step.

(3) If $A_k \in (B_1 - B_0)$ then A_k lies on at least one circuit, in <A, P> and hence in <A, T>, of length 1. If not proceed to step 2 and so on. If $A_k \in (B_2 - B_1)$ then A_k lies on at least one circuit, in <A, P> and hence in <A, T>, of length 2.

(4) In general if $A_k \in (B_n - B_{n-1})$ then A_k lies on at least one circuit, in <A, P> and hence in <A, T>, of length n, $n \geq 1$.

(5) If the cascade stops, as it must, without finding A_k in any $(B_k - B_{k-1})$ for some $k \geq 1$, then A_k does not lie on any circuit in <A, P>, and thus in <A, T>.

(6) Repeat for each $A_k \in A$ to check that no $A_k \in A$ lies on any circuit in <A, P>, and hence in <A, T>. ♦

Comment: To convert <A,T> to the directed graph <A, P> we read each tuple in T and convert it to an ordered pair $<A_i, A_j>$ for each tuple $T_k \in T$, where A_i is the first entry of T_k and A_j is the last entry of T_k. Note that we assume that $A_i \neq A_j$ in each case, i.e. that <A, T>, and hence also <A, P>, has no loops. For the rest, we run a cascade from $B_0 = \{A_m\}$ for each $A_m \in A$ in turn, to see if A_m lies on a circuit. If any $A_n \in A$ does lie on a circuit, stop. <A, T> is not a formal schema. See CS 1.2.12 or CS 1.2.13: It is clear, from either of those, how to "run" a cascade in <A, P>, in which all tuples are just ordered pairs.

The intention is that the primary concept-names are assumed to be "familiar" and that the goal concept-names are those that can be "generated" by relating them to primary concept-names via "intermediate" or secondary concept-names that are related to primaries and among each other.

A formal schema <A, T> is said to be ***complete*** if it has no isolates. (It can of course then not have any complete isolates.) <A, T> is said to be ***connected*** if, and only if, (i.e. precisely if) there is at least one semi-path between every pair of concept-name vertices in <A, T>.

Constructional Scheme 1.2.5: To check that a given formal schema <A, T> is complete.

Consider any $A_k \in A$. Read the first and last entries in each tuple $T_i \in T$. We must find that A_k is either a first entry or a last entry, but of course not both, in at least one tuple. Check every $A_j \in A$ in turn. Every $A_j \in A$ must occur as a first, or a last, entry in at least one tuple $T_i \in T$. ♦

Comment: Implementation is obvious.

Theorem 1.2.1: If a formal schema <A, T> is connected then it is complete, but the converse

is not generally true. ♦

Proof: See page 47 of [GVS99]. ♦

By the ***context schema*** <A, T> [A_r] of $A_r \in A$ in a formal schema <A, T> we mean the minimum subnet of <A, T> that is induced by R[A_r], i.e. that subnet that has precisely the tuples of R[A_r] and precisely every concept-name that belongs to the tuple set of at least one of the tuples in R[A_r], i.e. the union of the tuple sets of the tuples that belong to R[A_r].

In a formal schema <A, T>, "concept-name A_r" refers to the vertex $A_r \in A$ only, while "concept A_r" refers to <A, T> [A_r]. <A, T> [A_r] tells us all there is to know about A_r within the context of <A, T>. Deletion of A_r from <A, T> entails the deletion of the whole context schema <A, T> [A_r] from <A, T>.

Constructional Scheme 1.2.6: To find the context schema of a given vertex $A_k \in A$ in a formal schema <A, T>.

Comment: See CS 1.1.14.

We now move on to explicating what we mean by "generating" concept-names from the primaries of a complete formal schema <A, T>. This hinges on what we call ***derivability*** in a complete formal schema. If we are going to follow paths in complete formal schemas then we must be sure that in doing so we meet concept-names along the path and in the labels on the path in "an orderly fashion". By this we mean that we must follow the Ausubel approach ([ANH78], [Aus63], [Aus 80]) in which every newly met concept-name in such a path, whether on the path or in a label on the path, is related to previously met concept-names. This leads to the notion of a ***derivation path*** in a complete formal schema.

Here is an informal description of what we mean by a derivation path, followed by some informal comments.

(1) Every primary is trivially derivable by means of a derivation path of length zero.
(2) Every derivation path must start with a primary.
(3) Every arrow of a derivation path starts at a derived vertex.
(4) Every member of every tuple used to label an arrow on a derivation path, barring the first and last entries of the tuple, must be primary, **or** a vertex immediately derived from a set of vertices met previously on the path, by which we mean that there is, somewhere in the relevant formal schema, an arrow that starts with one of those previously met vertices and uses only previously met vertices in its tuple label and ends with the vertex with which we are concerned, **or** a hypothesis, which is a "new" (on or in the path) vertex that can be derived somewhere in the formal schema.
(5) The derivation path that derives a hypothesis may have hypotheses on or in it. Each of those must be derived somewhere in the formal schema. This recursive process must stop because the formal schema is finite and has at least one primary.

First we take every primary in every complete formal schema <A, T> to be a "known" concept-name with respect to <A, T>. Thus we assume that all primaries in <A, T> have partially established meanings from prior complete formal schemas or are primitive concept-names, i.e. are established solely by examples. (Partial establishment in prior complete formal schemas is assumed for all unmarked words and phrases in the statements from which the tuples of T arise.) Adding statements, and hence tuples, that involve a particular concept-name $A_r \in A$ to <A, T> enriches the meaning of concept A_r in the resulting formal schema, a

point to which we return later.

It can be shown that in any complete formal schema <A, T> every concept-name $A_r \in A$ is either primary, or a goal, or there is at least one primary $p \in A$ and at least one goal $g \in A$ such that A_r lies on at least one path $p \rightarrow g$ in <A, T>. Further, in every complete formal schema <A, T> there is at least one path from each primary of <A, T> to some goal of <A, T>, at least one path to each goal of <A, T> from some primary of <A, T>, and at least one path to each non-primary vertex of <A, T> from at least one of the primaries of <A, T> [Wol82]. Formally, each primary of <A, T> is (trivially) derived in <A, T> by a path of length zero. Our aim is to make every path in <A, T> a derivation path so that every vertex of <A, T> is a derived vertex. If <A, T> is a complete formal schema then the required paths are there, but can we force all paths to be derivation paths?

A tuple of a complete formal schema <A, T> is called a ***derivation tuple*** if, and only if, every entry in it, but possibly the last one, is either a primary of <A, T> or is the last entry of at least one other derivation tuple in <A, T>. The last entry of a derivation tuple in <A,T> can be any non-primary of <A, T>, and it may have been previously derived by means of some other derivation tuple. We can describe a ***Concept-Relationship-Knowledge Structure*** (CRKS) as a complete formal schema <A, T> in which every tuple $T_i \in T$ is a derivation tuple. We can see that in a CRKS there must be at least one derivation tuple in which every entry but the last one is an occurrence of a primary of <A, T>. Such a tuple is indeed a derivation tuple, and the last entry in the first such tuple that we construct is the first non-primary ***derived concept-name*** (vertex) of <A, T>. At this stage of the construction of a complete formal schema <A, T>, which is to be a CRKS, we have all the primaries of <A, T> and at least one non-primary derived vertex with which to continue the construction. We can now define derivation tuples, choosing appropriate statements and permutations that involve the concept-names that comprise A. Every entry but the last one in every tuple defined for <A, T> is either primary in <A, T> or an already derived concept-name of <A, T>, and the final entry of each such tuple becomes another, not necessarily "new", derived concept-name. Since <A, T> must be complete, every non-primary vertex of <A, T> must end up as a derived concept-name, so we construct a complete formal schema in which every vertex is a derived vertex. Any path that starts at a primary, and in which every tuple used in a label on the path is a derivation tuple, is called a ***derivation path***, and every vertex that lies at the end of a derivation path (including the trivial length zero paths to the primaries) in <A, T> is called a ***derived vertex*** of <A, T>.

It is relatively easy to prove the following. In any complete formal schema <A, T>:

- A path $p \rightarrow t$, p any primary and t any non-primary of <A, T>, is a derivation path in <A, T> if, and only if, every vertex on $p \rightarrow t$ and every entry in every label on $p \rightarrow t$ is a derived vertex in <A, T>.
- Every path $p \rightarrow t$, p any primary and t any non-primary of <A, T>, is a derivation path in <A, T> if, and only if, every vertex in <A, T> is a derived vertex in <A, T>.
- Every path $p \rightarrow t$, p any primary and t any non-primary of <A, T>, is a derivation path in <A, T> if, and only if, every tuple of <A, T> is a derivation tuple.
- A formal schema <A, T> is complete and every vertex of <A, T> is derived if, and only if, every non-primary vertex of <A, T> is a last entry in at least one derivation tuple in <A, T> and every non-goal vertex of <A, T> is a first entry in at least one derivation tuple in <A, T>, and every tuple in <A, T> is a derivation tuple. ♦

Proof of the fourth theorem above is presented on pages 148 and 149 of [GVS99]. With this background we can make a precise statement of what we mean by a CRKS.

A complete formal schema <A, T> is a **CRKS** if, and only if, every vertex $A_k \in A$ is a derived vertex in <A, T>. What is the significance of derivation in a complete formal schema? Derivation means that every concept-name in a CRKS is associated with primaries or other concept-names that have been previously met in that CRKS. Every non-primary concept-name in a CRKS is eventually related to the (partially) "known"/"familiar" primaries of that CRKS. This relationship is based on paths, so we had better know something about the paths in a complete formal schema!

Constructional Scheme 1.2.7: To construct a path tree, for a formal schema <A, T>, that displays and distinguishes every path from each primary in <A, T>. We refer to the vertices and arrows of <A, T> and to the nodes and branches of the path tree.

(1) Introduce an unlabelled dummy node to serve as the root of the path tree.
(2) Introduce one only node for each primary of <A, T>, and connect each such node to the root with an unlabelled branch. Label each non-root node with the appropriate primary concept-name from <A, T>.
(3) From each node for a vertex $A_k \in A$ the tree now develops as follows. Find every tuple in T that starts with A_k. Suppose that such a tuple is $<A_k = c_1, c_2, \ldots, c_j, \ldots, c_{n-1}, c_n>$. We now plot a new node for c_n for each tuple that starts with A_k and ends with $c_n \in A$, and insert a branch from each node for A_k to every node for c_n. Each branch is then labelled with the tuple that generates it, and each node for c_n is labelled with the concept-name c_n.
(4) Repeat (3) for every $A_k \in A$ and for every relevant tuple in T.
(5) The resulting tree exhibits, along each path from the root, every path from a primary to a goal in <A, T>, and distinguishes these paths. Each primary of <A, T> is represented by one only node, and each goal of <A, T> by at least one node. This constructional scheme works because every tuple in T is displayed on a separate labelled branch. ♦

Comment: Construction of the path tree for <A, T> can clearly be computer assisted in finding the appropriate tuples of part (3) from T. From there we can construct the path tree from those tuples.

Furthermore, we have the following for paths in a formal schema.

Constructional Scheme 1.2.8: To find all the paths of length ≥ 1 from $A_m \in A$ to $A_n \in A$ in a formal schema <A, T>.

(1) Run a fast access cascade backwards (reversing all arrows) from $B_0 = \{A_n\}$ in <A, T>. Let the resulting subnet be <A', T'> with $A' \subseteq A$ and $T' \subseteq T$. If A_m is not a member of A' then there are no $A_m \rightarrow A_n$ paths in <A, T>.
(2) If $A_m \in A'$ then proceed as follows in <A', T'>. Revert all the arrows of <A', T'> to their original direction. Find all the tuples in T' that start with A_m. Let these tuples be $T_1, T_2, \ldots, T_j$, and let their last entries be $v_1, v_2, \ldots, v_{j-1}, v_j$ respectively. Each time $v_k = A_n$ we have found an $A_m \rightarrow A_n$ path of length 1. Mark each such tuple as an $A_m \rightarrow A_n$ path of length 1 in <A, T>.
(3) Find all the unmarked tuples in T' that start with any vertex $v_k \neq A_n$ among the tuples found in step 2. We now plot a tree as follows. Plot a node for A_m and one only node for each of the v_k found in step 2. Join A_m to each v_k with a branch labelled T_k. Now insert a branch from each $v_k \neq A_n$ to a node for the last entry w_f of any tuple in T' that starts with this v_k and ends with w_f. If any of these w_f is A_n then we find all the $A_m \rightarrow$

A_n paths of length 2 in <A', T >, and hence in <A, T>. Mark all the tuples used in this step to find $A_m \rightarrow A_n$ paths of length 2. We have now marked all $A_m \rightarrow A_n$ paths of lengths 1 and 2 in <A', T'>, and hence in <A, T>. Proceed to step 4 with all the unmarked tuples in T' and all the w_f that are not A_m.

(4) Repeat step 3 for the next level of the tree, marking all the unmarked tuples of T' that are used in each stage of the generation of $A_m \rightarrow A_n$ paths of lengths 3, 4, …, if any, until all the useable tuples in T' have been marked , i.e. until we have exhausted T', or until we can mark no more tuples in T' by this procedure. The tree will distinguish and display all $A_m \rightarrow A_n$ paths in <A,T>. ♦

Comment: To run a fast access cascade backwards we follow CS 1.2.12, but read each tuple from last entry to first entry, i.e. we start by reversing the order of each tuple in T. For step (2) of CS 1.2.8 we restore the order of each tuple, and it is clear that a computer that stores T can find the relevant tuples for step (2). To continue to part (3), the computer can easily find the relevant unmarked tuples, from which we can plot the required tree as we cycle through steps (3) and (4) of CS 1.2.8.

A very strong property of a formal schema is connectedness; a property that may be desirable in some applications. Recall that if a formal schema is connected then it is complete.

Constructional Scheme 1.2.9: Test to see if a formal schema <A, T> is connected.

(1) Choose any $A_m \in A$ of <A, T>, and mark A_m. There must be at least one semi-path between A_m and each $A_n \neq A_m$ in <A, T> for <A, T> to be connected.

(2) Suppress all arrow heads in <A, T>, i.e. ignore direction in <A, T>. Further, delete all arrow labels in <A, T>. We then have a graph <A, G>, where G is the set of unordered pairs of members of A that arise from the suppression of arrow heads and the deletion of arrow labels from (the diagram of) <A, T>.

(3) Choose any $A_n \neq A_m$ in A. Run a cascade from $B_0 = \{A_m\}$ in <A, G>. If, at some step in the cascade, we find A_n then we mark A_n. If we do not find A_n at some step in the cascade then <A, T> is not connected.

(4) Repeat (3) for each unmarked vertex of A. If every vertex of A turns out to be marked then <A, T> is connected: Otherwise <A, T> is not connected. ♦

Comment: To construct <A, G> we read every tuple in T and extract from it the first and last entry. Suppose these are A_i and A_j for a particular tuple: Then we introduce the unordered pair $\{A_i, A_j\}$, and these pairs are the edges of the graph <A, G>. We need not produce a diagram of <A, G>. We then run a cascade - see CS 1.1.12 and CS 1.1.13, which are identical in <A, G> - in part (3). The computer can store the edges of <A, G> and run the cascade in each case as we cycle through steps (3) and (4) of CS 1.2.9.

Next we describe ***derivability*** formally.

A ***betweenness sequence*** for a path-family $f(A_1 \rightarrow A_n)$ in a formal schema <A, T> is found as follows. First, for all members of each $\lambda(<A_i, A_{i+1}>)$, i = 1,2,…,n-1, for each vertex adjacency (arrow) in $f(A_1 \rightarrow A_n)$ we list A_i, $\check{T}_{i1}$, $\check{T}_{i2}$, …, $\check{T}_{ir}$, …, $\check{T}_{im(i)}$, A_{i+1} where $\check{T}_{ir}$, r = 1,2,…,m(i), is the tuple set of $T_{ir} \in T$ for each $T_{ir} \in \lambda (<A_i, A_{i+1}>)$. Thus we have a sequence of vertices over A that starts with A_i and ends with A_{i+1}. Concatenating these sequences for each vertex adjacency of $f(A_1 \rightarrow A_n)$, in their order of appearance in $f(A_1 \rightarrow A_n)$, i.e. in the order of the vertex adjacencies in $f(A_1 \rightarrow A_n)$, we produce a betweenness sequence for $f(A_1 \rightarrow A_n)$. A betweenness sequence for a given path-family is not unique.

Definition 1.2.1 (*Derivability*):

(i) Given any formal schema <A, T> and a set $X \subseteq A$ we say that $A_r \in A$ is ***immediately derived from hypotheses*** X iff there is at least one tuple $T_i \in T$ by which there is a vertex adjacency x, T_i, A_r with $x \in X$ and with every member of $(\check{T}_i - \{A_r\})$ a member of X, where $\check{T}_i$ is the tuple set of $T_i \in T$.

(ii) Given any formal schema <A, T> and a set $X \subseteq A$ we say that $A_r \in A$ is ***derivable in terms of hypotheses*** X in <A, T> iff there is a path $A_j \rightarrow A_r$, $A_j \in A$, in <A, T> such that there exists at least one betweenness sequence S for $A_j \rightarrow A_r$ with the property that for every $A_i \in S$ we have

a) A_i is a primary of <A, T> or

b) $A_i \in X$ or

c) A_i is immediately derived from a subset of the set of all predecessors of A_i in S.

(ιι) We say that $A_r \in A$ is ***derivable from*** $P \subseteq A$ in a formal schema <A, T>, or simply ***derivable*** in <A, T>, where P is the set of all primaries of <A, T>, iff A_r is derivable in terms of some $X \subseteq A$ with either $X = \varnothing$ or such that every $x \in X$ is derivable (from P).

(iv) If $A_r \in A$ is derivable in a formal schema <A, T> by virtue of a path $A_j \rightarrow A_r$, where A_j is derivable in <A, T>, then $A_j \rightarrow A_r$ is called a ***derivation path*** for A_r in <A, T>, and each such path to A_r is called a derivation path for A_r in <A, T>, and A_r is said to be a ***derived vertex*** of <A, T>.♦

We have, for the recursive notion of derivability in a complete formal schema, the usual corresponding principle of induction.

Lemma 1.2.1: Let <A, T> be any CNR-net that is circuit free and has no isolates. Then <A, T> has at least one primary and at least one goal. ♦

Proof: Suppose that <A, T> has no primaries. Then every $A_r \in A$ has at least one arrow incident to it. But then A_r lies on at least one closed walk, and hence on at least one circuit in <A, T>, which is impossible. Thus <A, T> has at least one primary. A similar argument shows that <A, T> has at least one goal. ♦

Lemma 1.2.2: (***The induction principle for CRKS's***.) Let <A,T> be any CNR – net. If <A,T> has at least one primary and has no isolates, and if for an arbitrary non-goal A_s that is derivable in <A, T> we have that every $<A_s, A_t>$ in <A, T> is such that A_t is derivable by virtue of at least one derivation path that terminates with $<A_s, A_t>$, then every $A_r \in A$ is derivable in <A, T>, i.e. <A, T> is a CRKS. ♦

Proof: Suppose that A_s is a primary of <A, T>. Then, by the induction hypothesis, every A_t that is adjacent from A_s in <A, T> is derivable in <A, T>. Next let A_s be any vertex that is adjacent from a primary of <A, T>. A_s is derivable, so every A_t that is adjacent from this A_s in <A,T> is derivable in <A,T>, again by the induction hypothesis. Continuing these steps we must eventually reach every goal of <A, T> and show that every vertex $A_r \in A$ is derivable in <A, T>. Further, it is easy to see that every path in <A, T> must be a derivation path . <A, T> is a CRKS. ♦

It is easy to see that if the induction hypothesis holds then <A, T> can be generated from its primaries by a limited access cascade.

In section 4.1 of [GVS99] we proved theorems about formal schemas, which we include here.

Theorem 1.2.2: If $a \in A$ in a formal schema <A, T> is derivable in <A, T> by virtue of a derivation path $p \rightarrow a$, where p is immediately derived from a set of hypotheses $X = \varnothing$, the empty set, then p is a primary of <A, T>. ♦

Proof: p is immediately derived from $X = \varnothing$. The only vertices that can be immediately derived from $\varnothing$, trivially by a derivation path of length zero, are the primaries and isolates of <A, T>. Since $od(p) \neq 0$, p must be a primary of <A, T>.♦

Theorem 1.2.3: A complete formal schema <A, T> can be generated by a limited access cascade from its set of primaries if, and only if, (i.e. precisely if) every $A_r \in A$ is derivable in <A, T>. ♦

Proof: If <A,T> is generated by a limited access cascade from its primaries then, in each step, new vertices in A are generated in terms of primaries and preiously generated vertices of <A,T>. It follows at once that each new vertex generated is derivable (and of course each primary is derivable). Conversely, if <A,T> is complete formal schema in which every vertex is derivable then <A,T> can be generated from its primaries by a limited access cascade as follows: Let $B_0 \subseteq A$ be the set of primaries of <A,T>, and set $R_0 = \varnothing$. For each k = 1, 2, … we let $T_i \in R_k$, $T_i \in T$, iff $T_i = <A_1,\dots, A_\ell, \dots, A_{n(i)}>$ with $\{A_m \mid m = 1, 2, \dots, n(i) - 1\} \subseteq B_{k-1}$, where $A_{n(i)}$ may or may not belong to B_{k-1}. Suppose that we have reached step n in the limited access cascade, and consider step n + 1. Assume that we are confronted with an arrow <x, y> with an $A_j \in A$ in its label with $A_j \notin B_n$. Certainly $x \in B_n$. A_j is known to be derivable, so let the cascade, in steps k with $k \geq n$, follow a derivation path to A_j.

It must be able to do this from the definition of a derivation path, and since <A, T> is finite and circuit free there can be no infinite regression. Suppose then that the cascade reaches all such A_j in the label of <x, y> in step $m \geq n$. Then we can follow <x, y> in step m + 1 of the cascade. The cascade will stop precisely when it has generated all of <A, T>. ♦

Theorem 1.2.4: If vertex $A_s \in A$ is derivable in a complete formal schema <A, T> by virtue of a derivation path $A_r \rightarrow A_s$, with A_r primary in <A, T>, and a betweenness sequence S for $A_r \rightarrow A_s$, then every $s \in S$ is derivable in <A, T>. ♦

Proof: Let S be $s_0 = A_s, s_1, \dots, s_m, s_{m+1}, \dots, s_n = A_r$. $s_0 = A_s$ is a primary so it is immediately derived from hypotheses $X = \varnothing$ by a derivation path of length zero, so it is derivable in terms of hypotheses $X = \varnothing$, so it is derivable. s_1 is a primary so it is derivable, **or** it is a member of a set of hypotheses X for the derivation of A_r and hence, since $X \neq \varnothing$, s_1 is derivable because A_r is, **or** s_1 is immediately derived from hypotheses X, X being a subset of all its predecessors in S, i.e. it is immediately derived from $\{s_0\}$. Then it follows that, somewhere in <A, T>, there is a vertex adjacency s_0, T_i, s_1 with $T_i = <s_0, s_1>$ so that $\check{T}_i - \{s_1\} = \{s_0\} \subseteq X$. But since every member of X, i.e. s_0, is derivable, s_1 is derivable in this case too. Next assume that s_i is derivable for every i = 0,1,…,m, and consider s_{m+1}. Now either s_{m+1} is a primary, and hence derivable, **or** it is a member of a set of hypotheses X for the derivation of A_r, and, since $X \neq \varnothing$, s_{m+1} is derivable because A_r is derivable, **or** s_{m+1} is immediately derived from hypotheses X where X is a subset of all the predecessors of s_{m+1} in S, so s_{m+1} is derivable in <A, T> because, by the induction hypothesis, every member of this X is derivable in <A, T>. The theorem follows by the principle of induction. ♦

Theorem 1.2.5: Let <A, T> be a complete formal schema. Every vertex of <A, T> is

derivable in <A, T> if, and only if, every path $A_s \rightarrow A_r$, A_s some primary, in <A, T>, is a derivation path. ♦

Proof: Given on page 57 of [GVS99].

Theorem 1.2.6: Let <A, T> be a complete formal schema. <A, T> can be generated by a limited access cascade from its set of primaries if, and only if, every path from a primary $A_r \in A$ is a derivation path in <A, T>. ♦

Proof: Follows from theorems 1.2.4 and 1.2.5.

Lemma 1.2.3: Let <A, T> be a complete formal schema, and let $A_s \in A$ be an arbitrary primary of <A, T> and $A_r \in A$ a vertex with $id(A_r) \neq 0$ such that there is a path $A_s \rightarrow A_r$ in <A, T>. Then $A_s \rightarrow A_r$ is a derivation path iff every $A_t \in A$ that is vertex between A_s and A_r in $A_s \rightarrow A_r$ is derivable in <A, T>.♦

Proof: Bearing theorem 1.2.5 in mind, we need only show that if every A_t is derivable in <A,T> then $A_s \rightarrow A_r$ is a derivation path. Suppose that this is not so, i.e. $A_s \rightarrow A_r$ is not a derivation path. Then there is at least one A_t that is vertex between A_s and A_r and that is not derivable in <A, T>. The theorem follows from this contradiction. ♦

Definition 1.2.2: A complete formal schema <A, T> is called a ***Concept-Relationship Knowledge Structure***, or simply a ***CRKS***, iff every vertex of <A, T> is derivable in <A, T>.♦

Corollary 1.2.1: A complete formal schema <A, T> is a CRKS iff <A, T> can be generated by a limited access cascade from its primaries. ♦

Proof: Follows at once from theorem 1.2.6 and the definition of a CRKS. ♦

In summary we have: Let <A, T> be a complete formal schema. Then every $A_r \in A$ is derivable in <A,T>

- if, and only if, every path from a primary in <A,T> is a derivation path
- if, and only if, precisely the whole of <A,T> can be generated by a limited access cascade from its primaries.

Finally now, a complete formal schema <A, T> is called a ***Concept-Relationship Knowledge Structure***, or simply a ***CRKS***, if, and only if, every vertex of <A, T> is derivable in <A, T>. It then follows that a complete formal schema is a CRKS if, and only if, it can be generated from its primaries by a limited access cascade.
We thus have available an automated test to see if a given complete formal schema is a CRKS: A complete formal schema is a CRKS if and only if it can be ***inductively generated*** from its primaries by a limited access cascade.

Constructional Scheme 1.2.10: To test a complete formal schema <A, T> for CRKS form.

Let B_0 be the set of all primaries of <A, T>, and let R_1 be the set of all tuples of T for which every entry, but possibly the last entry, is in B_0. Run a limited access cascade from B_0 in <A, T>. <A, T> is a CRKS if, and only if, precisely the whole of A and precisely the whole of T are generated. ♦

Comment: See CS 1.1.13.

The properties of a CRKS are expressed, informally, on pages 58 and 59 of [GVS99], and also, in more detail, but still informally, on pages 145 to 149 of [GVS99]. In a CRKS the data items are concept-names, the information items are the tuples, and every (sub-)CRKS is a knowledge item. For examples of CRKS's see the references in Appendix 2.

We distinguish between presentation modes, such as blackboard and chalk, audio and video tapes or discs, computer presentation with guides, laboratory demonstrations and experiments etc, and presentation strategies, i.e. specific "traverses" of a selection of study material. Not even the most innovative presentation modes can overcome the problem of badly structured study material. We need a pattern in the study material, and hence patterned presentation strategies, to promote understanding of the material rather than rote learning. We achieve this by displaying an inherent "derivation pattern" in all study material. The (structural) models we use are special NET's.

1.3 Towards Hypernets

Early work on the system that was to become known as a relation net - see [VR76], [Wol82], [Wei83], [Gel93], [SVR93] - 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 [Wei83] for example).

To begin to introduce hypernets, we present a 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 (i.e. credit must be obtained) 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 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 that 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 of the parallel units, for example with an underline. We number each occurrence 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 can be an example of a hypernet.

Such a curriculum system for a host of “small” course units has pro’s and con’s. Its 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 planning, 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 that 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:

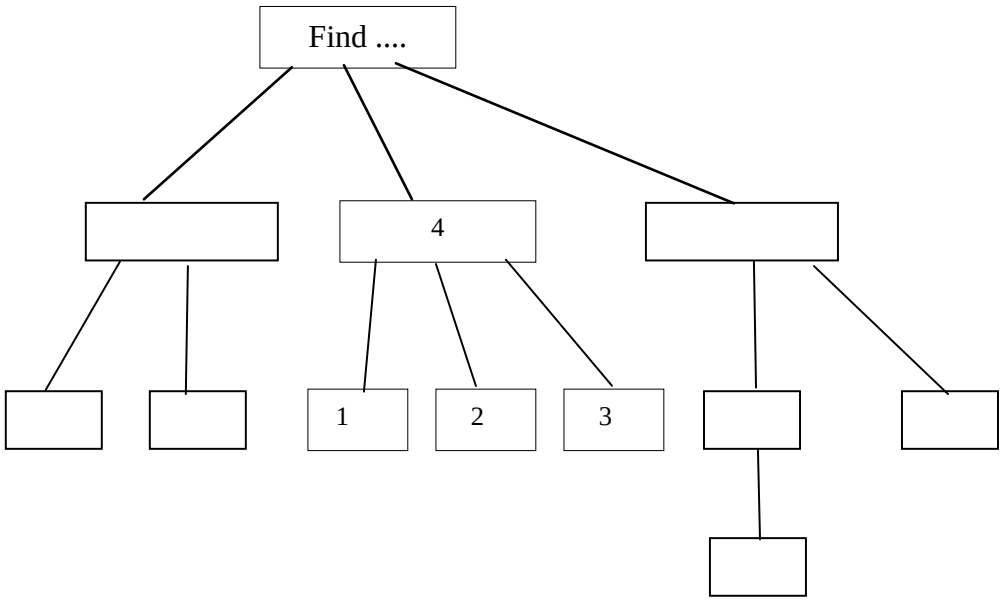


Figure 1.3.1. An example of a partial action diagram

Part of the resulting hypernet is:

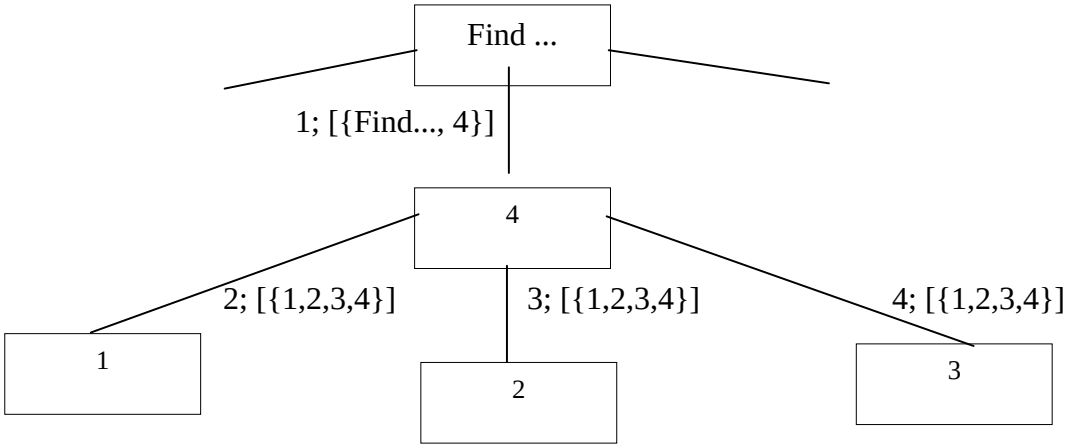


Figure 1.3.2 A partial hypernet for figure 1.3.1

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 that 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 is shown in figure 1.3.3.

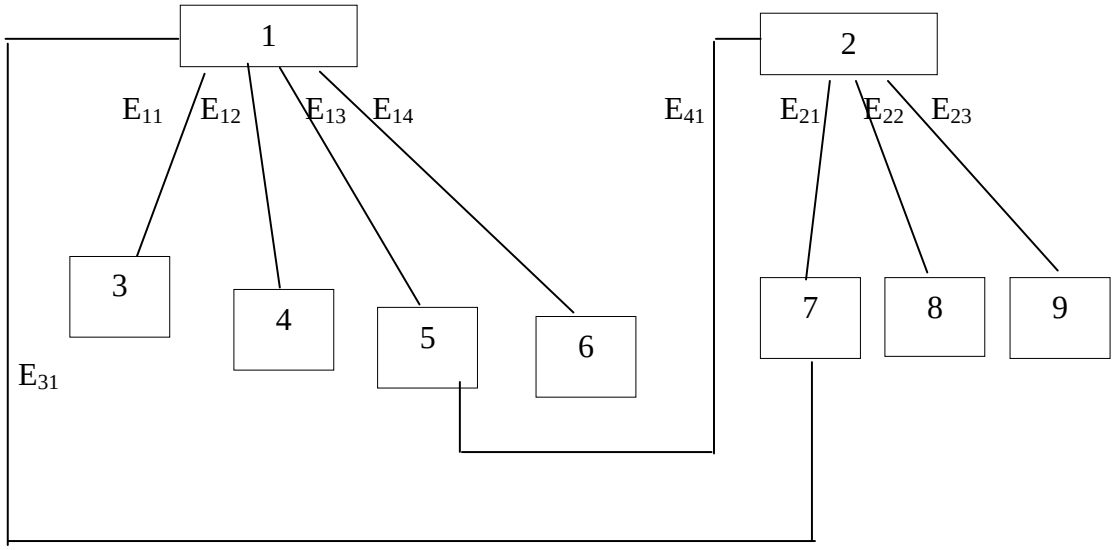


Figure 1.3.3. A hypernet from the partial action diagram on page 141 of [GVS99]

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}, and 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.

1.4 Relation Nets and Hypernets

Consider a finite set $A = \{A_1, A_2, \ldots, A_n\}$ and a family of relations $R = \{R_i \mid i \in I, I \text{ a finite index set}\}$ over A where all R_i have an arity of at least 2, i.e. $\text{card}(R_i) \geq 2$, written $|R_i| \geq 2$. 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 sets. 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 $T = \{T_x\}$.

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 $T_k = \langle a_0, \ldots, a_j \rangle$, where j is the arity of the relation R_i from which T_k arises, we draw an arrow from the a_0 vertex to the a_j vertex. Now label each such arrow <a, b> with a ***label set*** $\lambda(\langle a, b \rangle)$ where $\lambda(\langle a, b \rangle)$ is defined by $\lambda(\langle a, b \rangle) = \{T_k \in T \mid T_k = \langle a, \ldots, b \rangle\}$. There is no arrow from $a \in A$ to $b \in A$ iff $\lambda(\langle a, b \rangle) = \varnothing$.

Inspiration often arises from seeing a familiar situation from a new point of view. 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.

By a ***hypernet*** <A, E> we mean a finite set A and a finite set E of subsets of A. The members of A are called ***vertices*** of <A, E> and the members of E are called ***edges*** of <A, E>. Two edges E_i and E_j of <A,. E> are distinct if $i \neq j$, even if E_i and E_j are the same subsets of A.

More formally**,** by a ***hypernet*** <A, E> we mean a structure in which $A = \{A_1, A_2, \ldots, A_n\}$ 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 $i \neq j$, even though E_i and E_j may be the same set.

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 $E_i \cap E_j \neq \varnothing$, 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 $E_i \cap E_j \neq \varnothing$ and $E_k \cap E_j \neq \varnothing$. 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.

We see that if $E_i, E_j \in E$ and E_i is a singleton edge $E_i = \{A_r\}$, and if $A_r \in E_j$, then E_i and E_j are potentially edge adjacent by A_r.

N**ote:** 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.

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 $E_k \cap E_\ell \neq \varnothing$, 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 could be regarded as a subclass of the class of hypernets.

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 λ: A x 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 $\lambda(\{A_i, A_j\}) = \varnothing$ 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 descriptions given above are illustrated in figure 1.4.1:

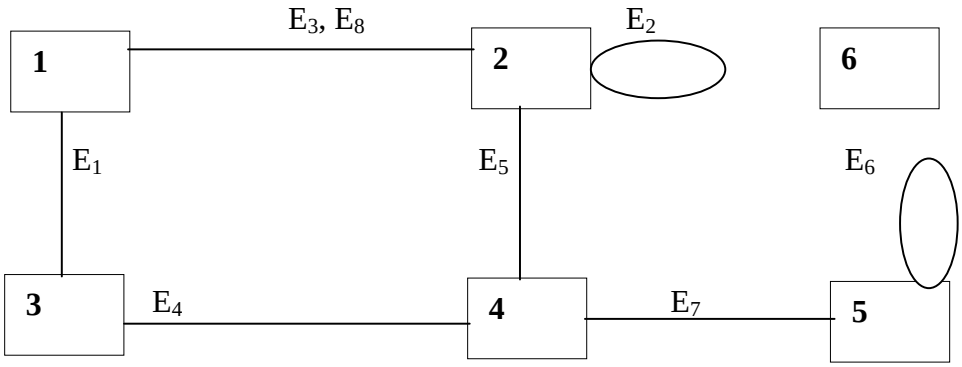


Figure 1.4.1. An example of a hypernet <A, E>

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.

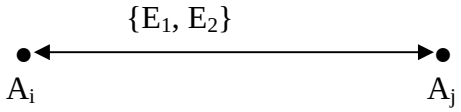
Vertex adjacency: vertices 1 and 2 by edge E_3 and by edge E_8 for example.
Edge adjacency: edge E_3 = {1, 2 }and edge E_5 = { 2, 3, 4} by vertex 2 for example.
Singleton (loop) edge: edge E_2 = {2} and edge E_6 = { 5 }for example.
Notice that a singleton edge $\{A_k\}$, $A_k \in A$, can only have label $\{A_k\}$. Singleton edges can be thought of as representing predicates. Thus, for example, $\{A_k\}$ could represent "A_k is red".

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 $\lambda(\{A_i, A_j\}) \neq \varnothing$. (Notice that we may have $A_i = A_j$, but singleton edges are not usually included.)

By an ***isolate*** of a hypernet <A, E> we mean an $A_i \in A$ for which A_i is not adjacent 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>.

By a ***walk*** in a hypernet <A, E> we mean an alternating sequence of vertices and arcs, A_1, $\{A_1, A_2\}$, A_2, $\{A_2, A_3\}$, A_3, ... , A_q, $\{A_q, A_{q+1}\}$, A_{q+1}, written $A_1 — A_{q+1}$, where for each k = 1, ..., q, A_k and A_{k+1} are vertex adjacent by arc $\{A_k, A_{k+1}\}$ in <A, E> and every arc $\{A_k, A_{k+1}\}$ is associated with only one edge $E_k \in \lambda(\{A_k, A_{k+1}\}) \subseteq E$. (Note that neither the $\{A_k, A_{k+1}\}$ nor the E_k need to be unique.) 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 $\{A_k, A_{k+1}\}$ are distinct arcs and thus associated with distinct edges E_k, k = 1, 2, …, q, then $A_1 — A_{q+1}$ is called a ***path*** in <A, E>. If $A_1 = A_{q+1}$ for a path $A_1 — A_{q+1}$ in <A, E>, then we call $A_1 — A_{q+1}$ a ***circuit.*** Thus, a circuit is a closed path.

Note that if we have, for example,



then A_i, E_1, A_j, E_2, A_i is **not** a circuit because, while E_1 and E_2 are distinct, the arc between A_i and A_j is used twice so this sequence is a closed walk, not a closed path.

We go back to our example in figure 1.4.1 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 also 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.

The kind of structure met here is a hypernet. We should note that, disregarding singleton edges, the diagram (fig 1.4.1) is that of a hypernet with circuits, but that reading such a hypernet from top-to-bottom imposes a “downward” direction on 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 illustrate some of the definitions that we have met, we consider the following example. (It deals with part of a module system 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. This sub-hypernet is the "context-hypernet" of COS211 in the whole module system: It represents all the inter-module 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 for 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 1.4.2. 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.

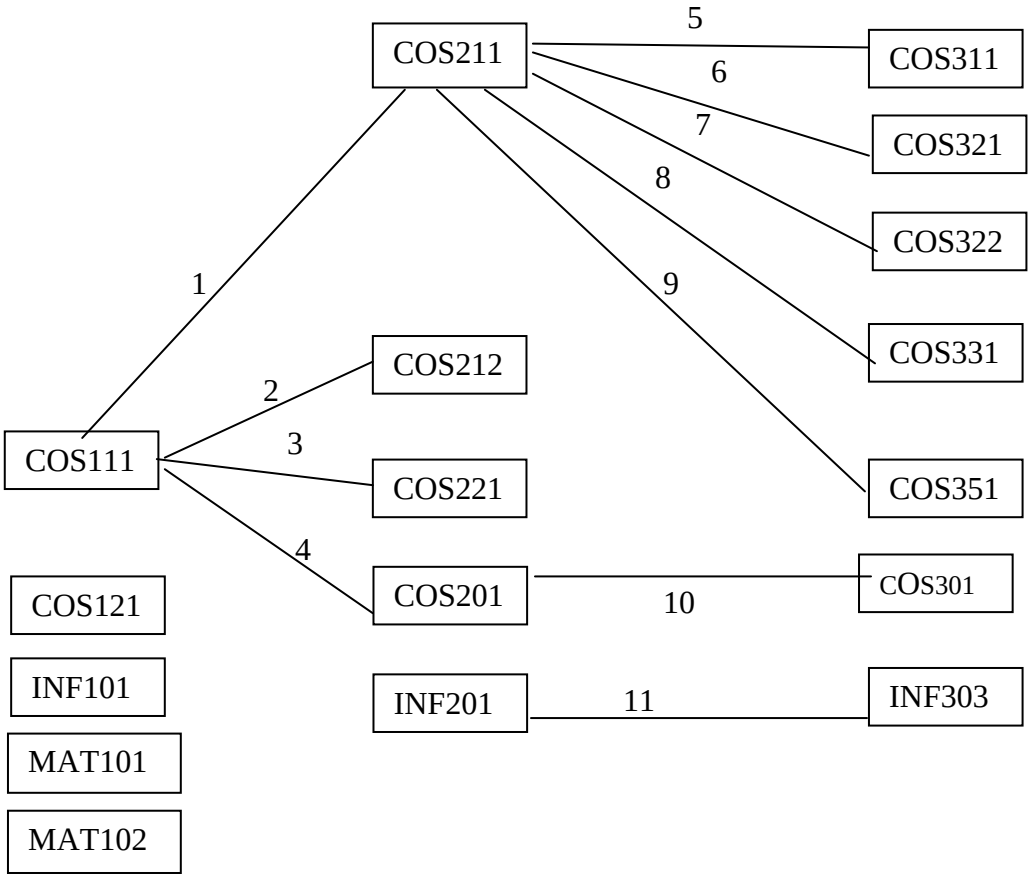


Figure 1.4.2. A diagram for part of a module system

By adding an entrance level module on the left we can easily arrange that this structure be a Knowledge Hypernet - see later - when read from left to right.

Consider now a Prerequisite Chain for Computer Science Major Courses as published by the University of Waterloo (4 March 2003 – http://www.cs.uwaterloo.ca/undergrad/courses/charts/majorPrereq.shtml) and as given in figure 1.4.3.

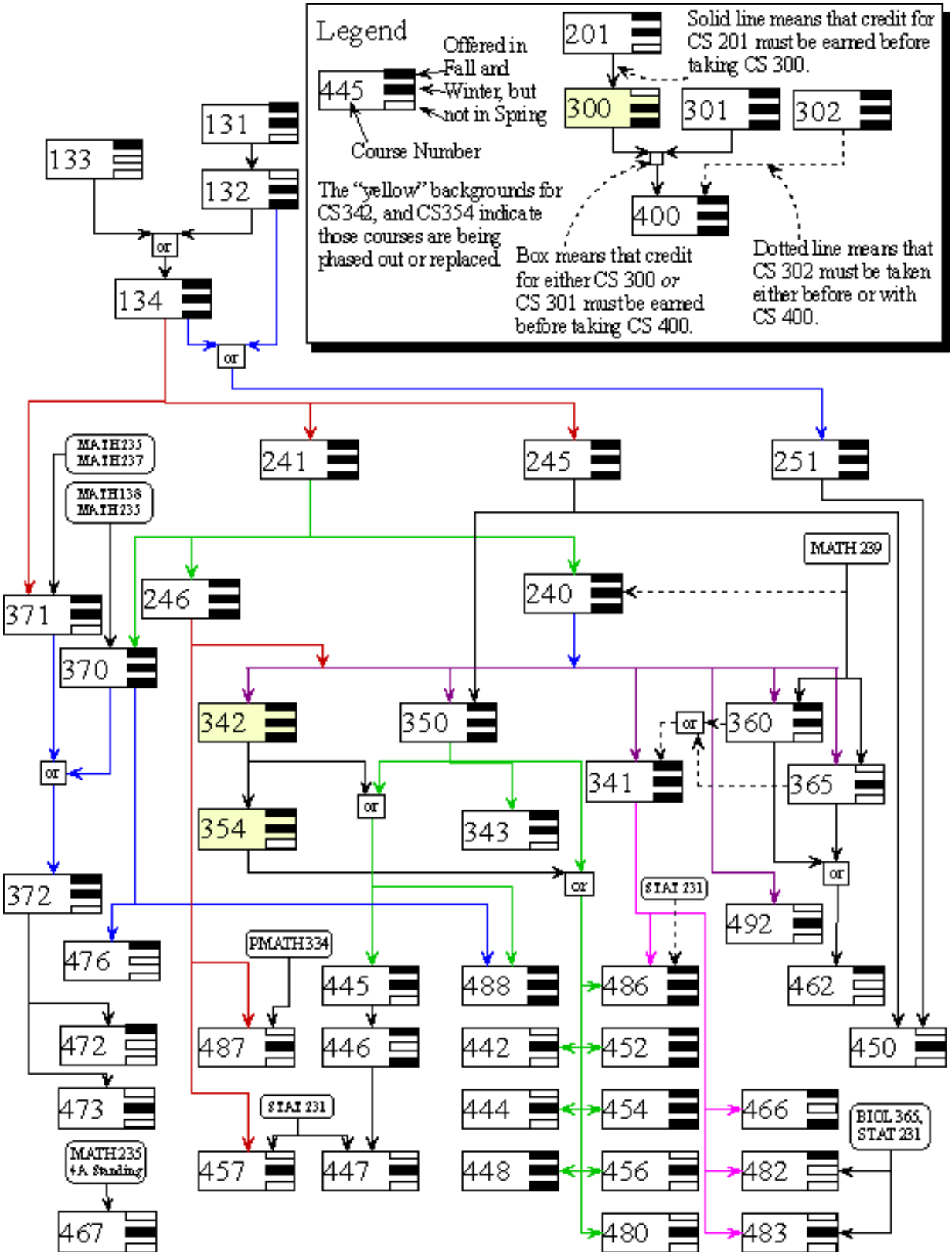


Figure 1.4.3 Prerequisite Chain for Computer Science Major Courses (University of Waterloo)

We want to present the hypernet representation of the information in figure 1.4.3. In doing so we will ignore the periods during which courses are offered, and the fact that CS 342 and CS 354 are being phased out or replaced. An OR box will be replaced by alternative paths. The "CS" part of the vertex labels is left out, as in figure 1.4.3.

The course codes are as follows:

- 131, 132, 133, 134.
- 240, 241, 245, 246, 251.
- 341, 342, 343, 350, 354, 360, 365, 370, 372.
- 442, 444, 445, 446, 447, 448, 450, 452, 454.
- 456, 457, 462, 466, 467, 472, 473, 476,
- 480, 482, 483, 486, 487, 488, 492.
- Math 138.
- Math 235, Math 235 and A standing, Math 237, Math 239.
- Pmath 334.
- Stat 231.
- Biol 365.
- We add a dummy vertex.

The condition sets are as follows, where parallel courses are underlined.

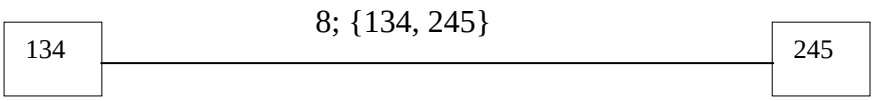
1. {131, 132}
2. {133, 134}
3. {132, 134}
4. {Math 235, 134, Math 237, 371}
5. {134, 251}
6. {132, 251}
7. {134, 241}
8. {134, 245}
9. {Math 138, 241, Math 235, 370}
10. {241, 246}
11. {241, Math 239, 240}
12. {240, 246, 342}
13. {240, 246, 492}
14. {Math 239, 240, 246, 360}
15. {Math 239, 240, 246, 365}
16. {245, 240, 246, 350}
17. {245, 251, 450}
18. {Math 239, 240, 246, 360, 341}
19. {Math 239, 240, 246, 365, 341}
20. {PMath 334, 246, 487}
21. {Stat 231, 246, 457}
22. {371, 372}
23. {370, 372}
24. {370, 476}
25. {370, 350, 488}
26. {370, 342, 488}
27. {342, 354}
28. {342, 445}
29. {350, 445}
30. {343, 350}
31. {354, 341, Stat 231, 486}
32. {354, 442}
33. {254, 452}
34. {354, 444}
35. {354, 454}
36. {354, 448}
37. {354, 456}
38. {354, 480}
39. {350, 341, Stat 231, 486}
40. {350, 442}
41. {350, 452}
42. {350, 444}
43. {350, 454}
44. {350, 448}
45. {350, 456}
46. {350, 480}
47. {360, 462}
48. {365, 462}
49. {341, 466}
50. {372, 472}
51. {372, 473}
52. {445, 446}
53. {446, Stat 231, 447}
54. {Math 235, A standing, 467}
55. {341, Biol 365, Stat 231, 482}
56. {341, Biol 365, Stat 231, 483}
57. {245, 251, 450}
58. {341, Stat 231, 486}

The conditions involving the dummy vertex, which represents the background courses required to enter this curriculum, are as follows.
{dummy, 131}, {dummy, 133}, {dummy, Math 235}, {dummy, Math 237},
{dummy, Math 138}, {dummy, 239}, {dummy, Stat 231}, {dummy, PMath 334},
{dummy, Math 235 and A standing}, {dummy, Biol 365}.

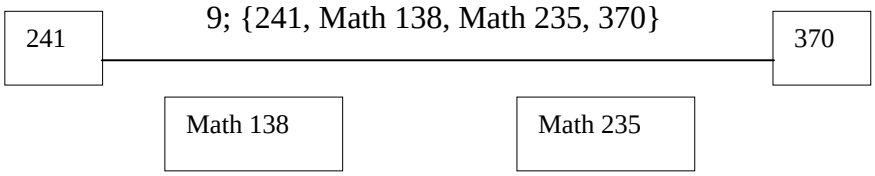
We see at once that the resulting hypernet is quite trival. Since the structure is a hypernet, we can apply hypernet search techniques and vulnerability analysis to it. In the latter case the results will be non-trivial. It is obvious at once, for example, that 350 is a key course. In a larger and more complex curriculum system vulnerability analysis may yield some embarrassing surprises. The more complex the curriculum the more apt it is to present and analyse it in hypernet form, essentially because a hypernet is a fairly simple but very rich mathematical structure with many relevant characteristics, and is a genuine and "pure" ***structural*** model.

As example of the sub-hypernet for an edge consider the following:

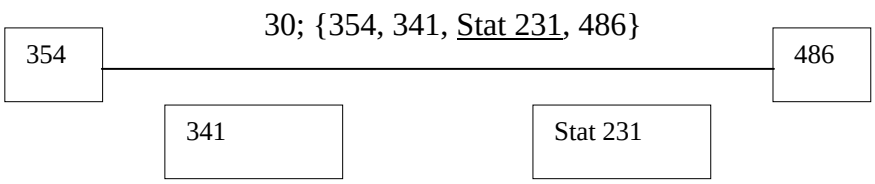
8.



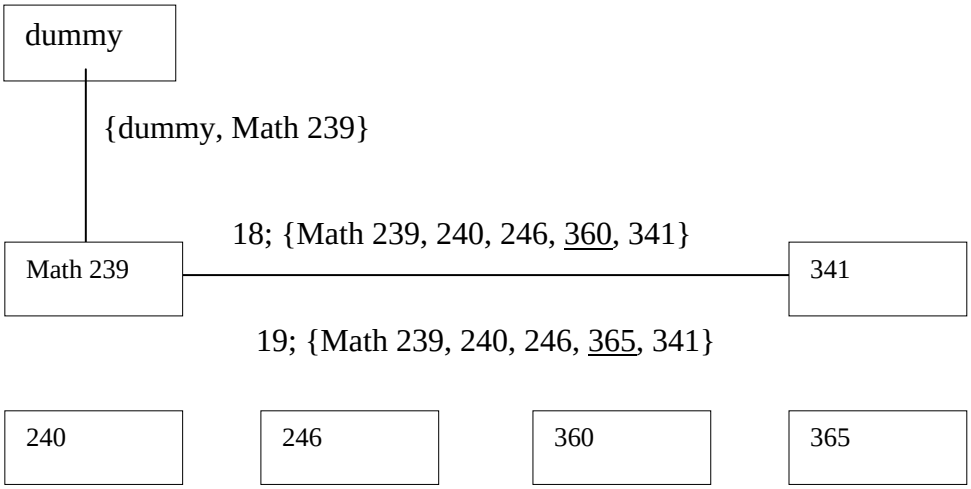
9.



30.



18, 19.



The edge labels are given in full here, but one could apply the obvious simplification of dropping the end vertices from the edge labels without losing information. Thus we would have, for example, 30; {341, Stat 231}. Any of the prerequisite entries in an edge can of course be chosen for an end vertex of that edge in a diagram of the hypernet.

Finally, notice that a host of conditions that involve parallels would make figure 1.4.3 rather more difficult to read, but would make little difference to the complexity of the equivalent hypernet and analysis of it. With appropriate choices of prerequisite as end vertex for each edge, and perhaps addition of some extra edges that repeat some of the information already there, we can arrange that this is a special kind of hypernet, called a Knowledge Hypernet - see later.

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

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 $\rightarrow$ 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 $E_i = \{A_1, A_2, \ldots, A_\ell, \ldots, A_{n(i)}\} \subseteq A$, $A_r \in E_i$, $A_s \in E_i$, (A_r, E_i, A_s) is mapped to precisely 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, $B_k = I(A_\ell)$ for some one $A_\ell \in E_i$, ℓ = 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, i.e. the set of entries, 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>.

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.

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 $\rightarrow$ A that maps <A, T> to a hypernet <A, E> and is defined as follows:
For every tuple $T_i = <A_1, A_2, \ldots, A_\ell, \ldots, A_{n(i)}> \in T$ in <A, T> the mapping M produces a set $E_i = \{M(A_1), \ldots, M(A_\ell), \ldots, M(A_{n(i)})\} \in E$ with $|E_i| \leq |T_i|$, E_i being 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 1.4.1: 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, for example, all the possible permutations of tuples in a CRKS - see [GVS99] and later - for examination and choice of "appropriate" ones.

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 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 all the vertex adjacencies.

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 relation nets, such as CRKS's, from <A, T>. This leads to the following descriptions.

By a ***sub-hypernet*** of a hypernet <A, E> we mean a hypernet <B, U> with $B \subseteq A$, $U \subseteq E$. 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>. The ***maximum sub-hypernet*** <B, $E\uparrow B$>, of a hypernet <A, E>, that is induced by $B \subseteq A$, is such that $E_i \in E$ belongs to $E\uparrow B$ 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, $E \uparrow (A - B)$ > where $B \subseteq A$. Then <X, $\angle$ > is a distributive lattice under $\cup$ and $\cap$ of hypernets, with null element <$\varnothing$, $\varnothing$> 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 $\cap$.

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>.

We now 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.

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: $A \rightarrow B$ and h: $S \rightarrow T$ which are such that tuple $T_i = <A_1, ...A_r, ..., A_n>$, where each entry is an entry of a member of A, belongs to S iff tuple $h(T_i) = <B_1, ..., B_s, ..., B_m>$, 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. 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 1.4.1

(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 $|E_1| = |U|$.

(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})}∪ {g(A_{1k}) ∈ A_2 | A_{1k} ∈ A_1 and A_{1k} ∈ (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.

(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>.

(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.

(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>.♦

Comment: The two hypernets are stored as two edge tables. It is clear that we would start with step 2 by choosing a vertex adjacency in <A_1,E_1> and reading the edge table of <A_2, E_2> to find a preliminary "match" for which we can define g and h. If a match is found we move to step 3 in which a computer examines each of the possible domain extension vertex adjacencies, i.e. edges of <A_1, E_1> that are incident with one of the vertices of that matched vertex adjacency, in <A_1, E_1>, from step 2. Notice that since <A_1, E_1> and <A_2, E_2> are given, the vertex adjacencies of both are fixed. (Normally one would, in storing an edge, let the first and last members of that edge set list in the edge table be the two vertices adjacent by that edge.) The rest of the implementation works in essentially the same way as step 3.

At this stage we should note that, as mentioned above, the vertex adjacencies of a hypernet are fixed, so there is no essential difference between the implementation of hypernet constructional schemes and the equivalent ones for relation nets. This is why we have not, in the text, even presented some of those hypernet constructional schemes that are essentially the same as their relation net equivalents.

In applying constructional scheme 1.4.1 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}) be 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:

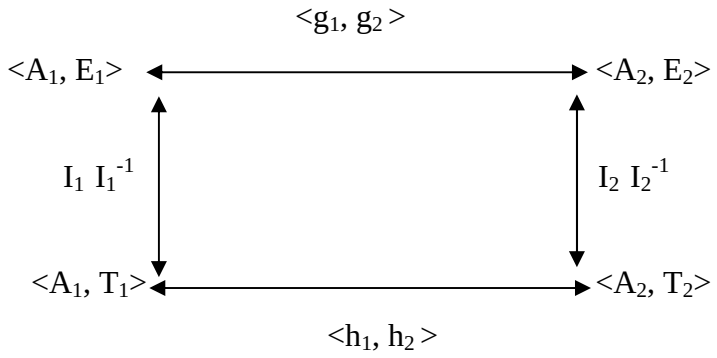


Figure 1.4.4. Isomorphisms and interpretations

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. All these mappings are one-to-one-correspondences, so their inverses are well-defined simple reversals.

Theorem 1.4.2: 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 necessarily 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: $A_1 \rightarrow A_2$ and h_2: $T_1 \rightarrow T_2$ 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 1.4.3: 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 1.4.2 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 1.4.2. 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 necessarily 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 $(I_2^{-1}(h_1(A_{11})), E_{2j}, I_2^{-1}(h_1(A_{12}))$. 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 1.4.2 can help in finding relation net isomorphisms.

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. Consider the following diagram:

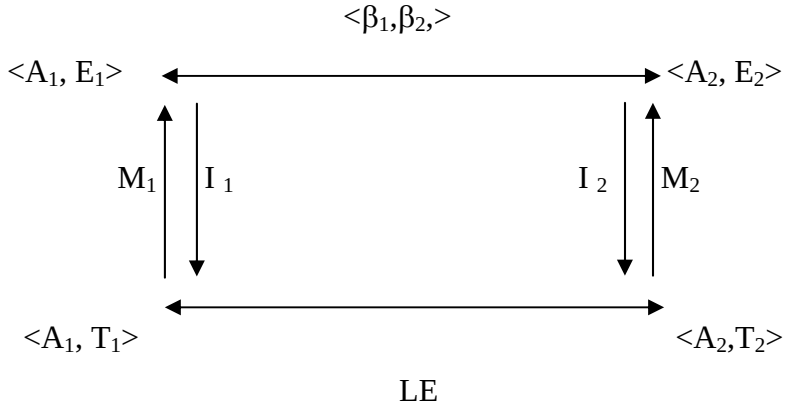


Figure 1.4.5 Language Equivalence

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 [GVS99] - 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. Preservation of vertex adjacencies can be a severe constraint.

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 $B \subseteq A$.

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

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

The meet of <B, F> and <C, G> is written <B, F> ∩ <C, G>, and their join is written <B, F> ∪ <C, G>. In (4) the only way in which F and G can share edges is that those shared edges are subsets of B ∩ C. Thus we have the following:

Theorem 1.4.4: If $E_i \in E$, and hypernet <A, E> = <B ∩ C, F ∩ 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.

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 $E_j \in E\} \cup \{A_r\}$.

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 pair-wise distinct in edge subsequences over E. By a ***sub-walk-family*** of $f(A_r — A_s)$, we mean a walk-family f $(A_m — A_n)$, $r \le m < n \le s$, 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.

We now define the following three terms for hypernets.

(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 on 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.) We write $(A_r - A_j - A_s)$.

(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>. Every vertex is taken to be reachable from itself by a path of length zero.

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

(4) The meanings of $\Gamma(B)$ and $\Re(B)$ for $B \subseteq A$ are obvious.

Next we tackle the notion of a cascade. Consider a hypernet <A, E> and a collection of subnets of <A, E>: $<B_0, D_0> \angle <B_1, D_1> \angle \ldots \angle <B_k, D_k> \angle \ldots \angle <B_n, D_n>$. Thus, for each k = 0, 1, …, n - 1, we have $<B_k, D_k> \angle <B_{k+1}, D_{k+1}>$, i.e. $B_k \subseteq B_k$ and $D_k \subseteq D_{k+1}$. Such a collection of subnets of <A, E> is called **a *fast access cascade*** from B_0 in <A, E> if, and only if (iff), we have

(1) $B_0 \subseteq A$ and $D_0 = \varnothing$, and
(2) $D_1 \subseteq E$ is chosen in such a way that for each vertex adjacency (A_r, E_j, A_s), $E_j \in E$, we have that E_j belongs to D_1 iff A_r or A_s belongs to B_0 .
(3) B_1 is B_0 together with the union of all the E_j in D_1 and in general for k = 2, 3, …
(4) $D_k \subseteq E$ is chosen in such a way that for each vertex adjacency (A_i, E_k, A_j), $E_k \in E$, we have that E_k belongs to D_k iff A_i or A_j belongs to B_{k-1}, or E_k belongs to D_{k-1}, so $D_{k-1} \subseteq D_k$, and
(5) B_k is B_{k-1} together with the union of all the E_k in 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, T> iff at each step k = 1, 2, ... , an edge E_c belongs to D_k, iff it belongs to D_{k-1} or all, but possibly one, of the members of E_c belong to B_{k-1}, and that one is either A_r or A_s in each vertex adjacency (A_r, E_c, A_s) used in choosing the E_c at each step k = 1, 2, … .

More formally, we have the following: The nested sequence $\{<B_k, D_k> \mid k \geq 0\}$ of sub-hypernets of a hypernet <A, E> is called a ***fast access cascade*** from B_0 iff

(1) $B_0 \subseteq A$ and $D_0 = \varnothing$, 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) $B_1 = B_0 \cup$ (the union of all the E_j that belongs to D_1), and in general for k = 2, 3, …
(4) $D_k \subseteq E$ 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_k iff it belongs to D_{k-1} or A_r or A_s belongs to B_{k-1}, so $D_{k-1} \subseteq D_k$, and
(5) $B_k = B_{k-1} \cup$ (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 it belongs to D_{k-1} or 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 adjacency (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 - see [GVS99].

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.

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 union of all the members of E $[A_k]$, i.e. the set of vertices $\{A_j \in A \mid A_j \in E_i$ and $E_i \in E\ [A_k]\}$. $<A, E>[A_k]$ is a hypernet.

We return to our example in figure 1.4.1 and illustrate the later notions met in this section:

- sets E (A_k) and E [A_k]: E(3) = {E_5} and E[3] = {E_1, E_6, E_5, E_4}.
- adjacency function: Γ(4) = {2, 3, 4, 5}, and Γ(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: ℜ (2) = A - {6}, ℜ (2) = ℜ (1) = ℜ (3) = ℜ (4) = ℜ (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 ∈ 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.

2: Some Characteristics of Hypernets

2.1 Connectedness, Components, and Vertex Bases

While this chapter applies to an arbitrary hypernet <A, E>, we are concerned mainly with the case in which <A, E> is a KH.

Definition 2.1.1: 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 2.1.1: 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. ♦

We need to know even more about paths. What characteristics of a hypernet are dependent upon what sorts of paths? Let's begin to see. Much of what we find is directly related to similar situations in standard graph theory and relation net theory - see [GVS99], Part III.

Definition 2.1.2: 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 2.1.2: 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. $B_0 \cap B_1 = \varnothing$. ♦

Proof: Suppose that $b \in B_0 \cap B_1$, and let $a \in B_0$ and $c \in B_1$. 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 $<B_0, U_0> \cup <B_1, U_1>$ lies in a single component of <A, E>. The theorem follows. ♦

Theorem 2.1.3: 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 distinct components of <A, E>. Then, as in the proof of theorem 2.1.2 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 2.1.4: 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 2.1.4 that:

Corollary 2.1.1: Reachability in a hypernet <A, E> partitions A into equivalence classes that are precisely the vertex sets of the components of <A, E>. ♦

We will see that in a sub-hypernet of a KH <A, E>, for example, it is vital to know what subsets of A are such that every vertex can be reached from at least one member of that subset in searching in <A, E>. The following theorems begin to give us a hold on this problem.

Definition 2.1.3: A ***vertex basis*** for a hypernet <A, E> is a set $V \subseteq A$ such that every $a \in A$ is reachable from at least one $v \in V$, and V is minimal in the sense that no proper subset of V has this property. ♦

Theorem 2.1.5: Every $a \in A$ 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 2.1.6: $V \subseteq A$ of a hypernet <A, E> is a vertex basis of <A, E> iff
(1) every $a \in A$ is reachable from at least one $v \in V$, i.e. $\Re(V) = A$, and
(2) no $v \in V \subseteq A$ is reachable from any $w \neq v$, $w \in V$, 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, $v \in V$ and that w and v are mutually reachable in <A, E>. Then every $a \in A$ 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 2.1.2: 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 2.1.3: Every hypernet <A, E> has at least one vertex basis. ♦

Proof: A certainly fulfils condition (1) of theorem 2.1.6, so we can find at least one $V \subseteq A$ that fulfils condition (2) as well. ♦

Theorem 2.1.7: If $V \subseteq A$ is a vertex basis of a hypernet <A, E> then there is precisely one $v \in V$ 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 $a \in A$. Suppose that v, $w \in V$ 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 2.1.8: If <B, U> $\angle$ <A, E> then every vertex basis of <A, E> contains a vertex basis of <B, U>. ♦

Proof: Let $V \subseteq A$ be any vertex basis of the hypernet <A, E>. Then every $a \in A$ is reachable from some one vertex $v \in V$. Since <B, U> $\angle$ <A, E> it is clear that every vertex $b \in B \subseteq A$ 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>. ♦

Recall: Let $a \in A$ of a hypernet <A, E>. By $\mathfrak{R}(a)$ we mean the collection of all $b \in A$ that are reachable from a by at least one path a — b in <A, E>. We always let $a \in \mathfrak{R}(a)$.

Since connectedness of a (sub-) hypernet is a rather desirable property, we should note the following:

Theorem 2.1.9: Given $a \in A$ of a hypernet <A, E>, the hypernet < $\mathfrak{R}(a)$, $E^{\uparrow}(\mathfrak{R}(a)$ >, i.e. the maximum sub-hypernet of <A, E> that is induced by $\mathfrak{R}(a)$, is connected. ♦

Proof: Every $b \in \mathfrak{R}(a)$ is reachable from a, and every $E_1 \in E^{\uparrow}(\mathfrak{R}(a))$ is a subset of $\mathfrak{R}(a)$. The theorem follows. ♦

It is easy to show that the set of all primaries of a KH <A, E> constitutes a unique vertex basis of <A, E>, and the set of all goals a unique vertex contra-basis, when we consider derivation paths in <A, E>.

We close this 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> ≤ d(a) in <A, U_2> ≤ d(a) in <A, E>.
(2) For all a, $b \in A$, if b is reachable from a in <A, U_1> 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>.

2.2 Vulnerability

Consider an arbitrary hypernet <A, E>. Each of the components of <A, E> should be "strongly connected", and it follows that analysis of the vulnerability of these components with respect to deletion of edges, and of vertices, is very relevant in order to see which edges and vertices are critical to the connectedness of <A, E>. This is particularly important for KHs.

We begin by considering the "critical edges" of <A, E>. Which edges are vital to the preservation of connectedness? Which edges can be deleted without affecting the connectedness of the components of <A, E>?

Definition 2.2.1: Let a, b ∈ 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 2.2.2: Let a, b ∈ A of a hypernet <A, E>, a ≠ 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.

The following theorems tell us something about how destructive deletion from <A, E> can be, and something about what kind of destruction arises. We need to be very careful about what we "trim" from <A, E> before we start to search in it!

Theorem 2.2.1: Let a, b ∈ A of a hypernet <A, E>, a ≠ 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 the (one) vertex adjacency by E_i is a subsequence, of length 1, of a — b. ♦

Proof: If (a - E_i - b) then a and b are joined in <A, E> and every path a – b is such that there is a vertex adjacency by E_i in a – b. If a and b are joined in <A, E> and every path a – b in <A, E> is such the vertex adjacency by E_i is in a – b then E_i is between a and b, i.e. (a – E_i – b), in <A, E>.♦

From the theorem we get

Corollary 2.2.1: If a and b of the theorem are adjacent vertices then $\lambda(\{a, b\}) = \{E_i\}$ ♦

and

Corollary 2.2.2: 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 2.2.3: 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 2.2.2: 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 2.2.2, there follows

Corollary 2.2.3: Every $E_i \in E$ of a disconnected hypernet <A, E> is a (0,0)-edge, i.e. E_i is neutral. ♦

Theorem 2.2.3: 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 2.1.1 <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. ♦

Next we characterize especially destructive edges.

Definition 2.2.4: 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). ♦

It follows at once that $E_i \in E$ is a bridge in <A, E>, where $|A| \geq 2$, iff there exists a vertex adjacency {c, d}, in <A, E> for which λ ({c, d}) = {E_i}.

Theorem 2.2.4: $E_i \in E$ 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 2.2.5: 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 2.2.4 that we have

Theorem 2.2.6: Every strengthening edge, i.e. (1, 0)-edge, in a connected hypernet <A, E> is a bridge in <A, E>.♦

Theorem 2.2.7: 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 2.2.2. ♦

If <A, Q> $\angle$ <A, E>, what is the comparative vulnerability of the two hypernets?

Theorem 2.2.8: 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 (1). 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 2.2.4: 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: <A, Q> is in C_1, and E_i is (1, 1) in <A, Q>, so there must be "a way round" E_i in <A, Q>, and since E_i cannot be (0, 1) in any hypernet, <A, E> must be in C_1 so E_i is a (1, 1)-edge in <A, E> too. Simply put, if <A, Q> is connected then <A, E>, with $Q \subseteq E$, must certainly be connected as well. ♦

Corollary 2.2.5: 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 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. ♦

2.3 Edge Bases

Reachability in <A, E> is critical for searches in it. If we "trim" some edges from <A, E> before a search, we must do it in such a way that critical reachability is preserved. This leads us to the notion of edge bases.

Definition 2.3.1: Let <A, E> be any hypernet with $B \subseteq E$. B is called an ***edge basis*** of <A, E> iff for all a, b ∈ 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.♦

We must now try to characterize an edge basis. Which edges belong to an edge basis, and how do we look for them in <A, E>?

Theorem 2.3.1: $E_i \in E$ of a hypernet <A, E> is between a and b in <A, E>, a, b ∈ 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 ∈ A such that $a \in \Re(b)$ and every path a — b goes via E_i, so (a - E_i - b). ♦

Theorem 2.3.2: If for a, b ∈ A of a hypernet <A, E> there is a unique path a — b in <A, E> then $\{E_i \in E \mid a \text{ — } b \text{ goes via } E_i\}$ 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 2.3.1 each such E_i belongs to every edge basis of <A, E>.♦

Theorem 2.3.3: 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 ∈ 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 ∈ A with (a - E_j - b). Then we can preserve the reachability of a from b without E_j, so we do not need E_j in B, i.e. a proper subset (B - $\{E_j\}$) ⊆ B will preserve reachability, so B is not an edge basis . ♦

Clearly edge bases, reachability and connectedness are related. How?

Theorem 2.3.4: $B \subseteq E$ 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 ∈ 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>. ♦

There must be a relationship between connectedness and (closed) spanning walks. What is it?

Theorem 2.3.5: 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>. ♦

Have we now got enough information to find an edge basis in a hypernet? The answer is yes.

Theorem 2.3.6: To find an edge basis for a hypernet <A, E> we may use the following:

Constructional scheme 2.3.1: 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 $\lambda(\{a, b\})$, for each of which we have (a, E_i, b), so $\{a, b\} \subseteq E_i$.

Stage 1

(1) Define a bipartite graph with vertex sets V_1 and V_2 where $V_1 = \{\{a, b\} \in \wp(A) \mid a \text{ and } b \text{ are adjacent vertices in } <A, E>\} = D$ 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 a unoriented edge. These are all the vertices and edges of our bipartite graph. Each vertex adjacency in $V_1 = D$ is adjacent with at least one $E_i \in V_2 = E$, and each $E_i \in V_2$ is adjacent with precisely one member of V_1, in our bipartite graph.

(2) For every $s \in V_1$, choose any one of the vertices $t \in V_2$ that is adjacent with s in our graph. Mark that vertex $r \in V_2$.

(3) Let $L^1 \subseteq V_2$ be the set of marked vertices after step (2). L^1 contains at least one edge basis of <A, E>, and $|L^1| \leq |E|$.

End of stage 1.

Proof of stage 1: It is clear that L^1 contains at least one edge basis of <A, E> at this stage because L^1 "covers" every vertex adjacency in <A, E>. If a is reachable with b, $a, b \in A$ and $a \neq b$, then it is clearly reachable with b in $<A, L^1>$. That $|L^1| \leq |E|$ follows from the fact that every $E_i \in L^1$ "covers" one "new" vertex adjacency. Further, L^1 is a minimal set of edges that "covers" every vertex adjacency in <A, E>, because each $E_i \in L^1 \subseteq E$ covers a vertex adjacency by E_i. ♦

Stage 2

(4) Examine L^1 as follows. Find an $E_i \in L^1$ 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 $L^1 - \{E_i\}$ only. If there is no such $E_i \in L^1$, then $\{E_i \in E \mid E_i \in L^1\}$ is an edge basis of <A, E>. If there is such an E_i, set $L^2 = L^1 - \{E_i\}$. Repeat the test on the members of L^2. Either $\{E_i \in E \mid E_i \in L^2\}$ is an edge basis for <A, E> or we define $L^3 = L^2 - \{E_i\}$ for some $E_i \in L^2$.

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 $L^n \subseteq L^1$ is an edge basis for <A, E> we must prove that $E_i \in L^1$

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^1 - {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^1 - {E_i}, then removal of E_i from L^1 means that a is not reachable with b in <A, L^1 - {E_i}>, so L^1 - {E_i} does not contain an edge basis of <A, E>. But L^1 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^1. Conversely, if for all a, b ∈ A such that there is at least one path a — b via $E_i \in L^1$ in <A, E> there is a path a — b in <A, L^1 - E_i}> then L^1 - {E_i} contains at least one edge basis of <A, E>, and so E_i does not necessarily belong to an edge basis $B \subseteq L^1$. Thus we have the correct criterion for rejecting an $E_i \in L^1$. ♦

Comment: Step 1 constructs a bipartite graph that can be generated by computer. The following steps use that graph and can easily be implemented by reading the bipartite graph, which will be stored as a set of edges from which we can read and mark edges as we use them in the CS. Step 4 will require that one constructs the path tree for <A, E>. If <A, E> is a KH then one can use a minor modification of CS 1.3.7. If not then we may use a relatively simple search in <A, E>, roughly as follows: Given a, b ∈ A, run a fast access cascade, in <A, E>, from B_0 = {a}. If we do not find b in some B_n then there are no a — b paths in <A, E>; b is not reachable from a, so drop b and look at a — c paths, c ∈ A and c ≠ b. If b ∈ B_n for some n, with <B_{n+1}, E_{n+1}> = <B_n, E_n>, then we have all a — b paths inside <B_n, E_n>. Now <B_n, E_n> contains all a — b paths and we can use essentially CS 1.3.7 to set up the path tree for <B_n, E_n>. Now delete $E_i \in L^1$ from <B_n, E_n> and see if there are still a — b paths left. If, for **each** $E_i \in L^1$, for this arbitrary a, b ∈ A, a ≠ b, we find that after deleting E_i from <B_n, E_n> there remains at least one a — b path that goes via members $L^1 - \{E_i\}$ only, and this holds for every choice of a and b, then E_i is not a member of any edge basis of <A, E> that is included in L^1 and we start again with $L^2 = L^1 - \{E_i\}$. If not, i.e. if E_i lies on **every** a — b path for at least one choice of a and b, then E_i belongs to at least one edge basis of <A, E> that is included in L^1, and if this is true for every $E_i \in L^1$, then L^1 is an edge basis of <A, E>. Computer implementation of part 4 is thus tedious, but simple in principle. We choose any a and b and set up the path tree displaying all a — b paths. We delete E_i from the path tree. We see if there is still at least one a — b path. If there is, we repeat for each choice of a and b. We reject E_i if there is an alternative path, which must be via members of L^1 - {E_i}, for every choice of a and b. If for even one choice of a and b we find that E_i lies on every a — b path then we keep E_i as a member of an edge basis of <A, E> and move on to test another member of L^1. Repeat until every member of L^1 has been tested for deletion or acceptance, thus moving through L^2, L^3, --- in turn we find an edge basis for <A, E> in L^1.

To close this section we return to theorem 2.3.5. Related to a set of edges that preserves reachability, i.e. an edge basis, is a set of edges that preserve connectedness.

Definition 2.3.2: 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 2.3.7: Let <A, E> be a connected hypernet. W is a spanning walk of minimal length in <A, E> iff E_W = { $E_i \in E$ | W goes via E_i} is a minimal set of edges that preserves the connectedness 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 $|E'|$. ♦

2.4 Deletion of Vertices

It is possible that we may want to "trim" vertices from <A, E> before certain searches are carried out. In doing so we must preserve the connectedness and reachability necessary for the search, so we must know, quite precisely, the vulnerability of <A, E> with respect to deletion of each $A_i \in A$. This is not an easy matter!

We open this discussion with a comment in the form of a lemma. Let <A, E> be any hypernet, and let $B \subseteq A$. Consider the following sub-hypernets of <A,E> - see section 1.5 and definition 2.1.1 -

- <B, E↑B>
- <A,E(B)>
- <A,E[B]>
- <A,E>[B]

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

Lemma 2.4.1:
(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. ♦

What we need to do, basically, is to examine the relationship between "joinedness" and vertices in order to begin to see what deletion of a vertex may do to a hypernet <A, E>.

Theorem 2.4.1: 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>.♦

What is a "worst case" vertex deletion?

Definition 2.4.1: A vertex $b \in A$ of a hypernet <A, E> is called a ***cut-vertex*** of <A, E> iff there exist $a, c \in A$ such that (a - b - c) in <A, E>.♦

Theorem 2.4.2: Let <A, E> be a connected hypernet. The following statements are logically equivalent for every $b \in A$:

(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 2.4.1: 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 2.4.2. ♦

The next small hold that we can get on the slippery business of vertex deletion involves trying to find out what deletion of a particular vertex from <A, E> does to its connectedness.

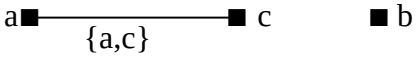
Definition 2.4.2: Vertex b ∈ A of a hypernet <A, E> is called an ***(x, y)- vertex*** of <A, E> iff <A, E> is in C_x and <A - {b}, E↑(A - {b})> is in C_y. b is called an ***strengthening vertex*** iff x > y, a ***neutral vertex*** iff x = y, and a ***weakening vertex*** iff x < y. ♦

Theorem 2.4.3: 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]. ♦

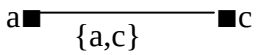
Proof: Follows at once from the fact that deleting the edges E(a) ⊆ E, i.e. the edges in the name of a, from <A, E> to produce <A, E^c(a)> cannot increase the connectedness class of <A, E> as there are no (0,1)-edges in any hypernet. Thus x ≥ y (see theorem 2.2.2). ♦

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

a) <A, E> in C_0:



b)<A{b},E↑(A{b})>in C_1:



Let's try to get as close to definite answers as we can.

Theorem 2.4.4:
(1) If $b \in A$ is an (x, y)-vertex in $\langle A, E^c(b)\rangle$ then it is a (z, y)-vertex in $\langle A, E\rangle$ with $z \geq x$.
(2) If $b \in A$ is a (z, y)-vertex in $\langle A, E\rangle$ then it is an (x, y)-vertex in $\langle A, E^c(b)\rangle$ with $z \geq x$. ♦

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

(1) Starting with $\langle A - \{b\}, E^c[b]\rangle$ we get $\langle A, E^c(b)\rangle$ by adding b and all the edges of $E^c(b) - E[b]$. The result $\langle A, E^c(b)\rangle$ is in C_x. To get $\langle A, E\rangle$ from $\langle A, E^c(b)\rangle$ we must add all the edges of $E - E^c(b)$, i.e. all the edges of E(b), and we get $\langle A, E\rangle$ 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 $z \geq x$.

(2) Starting with $\langle A - \{b\}, E^c[b]\rangle$, which is in C_y, we get $\langle A, E\rangle$ by adding b and all the edges of E[b] and $\langle A, E\rangle$ is in C_z. Now to get $\langle A, E^c(b)\rangle$ from $\langle A, E\rangle$ we must delete all the edges of E(b). Let the connectedness class of $\langle A, E^c(b)\rangle$ be C_x. Then by theorem 2.4.3, $z \geq x$. ♦

Corollary 2.4.2: For a hypernet $\langle A, E\rangle$ with $b \in A$, the particular cases of the theorem are:

a) b is x, y in $\langle A, E^c(b)\rangle \Rightarrow$ b is (z, y) in $\langle A, E\rangle$ with $z \geq x$

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 $\langle A, E\rangle \Rightarrow$ b is x, y in $\langle A, E^c(b)\rangle$ with $z \geq x$.

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

We cannot get much further with the topic of vertex deletion, which is not to say that it is an unimportant topic. The rest of this section lists some points that may be helpful in examining specific cases of vertex deletion.

Theorem 2.4.5: Let $B \subseteq A$ be any non-empty set for a hypernet $\langle A, E\rangle$, and let $B' = A - B$. Further let $E(B) = (\cup E(b) \text{ for } b \in B) \subseteq E$ and $E[B] = (\cup E[b] \text{ for } b \in B) \subseteq E$. Then we have
(1) $E^c(B) = (\cap E^c(b) \text{ for } b \in B)$ and $E^c[B] = (\cap E^c[b] \text{ for } b \in B)$.
(2) $\langle A - B, E\uparrow(A - B)\rangle = \langle B', E\uparrow(B')\rangle$ is a sub-hypernet of $\langle A - \{b\}, E\uparrow(A - \{b\})\rangle$ for every $b \in B$.
(3) $\langle A, E^c(B)\rangle$ is a sub-hypernet of $\langle A, E^c(b)\rangle$ for every $b \in B$.
(4) $\langle B', E[B']\rangle = \cap \langle A - \{b\}, E[A - \{b\}]\rangle$ for $b \in B$, where $B' = A - B$, so the order of the deletion of the $b \in B \subseteq A$ does not affect the result. ♦

Proof:
(2) and (3) follow at once because it is less "damaging" to $\langle A, E\rangle$ to remove one $b \in B$ from

<A, E> than 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}]> 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>. ♦

2.5 Hypertrees

The simplest hypernet in which to conduct searches will be produced if <A, E> is trimmed to a hypertree or a spinney. These structures are described below.

Definition 2.5.1: 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.
- For every a, b ∈ A of a hypertree <A, E_T>, either $\lambda(\{a, b\}) = \varnothing$ 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>.

Hypertrees are, as one might expect, typical of trees and play the same sort of role in hypernet theory as trees do in graph theory. Hypertrees are simple, but can be important as sub-hypernets of <A, E>. We present a few theorems that help to characterize hypertrees.

Theorem 2.5.1: 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 belong to one label set then there is an arc that has no label from T, which is impossible, so 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 2.5.2: 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 = <A, E_T> 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 2.5.2: An element $a \in A$ of a hypertree T = <A, E_T> is a pendant of T iff there is precisely one edge $E_i \in E_T$ with $\{a\} \subset E_i$ that is incident with a. ♦

Proof: If $a \in A$ is a pendant then we must have precisely one edge $E_j \in E_T$ incident with a and with $\{a\} \subset E_j$ so that d(a) = 1. Further, there can be no other edge $E_k \in E_T$ that is incident with a, because then d(a) would not be 1 and so a would not be a pendant of T. Conversely, if we have precisely one edge $E_j \in E_T$ with $\{a\} \subset E_j$ that is incident with a then it is clear that d(a) = 1, so a is a pendant of T. ♦

Next we meet a property of hypertrees that does not appear for trees.

Theorem 2.5.3: Deletion of a pendant $a \in A$ from a hypertree T = <A, E_T> 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 $a \in (E_i - \{a, d\})$ 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 2.5.3: Given any connected hypernet <A, E>, T = <A, E_T> with $E_T \subseteq E$ is said to be a ***spanning hypertree*** of <A, E> iff T is a minimally connected sub-hypernet of <A, E>.♦

Theorem 2.5.4: Every connected hypernet <A, E> has at least one spanning hypertree. ♦

Proof: <A, E> is connected. By part (2) of theorem 2.5.1, if <A, E> has no circuits then it is a hypertree and is of course spanning. If <A, E> 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 <A, E>. ♦

Let <A, E> be a hypernet and let T = <A, E_T> be a spanning hypertree of <A, E>. The $|E_T| = |A| - 1$ edges are called ***branches*** of <A, E> with respect to T, and the remaining $|E - E_T|$ edges of <A, E> 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 ∈ A and E_i, E_j ∈ E, E_i ∈ E_T, E_j ∈ E'_T, and $E_i \neq E_j$. This leads to the following definition.

Definition 2.5.4: 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 ∈ 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 ∈ E different from the edge E_i ∈ E_T by which the same two vertices are adjacent in T, is called a ***spinney*** of <A, E>.♦

A spinney has no circuits.

The following theorem is a standard for spanning trees, but for hypertrees it has a slight twist in the tail.

Theorem 2.5.5: 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 ∈ A, transferring any E_i ∈ (λ({a, b}) - λ_T({a, b})) to λ_T({a, b}), where λ_T is the labelling function of T, yields a connected spanning sub-hypernet <A, (E_T ∪ {E_i})> of <A, E> such that <A, (E_T ∪ {E_i})> has precisely one closed walk of length 2 that uses two distinct edges but only one arc {a, b}. ♦

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

While we will not pursue the topic here, the next definition opens the way to the study of circuits in terms of spanning hypertrees.

Definition 2.5.5: 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 ∪ {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>. ♦

An analysis of circuits in a trimmed <A, E> is part of further trimming to a hypertree or spinney. We have seen how to use fast access cascades, or in this case, equivalently, limited access cascades, on the arcs (without labels) of a trimmed <A, E> to find out whether any given A_i ∈ A lies on at least one circuit - see constructional scheme 1.3.4.

2.6. Connectivity and Cut-Sets

Another approach to the analysis of <A, E> is to examine its "connectivity" - see below. An introduction to the notion of connectivity in hypernets follows. We focus on the question of connectedness of components of <A, E> (or a trimmed <A, E>). How "strongly" are they connected, and what do we have to do, minimally, to destroy their connectedness? This clearly is of some importance when "trimming" before a search.

Definition 2.6.1: 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 A1 and A2 then any minimal set of edges of <A, E> the deletion of which cuts all the paths a_1 — a_2 with $a_1 \in A1$ and $a_2 \in A2$ 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 2.6.2: 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 $\lambda(\{a, b\})$, and that this does not delete the arc between a and b unless $\lambda(\{a, b\}) = \{E_i\}$.

Theorem 2.6.1: Let <A, E> be a connected hypernet. Then vc<A, E> ≤ 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> ≤ ec<A, E>. It is clear that ec<A, E> = min d(a). ♦

The next three theorems present simple characteristics of an edge cut-set.

Theorem 2.6.2: $R \subseteq E$ is an edge cut-set of a spinney S = <A, E> iff there is at least one pair $\{a, b\} \subseteq A$ for which $R = \lambda(\{a, b\})$.♦

Proof: If $R = \lambda(\{a, b\})$ 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}) for some a, b ∈ A, and no proper subset of R will "cut" a from b, so R =λ({a, b}) . ♦

Theorem 2.6.3: 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 2.6.4: 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. ♦

Theorem 2.6.5: A set R ⊆ 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 ⊆ 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 2.6.1: Let R ⊆ 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 $E_\ell \in R - \{E_k\}$ such that E_ℓ is a bridge in <A, E - $\{E_k\}$>. Then $\{E_k, E_\ell\} \subseteq R$ 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 $E_\ell \in R - R^t_2$ such that E_ℓ is a bridge in <A, E - R^t_2 >. Then $R^t_2 \cup \{E_\ell\}$ 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>. ♦

Comment: What we need to do is to find a bridge in (a sub-hypernet) of <A, E>, if one exists - see definition 2.2.4. First we recall that if E_i is a bridge in <A, E> then there is a vertex adjacency {a, b} in <A,E> such that λ ({a, b}) = {E_i}, where |A | ≥ 2. To find a bridge in a hypernet (with two or more vertices) then, we choose any vertex adjacency from the edge table of that hypernet and read the edge table to find whether that vertex adjacency occurs more than once. If it does then the edge on our choice vertex adjacency is not a bridge. If it does not, i.e. if it occurs only once in the edge table, then the edge on our vertex adjacency is a bridge in our hypernet. Test every vertex adjacency in turn until we find a bridge and stop, or until we have tested all vertex adjacencies and found no bridge.

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>.

What about spinneys?

Theorem 2.6.6: B ⊆ 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 ∉ B and b ∉ 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 ∠ <A, E>, so <A - B, E↑(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>. ♦

There is more about spinneys.

Theorem 2.6.7: Let <A, E> be a connected hypernet and B ⊆ 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 $E_i \not\subset B$.

(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 = {$E_i \in E \mid E_i \subseteq B$}.

(1) If T is a spanning hypertree of <A, E - E↑B > then every $E_i \in E_T$ has $E_i \not\subset B$ 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 $E_i \not\subset B$ then every $E_i \in E_T$ belongs to E - E↑B, so deletion of E↑B ⊆ E from <A, E> 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) = {$E_i \in E$ | (a, E_i, b), a, b ∈ A and (E_i - {a, b}) ∩ B ≠ ∅}, 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 or at least one pendant of T that has a ∈λ({c, d}) for at least some c ≠ a and d ≠ a, a, c, d, ∈ A but c, d ∉ B, 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>. ♦

A little more about the usefulness of spinneys follows, and then we close by pointing out a relationship between a cut-vertex and an edge cut-set.

Theorem 2.6.8: 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}) ∩ B ≠ ∅, 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}) ∩ B ≠ ∅. 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}) ∩ B ≠ ∅, i.e. $\lambda_S(\{a, b\}) \subseteq E(B)$, then <A, E - E(B)> is disconnected. ♦

Theorem 2.6.9: If a ∈ A is a cut-vertex of a connected hypernet <A, E>, but not of <A, E - E(a)>, then E(a) = {$E_i \in E$ | (c, E_i, d) is a vertex adjacency by E_i in <A, E> and a ∈ (E_i - {c, d})} includes an edge cut-set of <A, E>. ♦

Proof: Deletion of a ∈ 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↑(A - {a}) ⊆ (E - E(a)). In corollary 2.8.2 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 contra-positive is:
If a is not a cut-vertex of <A, E> then <A, E - E (a) > is connected.

2.7 Blocks

Inside components of <A, E>, or a trimmed <A, E>, we may have "less vulnerable" sub-hypernets called ***blocks***. Producing a relevant block in a component of <A, E>, by "trimming", can be important in restricting search regions in <A, E>. An introduction to blocks follows.

Definition 2.7.1: 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>.

Characterizing blocks is reasonably easy, and can be important with reference to searching in blocks of <A, E>. The following theorems approach such a characterization, culminating in theorem 2.7.3.

Theorem 2.7.1: 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 2.7.2: 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}, $G\uparrow(B - \{b\})$> 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}, $G\uparrow(B - \{b\})$> 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 2.7.1:
(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 ∈ B, each c ∈ B with c ∈ (E_i - {a, b}) would be a cut-vertex of <B, G>.
(3) If there were such an a ∈ B, it would be a cut-vertex of the block <B, G>. ♦

Theorem 2.7.3: 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 ∈ 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 ∈ B of a block <B, G> ∠ <A, E>, there exists a path P_1 joining a and c in <B, G> that satisfies the following conditions:
   a) P_1 has length ≥ 2.
   b) Given any b ∈ (B - {a, c}) 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 ∈ 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). ♦

A final small point is set out in the next theorem.

Theorem 2.7.4: Let <B_0, G_0> and <B_1, G_1> be distinct blocks, of a connected hypernet <A, E>, for which $B_0 \cap B_1 = B_{01} \neq \emptyset$. Then B_{01} = {b}, a singleton, and given any a ∈ (B_0 - B_{01}) and any c ∈ (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 ∈ $B_0 \cup B_1$ such that b is a cut-vertex of <$B_0 \cup B_1$, $G_0 \cup G_1$>. Now b ∉ (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 ∉ (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 $\langle B_0, G_0 \rangle$ such that b is not between a and p on $a — p$ because b is not a cut-vertex of $\langle B_0, G_0 \rangle$. Similarly we can find a path $p — c$ in $\langle B_1, G_1 \rangle$ such that b is not between p and c on $p — c$ because b is not a cut-vertex of $\langle B_1, G_1 \rangle$. 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 $\langle B_0 \cup B_1, G_0 \cup G_1 \rangle$. Thus there is no such $p \in B_{01}$, so $B_{01} = \{b\}$, and since b is a cut-vertex of $\langle B_0 \cup B_1, G_0 \cup G_1 \rangle$ it follows that b must be between a and c on every path $a — c$ in $\langle B_0 \cup B_1, G_0 \cup G_1 \rangle$ where $a \in B_0$ and $c \in B_1$ and $a \neq b$ and $c \neq b$. ♦

3: Knowledge Hypernets

3.1 Derivability

We now turn our attention to the hypernet equivalent of a CRKS, called a Knowledge Hypernet (KH). Much of the work on KHs parallels that on CRKS's. In a KH the data items of the NET are concept-names that could label course units.

Let <A, T> be any CRKS, and let M(<A, T>) = <A, E> be the unique hypernet that arises from <A, T> by abstraction M. <A ,E> is then simply the hypernet that is abstracted from <A, T> by replacing each tuple $T_i \in T$ with the tuple set E_i of T_i and preserving each vertex adjacency. It is clear that <A, E> inherits the CRKS characteristics of <A,T> and we call every hypernet that is abstracted from a CRKS a Concept-Relationship Knowledge Hypernet, abbreviated to KH. Thus a KH has the following inherited properties and these may be regarded as a definitive description of a KH.

By a ***Concept-Relationship Knowledge Hypernet,*** or simply a **KH**, we mean a hypernet <A, E> such that

(1) for all $a \in A$, $a \in E_i \in E$ for at least one (non-loop) edge E_i, so $E[a] \neq \varnothing$. Thus each $a \in A$ is associated with at least one other vertex of <A, E>.
(2) There are no loop edges in <A, E>.
(3) <A, E> is ***complete***, by which we mean here that <A, E> has no isolates.
(4) Every $a \in A$ is derivable in <A, E>.
(5) There is at least one $p \in A$ that is a (hyper-)***primary*** of <A, E>.
(6) There is at least one $g \in A$ that is a (hyper-)***goal*** of <A, E>.
(7) <A, E> has no closed derivation walks. ♦

From this point on we can visualize a direction for every vertex adjacency {a, b}, a, b ∈ A, in any KH <A, E>, namely the direction imposed by derivation. Thus we may replace arcs with arrows in each KH. How do we view the (hyper-)primaries and goals of a KH?

Let <A, E> be any KH. Then p is a (hyper-)***primary*** of <A, E> iff 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 only one (trivial) derivation by a path of length zero and a set of hypothesis $X = \varnothing$ and
(2) b is derivable by virtue of at least one X and betweenness sequence, for the vertex adjacency {p, b}, that starts with p and ends with b.

Next, g is a (hyper-)***goal*** of <A, E> iff 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, E> is a derived vertex, we must have the following:

(1) There is at least one (hyper-)primary $p \in A$ of <A, E> for which there exists at least one vertex adjacency (p, E_i, b), $b \in A$, in <A, E> for which every member of $(E_i - \{b\})$ is a primary of <A, E>.
(2) For every (hyper-)goal $g \in A$ of <A, E>there is at least one derivation path, from some primary to g, for which every betweenness sequence ends with g and has no other entry of g in it.

We will now henceforth drop the prefix *hyper-* for KHs.

It is easy to show that, in a KH <A, E>, no primary of <A, E> is reachable from any other primary of <A, E> by means of any derivation path. The same applies to the goals of <A, E>. Notice that in every KH there must be at least one edge in which all but one member of that edge are primaries of that KH, just as is the case for CRKS's.

We now turn to a description of the inherited derivability in a KH. It is essentially the same as that for a CRKS, but we repeat it here, and examine it quite closely in section 3.2. It is vital to have a complete grasp of the technical aspects of derivability when working with CRKS's and KHs in practice.

Definition 3.1.1: A ***betweenness sequence*** for a path-family $f(a_1 — a_n)$ in a hypernet <A, E> is found as follows. First, for all the members of $\lambda(\{a_i, a_{i+1}\})$, 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 $\{v \in A \mid v \in E_{ix}\}$, i = 1, 2, ..., n - 1, 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.) ♦

Next we deal with the detail of the inherited derivability in a KH. The following definition of derivability should be read in conjunction with that for formal schemas.

Definition 3.1.2:

(1) Given any KH <A, E> and a set $X \subseteq A$, we say that $a \in A$ is ***immediately derived from hypotheses*** 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 $(E_i - \{a_{n(i)}\})$ a member of X.

(2) Given any KH <A, E> and a set $X \subseteq A$, we say that $a \in A$ is ***derivable in terms of hypotheses*** 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 $X = \varnothing$ or such that every $x \in X$ is derivable from P.

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

Definition 3.1.3: The ***context-hypernet*** of $a \in A$ in a knowledge hypernet <A, E> is a hypernet <A, E>[a] = <A[a], E[a]> $\angle$ <A, E> that is defined as follows. E[a] is, as defined before, the set of all $E_i \in E$ that have $a \in E_i$, and $A[a] = \{b \in A \mid$ b belongs to at least one of the $E_i \in E[a]$ }.♦

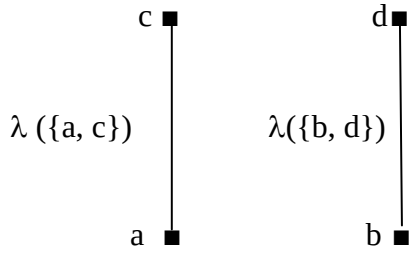
Thus we can write $A[a] = \{\cup E_i \mid E_i \in E[a]\}$. Since $E[a] = E\uparrow A[a]$, because $E\uparrow A[a]$ is the set of all $E_i \in E$ with $E_i \subseteq A[a]$ and each such E_i must have $a \in E_i$ given that $A[a] = \{\cup E_i \mid E_i \in E[a]\}$, we can also write $<A,E>[a] = <A[a], E\uparrow A[a]>$, the maximum sub-hypernet of $<A,E>$ that is induced by $A[a] \subseteq A$. So $<A,E>[a] = <A[a], E[a]> = <A[a], E\uparrow A[a]>$. Deleting $a \in A$ from a hypernet $<A,E>$ entails deleting all the vertices and edges of $<A,E>[a]$.

Definition 3.1.4: A KH <A, E> is said to be ***connected*** iff there is at least one path between every pair of distinct vertices in <A, E>.

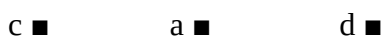
As for CRKS's, we have the following:

Theorem 3.1.1: If a KH <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 hypernet, which is complete but not connected:



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



The next two theorems tell us something useful about paths in a KH.

Theorem 3.1.2: Let <A, E> be a KH. There is at least one derivation path that joins each primary of <A, E> to some goal of <A, E> in <A, E>, and there is at least one derivation 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 "derivation semi-path" that is not a derivation path. ♦

Theorem 3.1.3: Let <A, E> be a KH, and let $a \in A$ be neither a primary nor a goal of <A, E>. Then there is at least one derivation 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 KH, a is a derived vertex in <A, E>. Since a is a derivable, there is a derivation path p—a, p ∈ A, with p primary, in <A, E>. By theorem 3.1.2 this path must continue on to some goal of <A, E>.♦

We now need to say something more definite about paths in a KH.

Constructional Scheme 3.1.1: To construct a derivation path tree, for a KH <A, E>, displaying and distinguishing every derivation 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 KH. It is clear that we can follow paths in a KH in the "derivation direction" only. This directional ordering on paths in a KH may appear just to reduce a KH to a CRKS, but in the case of a KH we have

(1) a simplification of the process of finding CRKS isomorphisms and
(2) no ordering and no repetition of vertices in the edges by which vertices are adjacent other than preservation of vertex adjacencies.
(3) In an interpretation of a KH <A, E>, the tuple that arises from any $E_i \in E$ can use the members of E_i in any order and can repeat any $v \in E_i$ any number of times in that tuple.

Thus we have the potential, for example, to use any teaching meta-language when we pick an interpretation of a KH in the educational applications mentioned in [GVS99]. The KH model is more flexible than the CRKS one in applications, and we have a strong link between the two models.

One final point before we tackle the constructional scheme: Derivability of a vertex b by virtue of a path a — b in a hypernet 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 (derivation) 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 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 that $c_{n(i)}$. Repeat this for every $E_i \in E$. The resulting tree exhibits, along the paths from the root, every (derivation) 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. ♦

Comment: Similar to CS 1.2.7.

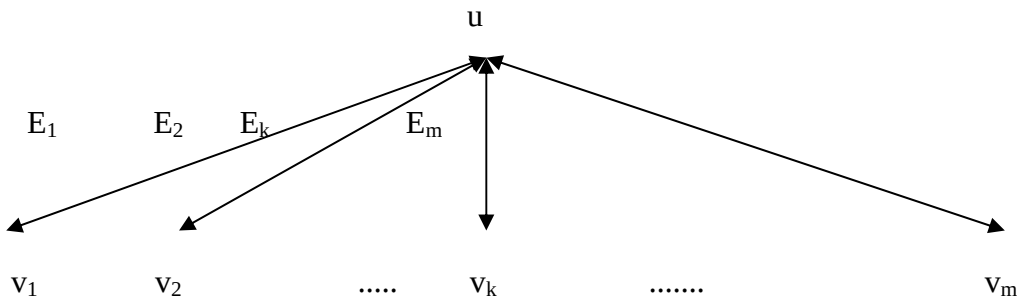
Constructional Scheme 3.1.2: Find all the derivation paths between vertex u and vertex v in a KH <A, E>. Because of the derivation induced directionality in <A, E>, we can think of ourselves looking for all paths "from" a given $u \in A$ "to" a given $v \in A$.
First we should note that we can run a fast access cascade against the derivational direction in

any KH just as easily as with this direction or without direction - see section 1.4.

(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'> $\angle$ <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. ♦

Comment: Similar to CS 1.2.8.

Before we move on to consider some theorems about KHs, we should mention the following point.

Theorem 3.1.4: Let <A, E> be a KH and consider any vertex adjacency / arc $\{A_i, A_j\}$ in <A, E>. Let $\lambda(\{A_i, A_j\}) = \{E_1, E_2, \ldots, E_n\}$, and suppose that A_j is derived from A_i with E_k for at least one value of k = 1, …, n. Then A_i is not derived from A_j by any member of $\lambda(\{A_i,A_j\})$.♦

Proof: If A_j is derived from A_i with any member E_k of $\lambda(\{A_i,A_j\})$, and A_i is derived from A_j with any member E_m of $\lambda(\{A_i,A_j\})$, then <A, E> has a closed derivation walk, which contradicts the fact that <A, E> is a KH. ♦

3.2 Knowledge Hypernet (KH) Theorems

As for CRKS's, we have the following key point:

Theorem 3.2.1: A KH <A, E> can be generated by a limited access cascade from 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>.♦

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 (derivation) path p — v, of length n, in $<B_n, E_n>$, and each such path has a betweenness sequence S. Every member of B_0 is primary, so it is trivially a derived vertex by virtue of a derivation path of length zero. B_1 is the union of a number of sets of the form $B_0 \cup \{t\}$ for at least one non-primary t $\in$ A, and, in $<B_1, E_1>$, E_1 is a set of edges, each with one or more primaries and t, and each such edge labels a vertex adjacency, i.e. a path of length 1, {p, t} where p is some primary. This path is clearly a derivation path for each such t in <A, E>, so each such t is a derived vertex and hence every member of B_1 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, $<B_n, E_n>$ = <A, E> because every a $\in$ A is derivable in <A, E> and the cascade generates only derivable new vertices in each step. ♦

The following theorems and corollaries bring out the major characteristics of derivability in a KH. These are essential in "rounding off" the initial recursive definition of derivability: We must, from now on in all applications of KHs, fully understand the notion of derivability, and this understanding can be enriched with the aid of these theorems and corollaries and their proofs.

Theorem 3.2.2: If a $\in$ A of a KH <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 cascade 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 3.2.3: Let <A, E> be a KH with a ∈ A any non-primary vertex of <A, E>. If a is derivable in <A, E>, by virtue of a path p — a where p is immediately derived from hypotheses X = ∅, then p is a primary vertex of <A, E>. ♦

Proof: p is immediately derived from X = ∅. The only vertices that can be immediately derived from ∅, trivially by a derivation path of length zero, are the primaries and isolates of <A, E>. <A, E> has no isolates, so p must be a primary of <A, E>.♦

We now set out some corollaries of theorems 3.2.1, 3.2.2, and 3.2.3.

Corollary 3.2.1: Let S be a betweenness sequence for a (derivation) path p — a in a KH <A, E>. If every t ∈ S is derivable in <A, E> then every t ∈ S is immediately derived from some X ⊆ A in <A, E>.♦

Proof: Let t ∈ S. Then t is derivable, so there must be at least one derivation path to t in <A, E>. Let {s, t} be an arc on that derivation path and let $E_i \in \lambda$ ({s, t}). Then let X = E_i-{t}, and t is immediately derived from X in <A, E>.♦

Corollary 3.2.2: If vertex a ∈ A of a KH <A, E> is derivable in <A, E> then a is immediately derived from some $X_a \subseteq A$. ♦

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 ∈ E, and let X_a = (E_i - {a}). Then a is immediately derived from hypotheses X_a in <A, E>.♦

The converse of Corollary 3.2.2 is as follows.

Corollary 3.2.3: Let <A, E> be a KH. If every a ∈ A is immediately derived from some set of hypotheses $X_a \subseteq A$, then every a ∈ A is derivable in <A, E>.♦

Proof: If a ∈ A is immediately derived from X_a then there is at least one arc {x, a} with at least one label element $E_a \in \lambda$ ({x, a}) such that {E_a - {a}} ⊆ X_a. Now consider x: x is immediately derived from some $X_x \subseteq A$, so we can find an arc {y, x} and a label element E_x for it such that {E_x - {x}} ⊆ X_x. Consider y. We can repeat the above for y, finding an arc{z, y} and a label E_y for it such that {E_y -{y}} ⊆ X_y. Continuing in this fashion we work our way back to a primary p, since <A, E> is finite and has at least one primary, which is immediately derived from X_p = ∅, and we have found a path p — a, in <A, E>, with an obvious betweenness sequence S for it.

Now suppose that p — a is not a derivation path. Then there is at least one b ∈ S that is not derivable in <A, E> - see the contra-positive of theorem 3.2.2. But we know that b is immediately derived from some $X_b \subseteq A$, so there is an arc {c, b} with a label E_j on it, and {y} ∪ {E_j - {b}} ⊆ X_b ⊆ A. As for a, we can trace back to define a path from a primary to b in <A, E>. If this path is a derivation path for each such b then we are done. If not, then there is at least one c in this path that is not derivable, and we repeat the procedure for each such c.

This regression of paths must stop with a derivation path in each case because <A, E> is finite and has at least one primary. It follows that p — a must be a derivation path, because every member of S is derivable in <A,E>, so a is derivable in <A,E> - see parts (3) and (4) of definition 3.1.2. ♦

Corollary 3.2.4: If every path incident with a primary of a KH <A, E> is a derivation path in <A, E> then every a ∈ A is derivable in <A, E>. ♦

The **proof** follows at once from the definitions of derivation path, derivable and KH. ♦

Corollary 3.2.5: Let <A, E> be a KH 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 if b ∈ S for some S, and by theorem 3.2.2 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 for some 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 $X \subseteq A$ with $X = \varnothing$ 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 $X = S \neq \varnothing$ for our path p — a and it follows that p — a is a derivation path for a in <A, E>.♦

Corollary 3.2.6: Let <A, E> be a KH. 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 3.2.3. 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 3.2.7: A KH <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. ♦

The **proof** follows at once from theorem 3.2.1 and Corollary 2.2.1. ♦

Corollary 3.2.8: Let <A, E> be a KH. Every a ∈ A is derivable in <A, E> 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, $E_i \in E$. Then a is immediately derived from $X_a = (E_i - \{a\})$, and every x ∈ X_a is derivable. Conversely, let every a ∈ A be immediately derived from some set X_a of hypotheses such that every x ∈ X_a is a derived vertex in <A, E>. Then there exists at least one vertex adjacency (x, E_j, a), $E_j \in E$, 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 3.2.4 that every b ≠ a in p — a is derivable. Let S be an appropriate betweenness

sequence for p — a, and set X = S. 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 3.2.4: Let <A, E> be a KH. 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 KH, 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 ∈ 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 hypernet <A, E> provides an automated method of testing <A, E> for KH status.

Definition 3.2.1: By a ***derivation adjacency*** in a KH <A, E> we mean a vertex adjacency (a, E_i, b), a, b ∈ A and E_i ∈ E, that lies on a derivation path for b in <A, E> and is such that every x ∈ (E_i - {b}) is derivable in <A, E>. ♦

Theorem 3.2.5: Let <A, E> be a KH. 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 KH 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 KHs outlined for CRKS's in [GVS99].

Definition 3.2.2: Given a KH <A, E> and any non-primary a ∈ A, we define a ***derivation path hypernet*** 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 hypernet 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. ♦

We should notice that a derivation path hypernet for a ∈ A in <A, E> is not generally unique because there may be several derivation paths for a in <A, E>.

Definition 3.2.3: Given a KH <A, E> with a ∈ A, we define the ***predecessor hypernet*** 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 =

∅. <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, including those 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 KH 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 3.2.6: Given a KH <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-hypernet of P(a). ♦

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

Definition 3.2.4: Let <A, E> be a KH and E_i ∈ E an edge of <A, E>. By a ***hypercluster*** for E_i we mean any minimal sub-KH, 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 for E_i then the resulting hypernet is not a KH that has E_i in it. ♦

A hypercluster for a given E_i ∈ E in a KH <A, E> is not generally unique. A cluster <A, T> for a tuple T_i has vertex set A at least the tuple set of T_i and has at least T_i ∈ T. A hypercluster <A, E> for an edge E_i has E_i ⊆ A and E_i ∈ E, i.e. it has at least vertex set E_i and at least edge set {E_i}.

Constructional schemes to find the D(p — a), and P(a), in a KH <A, E> are easily adapted from [GVS99]. The last three definitions 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 3.2.7: C is a cluster for T_i ∈ T in a CRKS <A, T> iff D is a hypercluster for E_i = I[T_i] in a KH <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, and the definition of a KH. ♦

3.3 Restricting the Domain of a Search

Given a KH <A, E>, it is often important to have ways of "trimming" <A, E>, i.e. of restricting the region of search before a search for items begins. There are four basic ways to do this in hypernet theory, each of which confines a search to a "relevant" sub-hypernet.

Suppose that a search is to begin with a set $A^0 \subseteq A$. Items pertinent to this choice of A^0 lie in the sub-hypernet found by running a fast access cascade, (either "forward" or "backward" or independent of "derivation direction", depending on the kind of search intended), starting with $<A^0, E^0> = <A^0, \varnothing>$, in <A, E>. Every walk in the resulting sub-hypernet $<A^n, E^n>$ can be followed through and to items that are related to a member, or to members, of A^0. The cascade can be run automatically to termination, or it can be run interactively step-by-step until the user decides that items in further steps will be irrelevant for his/her purpose.

Another interactive method of search is to run a sequence of fast access cascades. We start with $<A^0, \varnothing>$ and run the first cascade until we reach a step in which we want to discard some items. Thus we reach $<A_1^{n(1)}, E_1^{n(1)}>$, and then we choose the sub-hypernet $<A_2^0, E_2^0> \angle <A_1^{n(1)}, E_1^{n(1)}>$ induced by our choice of $A_2^0 \subseteq A_1^{n(1)}$. Now we start a new fast access cascade in <A, E> with $<A_2^0, E_2^0>$ and run it till we wish to discard some items from $<A_2^{n(2)}, E_2^{n(2)}>$. Next choose $<A_3^0, E_3^0> \angle <A_2^{n(2)}, E_2^{n(2)}>$, the sub-hypernet induced, by $A_3^0 \subseteq A_2^{n(2)}$, in $<A_2^{n(2)}, E_2^{n(2)}>$, and run a third fast access cascade from $<A_3^0, E_3^0>$ in <A, E>. Continue until we decide to stop or the sequence of cascades terminates.

A different sub-hypernet of <A, E> that will contain items pertinent to the choice of A^0, i.e. related to the member or members of A^0 in <A, E>, is the one found by running a limited access cascade from $<A^0, \varnothing >$ in <A, E>. Again this cascade can be run until it terminates, or it can be run step-by-step as for fast access cascades. Of course the resulting sub-hypernet may be $<A^0, \varnothing >$. This will happen if no edge incident with any member of A^0 is a subset of A^0 but for one member of that edge (for use in the first step of the cascade).

We can of course run a sequence of limited access cascades as described above for fast access cascades. We can also run a sequence of cascades from $<A^0, \varnothing >$, in <A, E>, in which we not only choose what sub-hypernet with which to begin the new cascade at each stage, but also whether to continue with a fast access cascade or a limited access cascade.

A third method of constraint of search domain in <A, E> falls into two sub-types. If the user is interested in every edge of <A, E> that involves a given $A_i \in A^0$ then this information is to be found in the context-hypernet <A, E> $[A_i]$ of A_i in <A, E>. This is of course easy to read off from the edge table of <A, E>.

If A^0 has more than one member then we have two possibilities. The first is to form the join of the context-hypernets of the members of A^0, and the second is to form the meet of those context-hypernets over the members of A^0. In the first case the user has potential access to every edge of E that involves vertices related to at least one member of A^0, and in the second case to every edge of E that involves vertices related to all members of A^0. Thus in the first case the set of edges $E[A^0]$ induces the join and in the second case the edges are those of the meet of the $E[A_i]$, $A_i \in A^0$, over A^0, and these edges induce the meet of the context-hypernets of the $A_i \in A^0$.

3.4 Trimming a Knowledge Hypernet

We now return to the prerequisite chain of figure 1.4.3. The hypernet version presented in section 1.4 is an example of a knowledge hypernet if the vertex adjacencies are chosen appropriately. In figure 3.4.1 below we display the sub-hypernet that arises from a fast access cascade that is run backwards, i.e. opposite to the derivation direction, from module 454 in the hypernet equivalent of figure 1.4.3. This is one example of trimming, and the result is itself a KH. We then run a limited access cascade from the only primary, the dummy vertex, demonstrating a partial ordering of the modules in the "routes" from the dummy to 454. This information can be abstracted from figure 1.4.3. In the KH version, that abstraction is a simple matter of reading from the edge table of the whole KH, which can be done by a straightforward computer program.

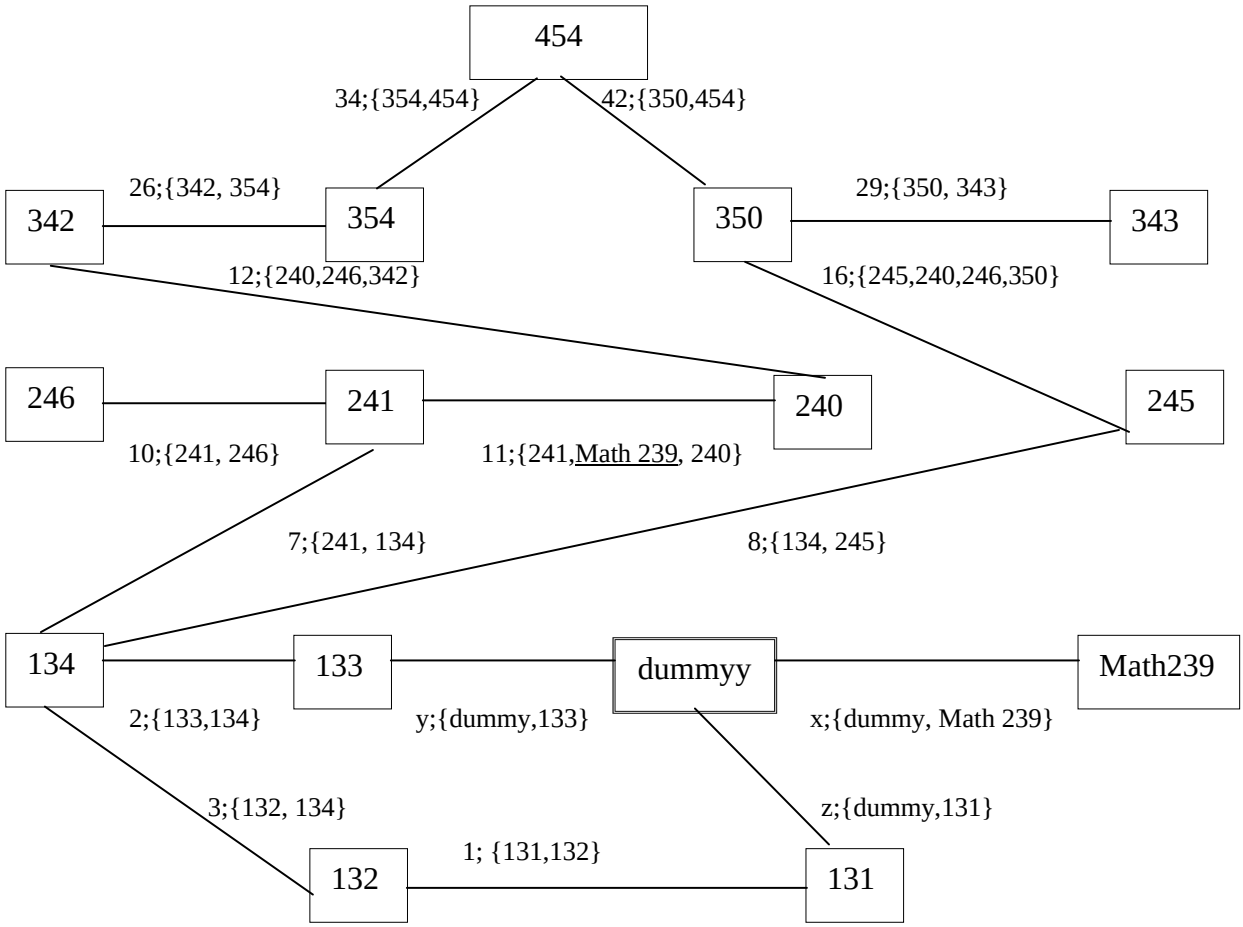


Figure 3.4.1: An example of a trimmed KH

In a fast access cascade backward from 454, the new vertices met in each step are:

Step 1	454
Step 2	354,350
Step 3	342, 343, 245, 240, 246
Step 4	134, 241, Math 239
Step 5	133, 132, dummy
Step 6	131

That cascade follows edges set out below:

34	{454, 354}	8	{134, 245}
42	{350, 354}	7	{241, 134}
26	{342, 354}	2	{133, 134}
29	{350, 343}	3	{132, 134}
16	{245, 240, 246, 350}	1	{131, 132}
12	{240, 246, 342}	x	{dummy, Math 239}
11	{241, Math 239, 240}	y	{dummy, 133}
10	{241, 246}	z	{dummy, 131}

In a limited access cascade from {dummy}, the new vertices met in each step are:

Step 0	Dummy vertex	Step 4	240, 246
Step 1	131, 133, Math 239	Step 5	350, 342
Step 2	132, 134	Step 6	454, 343, 354
Step 3	245, 241		

The dummy vertex is the only primary. 454, 343, Math 239 and 246 are the goals of the trimmed KH. The information in this table is incomplete in the sense that we are not told how, and at what stage a student can get to Math 239 on his way to 454. Our limited access cascade from the dummy, shows how a student can progress through the curriculum to 454. If we run a fast access cascade backward from another fourth year module, for example, and then join the resulting KH with that for 454, we get an interesting KH in which to now run a limited access cascade from the dummy vertex. Clearly the join and meet of the KHs will yield important information. KH representation of a curriculum lends itself to proper planning of a curriculum via simple analysis of the resulting KH as we begin to see here - see chapter 4,5 and 6. Here we notice, in passing, the critical vulnerability of the “route map” to 454 at module 134.

3.5 Menger's Theorem

It is clear that we need techniques aimed at analysing a hypernet <A, E>. In particular we need to examine the inter-relational status of the vertices of <A, E> and to investigate the paths in <A, E>. The two are intimately associated.

Suppose that we have a (trimmed or untrimmed) hypernet <A, E>. We assume the <A, E> has no complete isolates in it. It certainly has walks, and hence paths, in it if it is to be useable, and in this section we begin to examine the "flow" of the search in <A, E>, from an initial sub-hypernet $<A^0, \varnothing> \angle$ <A, E>. Here is the first of a number of techniques that assist in an analysis of a NET and in the design of new material that is to be added to the NET, and this one involves Menger's Theorem in <A, E>.

We will introduce the theorem, and state and prove it, stage by stage in parallel for relation nets and hypernets; the left part denotes the part for relation nets, the right part that for hypernets.

Definition 3.5.1:

The **path-net** N(P) of a path P in relation net <A,T> is the minimum subnet <B,U> $\angle$ <A,T> that contains P. By this we mean that U $\subseteq$ 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. ♦

The **path-hypernet** N(P) of a path P in a hypernet <A,E> is the minimum sub-hypernet <B,U> $\angle$ <A, E> that contains P. By this we mean that U $\subseteq$ 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 3.5.2:

Two u $\rightarrow$ 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 $\rightarrow$ v paths in <A, T> is called an ***interdependent set*** iff $\cap N(P_r)$, 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 $\rightarrow$ v paths in <A, T>.♦

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 $\cap N(P_r)$, 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 to the paths in <A, E> = M [<A,T>].
The next two theorems show why interdependent sets are important in connection with vulnerability of paths between any two given vertices.

Theorem 3.5.1: (see also theorems 5.1 and 12.6 of [GVS99])

Let $\{P_0, ..., P_n\}$ be any interdependent set of $u \rightarrow v$ paths in <A, T>. Deletion of any $w \in (A - \{u, v\})$ that belongs to the vertex set of $\cap N(P_r)$ 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>.♦

Let $\{P_0, ..., P_n\}$ be any interdependent set of u — v paths in <A, E>. Deletion of any $w \in (A - \{u, v\})$ that belongs to the vertex set of $\cap N(P_r)$ 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 $w \neq u$ and $w \neq v$, of $\cap N(P_r)$, then it is between u and v on every P_r. Let w be a vertex of $\cap N(P_r)$, 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 $\cap N(P_r)$, which contradicts the hypothesis. ♦

Theorem 3.5.2: (see theorems 5.2 and 12.2 of [GVS99])

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

Let $S = \{P_0, ..., P_n\}$ be a maximal interdependent set of u — v paths in <A, E>. Deletion of any $w \in (A - \{u, v\})$ that belongs to the vertex set of $\cap N(P_r)$ 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 3.5.1 we know that deletion of w cuts all the $P_r \in S$. Assume that deletion of w from <A, E> cuts at least one u — v path $P \notin S$. 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 $P_r \in S$, S is not a maximal interdependent set because the vertex set of $(\cap N(P_r)) \cap N(P)$ contains {u, v, w}. The theorem follows. ♦

Theorem 3.5.3: (see theorems 5.3 and 12.8 of [GVS99])

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

The set of all u — v paths, in <A, E>, that are cut by the deletion of $w \in (A - \{u, v\})$ 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 $w \in (A - \{u, v\})$ from <A, E>. Then w is between u and v on every $P_r \in S$, and hence w belongs to the vertex set of every $N(P_r)$, $P_r \in S$. It follows that $\cap N(P_r)$ 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 $\cap N(P_{0r})$, 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 $\cap N(P_{1r})$, 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 3.5.3:

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

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. ♦

We go even further on the question of vulnerability with respect to paths between two given vertices, introducing some simple combinatorics with the next theorem and its corollaries.

Theorem 3.5.4: (see also theorems 5.4 and 12.9 of [GVS99])

If $\{S_0, ..., S_m\}$ is a partition of the set of all $u \rightarrow v$ paths in $<A, T>$ such that S_0 is a maximal interdependent set of $u \rightarrow v$ paths in $<A, T>$ and, for each $r = 0, 1, ..., m$, S_r is a maximal interdependent set of $u \rightarrow v$ 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 $\cap N(P_t)$ over $P_t \in S_0$ and w_r belongs to the vertex set of $\cap N(P_t)$ over $P_t \in S_r$, then there exists a separation $B(u \rightarrow v)$ for u and v in $<A, T>$ that has precisely m elements. ♦

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 $\cap N(P_t)$ over $P_t \in S_0$ and w_r belongs to the vertex set of $\cap N(P_t)$ 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. ♦

From definition 3.5.3 and the partitioning we have the following

Corollary 3.5.1: (Corollaries 5.1 and 12.1 of [GVS99])

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 3.5.4, is equal to the minimum number of vertices in a separation $B(u \rightarrow v)$ for u and v in $<A, T>$.♦

The minimum number of elements in a partition of the u — v paths in $<A, E>$ into maximal interdependent sets, constructed as in Theorem 3.5.4, is equal to the minimum number of vertices in a separation $B(u$ — $v)$ for u and v in $<A, E>$.♦

Corollary 3.5.2: (Corollaries 5.2 and 12.2 of [GVS99])

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. ♦

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 is not necessarily maximal. ♦

By introducing the notion of quasi-disjoint path, we get to Menger's Theorem, in its max-flow, min-cut form, in the next corollary.

Definition 3.5.4:

Let P_r and P_t be u → v paths in <A, T>, where u ≠ 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, T> iff they belong to distinct maximal interdependent sets of u → v paths in <A, T>. ♦

Let P_r and P_t be u — v paths in <A, E>, where u ≠ 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 3.5.1 in Mengerian form.

Corollary 3.5.3:

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

The maximum number of pairwise quasi-disjoint u — v paths in <A, E> is equal to minB(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 → v)| maximal interdependent sets as referred to in Corollary 3.5.1, and that B(u — v) is one of the corresponding separations. How many pair-wise quasi-disjoint u — v paths can we find in <A, E>? Certainly we can find at least min |B(u → 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 ∈ B(u — v) cuts more than one of these paths then those paths cut are not pair-wise quasi-disjoint paths. Further, we cannot find more than min |B(u → 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 → 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 3.5.4, which, in turn, is equal to the maximum number of pair-wise 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.

The user of a hypernet, particularly a KH, will be moving back and forth along chosen paths in <A, E> during any search, so we need to know as much as possible about the paths in <A, E>. Menger's theorem can tell us something about the paths available and the vulnerability of searches in <A, E>. We will return to Menger's Theorem and the notion of "flow" later.

We now turn our attention mainly to the case in which <A, E> is a KH. Much of the theory that follows also applies to the kinds of restriction of <A, E> as outlined in Section 2.3 if those restrictions are complete hypernets in the sense that no $a \in A$ has $d(a) = 0$ when we disregard loops.

3.6 Structural Analysis of a Knowledge Hypernet

We now turn to structural characteristics of KHs. These are similar to those exposed in the chapter on presentation strategies in [GVS99]. Note that we make some modifications to that chapter of [GVS99] here.

The most basic structural characteristics of a KH are its vertex basis and its edge bases. The set of primaries of a KH is its unique vertex basis, and, in the terminology of graph theory, the set of goals of a KH is its unique vertex contra-basis.

Next we re-visit Menger's Theorem. We visualize certain searches in terms of "flows", so Menger's Theorem is essential to this endeavour.

Application of Menger's Theorem in a KH yields two interesting insights into the structure of a KH. Let K = <A, E> be a KH with set of primaries P and set of goals G. Convert K to a KH Z as follows: Delete from K all edges 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-hypernet 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 $<A - \{v_0\}, E\uparrow(A - \{v_0\})>$ 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 $|B(\pi — \gamma)|$. 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 at least one 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 3.6.1: 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 3.6.1: 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 pair-wise 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 maximum 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 pair-wise 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 $|E(\pi — \gamma)|$.

Since two quasi-edge-disjoint paths can share an edge 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 3.6.2: 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 3.6.2: The measure of a maximum edge-flow for π and γ through K is less than or equal to min $|E(\pi—\gamma)|$.♦

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 pair-wise 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 $<B_0, E_0> \angle K$ be the hypernet that is defined to be the context-hypernet 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-hypernets 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-hypernet 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} has no π — γ paths. 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 3.6.3: 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 $\pi — \gamma$ 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 $\pi — \gamma$ 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 $\pi — \gamma$ paths in Z into sets $S'_0, S'_1, ..., S'_n$. Now two $\pi — \gamma$ 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 3.6.4: The measure of a maximum edge-flow for π and γ through Z is equal to $\min |G(\pi — \gamma)|$, where $G(\pi — \gamma)$ is an edge-flow-separation for π and γ in Z, i.e. $G(\pi — \gamma)$ is a set of edges such as $e_0, e_1, ..., e_n$ that generate a partition of $\pi — \gamma$ 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 3.6.5: If two $\pi — \gamma$ 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 $\pi — \gamma$ paths. ♦

Corollary 3.6.1: $\min |F(\pi — \gamma)| \leq \min |G(\pi — \gamma)|$ in Z. ♦

Proof: Follows at once from Theorem 3.6.5. ♦

Since deleting the context-hypernet of all vertices in all the $\pi — \gamma$ paths on which some vertex v lies is more destructive than deleting only the context-hypernet of v, we have:

Theorem 3.6.6: $\min |F(\pi — \gamma)| \leq \min |B(\pi — \gamma)|$. ♦

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

Theorem 3.6.7: $\min |G(\pi — \gamma)| \leq \min |E(\pi — \gamma)|$. ♦

Finally, for the same reason, we have:

Theorem 3.6.8: $\min |B(\pi — \gamma)| \leq \min |E(\pi — \gamma)|$. ♦

Thus we have:

Corollary 3.6.2:
$\min |F(\pi — \gamma)| \leq \min |G(\pi — \gamma)| \leq \min |E(\pi — \gamma)|$ 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.

Constructional Scheme 3.6.1: To find a set of quasi-disjoint $\pi — \gamma$ paths in Z.

(1) Choose any non-primary, non-goal vertex v_0 of $Z = Z_0$. Delete all $\pi — \gamma$ paths that use any edge that has v_0 in it. This deletes bundle b_0 of $\pi — \gamma$ paths; every such path is "cut" by deletion of v_0. Choose and mark any one path from bundle b_0. Delete the context-hypernet of v_0 from $Z = Z_0$. Let the resulting sub-hypernet of Z be Z_1.

(2) Repeat (1) with the subscript 0 running through the values 1, 2, 3, …, and subscript 1 running through values 2, 3, 4, …, defining v_1 and b_1 and Z_2, v_2 and b_2 and Z_3, and so on until, after deleting the context-hypernet of v_n from Z_n, there are no more $\pi — \gamma$ paths left in Z_{n+1}. The paths chosen, one from each bundle b_0, b_1, b_2, …, b_n, constitute a set of quasi-disjoint $\pi — \gamma$ paths in Z, and the set $\{v_0,v_1,\ldots,v_n\}$ constitutes a vertex separation for Z. While it is minimal, it is not necessarily a minimum separation. ♦

C**omment:** Z is a KH. Use CS 3.1.1 to construct the path tree for Z. Mark every $\pi — \gamma$ path on which there is an edge E_i with $v_0 \in E_i$. This is bundle b_0 of paths. Choose any one $\pi — \gamma$ path from b_0 and store that path. Delete the context-hypernet of v_0 from $Z = Z_0$ - see definition 3.1.3. - and let the resulting sub-hypernet of $Z = Z_0$ be Z_1. Now we just repeat as per step 2.

Constructional Scheme 3.6.2: To find a set of quasi-edge-disjoint $\pi — \gamma$ paths in Z.

(1) Choose any edge of $Z = Z_0$ that is not incident with π or γ. Call that edge E_0. Delete all $\pi — \gamma$ paths in Z_0 that use E_0. These paths constitute edge bundle b_0. Choose any one path from b_0. Now delete that edge from Z_0, and let the resulting sub-hypernet of $Z = Z_0$ be Z_1

(2) Repeat (1) with subscript 0 running through the values 1, 2, 3, …, and subscript 1 running through the values 2, 3, 4, …, defining E_1 and b_1 and Z_2, E_2 and b_2 and Z_3, and so on until, after deleting E_n from Z_n, no more $\pi — \gamma$ paths are left. The paths chosen, one from each bundle b_0, b_1, b_2, …, b_n, constitute a set of quasi-edge-disjoint $\pi — \gamma$ paths in Z, and the set $\{E_0, E_1,\ldots, E_n\}$ constitutes an edge separation for Z. While it is minimal, it is not necessarily a minimum edge separation. ♦

Comment: Similar to CS 3.6.1, but here we are deleting edges, so in this case we do not need to find a context-hypernet at any stage.

Matchings and **Coverings**. 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 at ***matchings*** and ***coverings*** as both are important facets of the structure of a KH. One of the key approaches to finding matchings is the construction of a bipartite graph G from a KH <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 3.6.3: A ***matching*** $M \subseteq E$ in a KH <A, E> is a set of edges of <A, E> that are pair-wise 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 pair-wise "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 3.6.4: A ***vertex cover*** of a KH <A, E> is a set of edges $E_c \subseteq E$ which is such that $\cup E_i$, $E_i \in E_c$, 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>.

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

Definition 3.6.5: 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>. ♦

(a) Starting with the primary edges of <A, E>, we can order the edges of <A, E> as follows:

(b) 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.

(c) Define 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. Now partially order the members of L_1 as follows. Let the ***order*** of each $\ell_1 \in L_1$ be $|\ell_1 \cap A_0|$ where A_0 is the set of all vertices that belong to any member of L_0, 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.

(d) Repeat step b with L_0 and A_0 replaced by L_1 and A_1, and L_1 and A_1 replaced by L_2 and A_2, where A_n, n = 1, 2, …, includes all previous A_i, and so on, until L_k has been defined and we then find $L_{k+1} = \varnothing$. We have then dealt with some of the edges of <A, E> in a partial order that consists of successive steps with a

partial ordering of edges in each step.

(e) 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 KH. 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 $\cup L_i \subseteq E$ may constitute only a very small subset of E, but we can consider it as displaying "core associations" among (some of) the vertices of <A, E>.

To implement this procedure is easy.

Another indication of the kind of association that should be examined in a KH <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 hypernet P(a) of a $\in$ A, a sub-hypernet that contains at least one derivation path, from a primary, 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-hypernet of P(a) that contains the join of the derivation path hypernets 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-hypernet*** of a in <A, E>, and it contains at least one derivation path hypernet, 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-hypernet 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-hypernet can be dealt with, and then P(a), and then finally the context-hypernet 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-hypernet of a $\in$ A in <A, E>. A recursive sub-hypernet is unique.

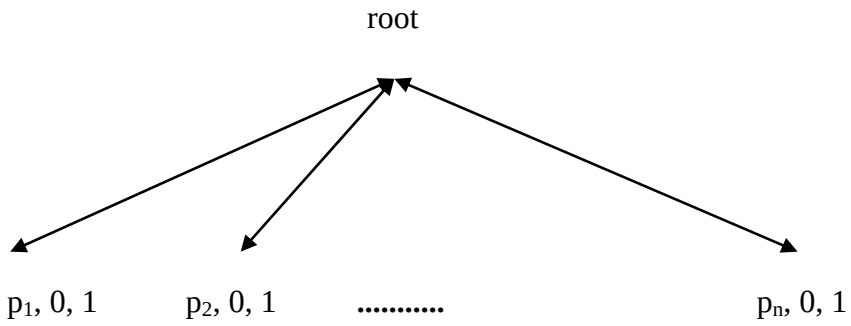
Deductive Complexity of a CRKS - see [GVS99] - can be usefully transcribed to a KH. It is clear that a limited access cascade from the primaries of a KH <A, E> generates a hierarchy, in <A, E>, in the form of a nested sequence of KH sub-hypernets of <A, E>. We will be concerned with that hierarchy and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

Definition 3.6.6: The ***deductive distance*** from the primaries of a KH <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 CS 2.1.1. 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 $E_i = \{u = c_1, c_2, \ldots, c_m = v\}$. 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 ∅ 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 ∅ 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 $v \in A$ 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 $v \in A$ 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 $n \geq 1$. Thus every vertex of <A, E> that lies in level $m \leq n$ 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 $m \leq n$, 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 $m \leq n$, 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 KH <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].

3.7 Gauges of Complexity

In this section we begin to see how we can investigate the complexity of a KH, from various points of view, by introducing some gauges of complexity for a KH. We will see how this is relevant to our model <A, E>.

Definition 3.7.1: The ***vertex context number*** of a $\in$ A in a KH <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-hypernet of a in <A, E>. ♦

Definition 3.7.2: By the ***degree*** d(a) of a $\in$ A in a KH <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 3.7.3: By the ***flow*** at a $\in$ A in a KH <A, E> we mean the number f(a) = min{id(a), od(a)}. ♦

Definition 3.7.4: By the ***path-multiplicity*** at a $\in$ A in a KH <A, E> we mean the number p(a)= id(a) $*$ od(a). ♦

Definition 3.7.5: By the ***local context number*** of a $\in$ A in a KH <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 KH model of a "real world" situation. Relatively low values will indicate a weakness of association among vertices.

Definition 3.7.6: Let <A, E> be a KH, 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 3.7.7: Let <A, E> be a KH. A sub-family $E_M \subseteq E$ is called a ***matching*** if the edges of E_M are pair-wise disjoint. ♦

Definition 3.7.8: A ***transversal*** of a KH <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 KHs 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 3.7.9: Let <A, E> be a KH, 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 $wdd(a) = \cup N_i$ where the union is taken over $i \in \{0, 1, ..., n - 1\} = n \in N$. ♦

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 3.7.10: Let a $\in$ A of a KH <A, E> belong to an n-slice N_n in <A, E> for some n $\in$ N. 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 KH <A, E> is the following.

Definition 3.7.11: Let <A, E> be a KH, 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 $S_c(B) \subseteq E$ defined by $S_c(B) = \{E_i \in E \mid B \cap E_i \neq \varnothing\}$. By the ***scope number*** of B $\subseteq$ A in <A, E> we mean $|S_c(B)|$. S_c (P) is called the ***primary scope*** of <A, E>, and $|S_c(P)|$ the ***primary scope number***. ♦

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

Definition 3.7.12: Let <A, E> be a KH with $E_i \in E$ and S $\subseteq$ A. The ***edge rank*** $r(S, E_i)$ of E_i with respect to S is defined by $r(S, E_i) = |S \cap E_i|$. ♦

Definition 3.7.13: By a ***vertex covering*** C of a KH <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 KH <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 $E_0 = \varnothing$ as usual. Suppose we have completed step n of the cascade, i.e. we have $<B_n, E_n> \angle <A, E>$. $(B_n - B_{n-1})$ 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 of E_i 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 $r_2[(B_n - B_0), E_j]$ class. Inside this equivalence class we define a third equivalence relation as follows. Let edge rank 3 be defined by $r_3(B_0, E_k) = |B_0 \cap E_k|$, with $E_k \in r_2[(B_n - B_0), E_j]$. This specifies equivalence 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 equivalence 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 equivalence 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 KH $<A, E>$. 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 $I [E_\ell]$, $E_\ell \in r_3[B_0, E_k]$, regarding those edges corresponding with minimal length tuples to be the least difficult in $r_3[B_0, E_k]$. This defines equivalence 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].

Definition 3.7.14: Let p — g be any primary to goal (derivation) path in a KH $<A, E>$, and write p — g as $p = A_1, <A_1, A_2>, A_2, <A_2, A_3>, A_3, \ldots A_q, <A_q, A_{q+1}>, A_{q+1} = g$. For all $<A_i, A_{i+1}>$, $i = 1, 2, \ldots, q$, let $|\ell (<A_i, A_{i+1}>)| = n_i$. The ***pumping constant*** for p — g is the product of all the n_i. ♦

The pumping constant for p — g tells us how many distinct p — g paths there are in the path-family f(p — g). The constant is ≤ the total number of distinct p — g paths in $<A, E>$. It tells a teacher how many choices he has in getting from p to g by the path-family f(p — g). The notion of a pumping constant also applies to any sub-path of p — g.

3.8 Accommodation, Analogy and Reasoning

Assuming the hypernet <A, E> is a KH, we now turn to three other facets:

- Adding to a KH.
- Analogy.
- Reasoning.

Definition 3.8.1: By an ***accommodation*** of a KH <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 3.8.2: In the case of unit accommodations and hypercluster accommodations of a KH <A, E>, we say that the accommodation is ***assimilated*** by <A, E> iff the restructured hypernet that results is itself a KH. ♦

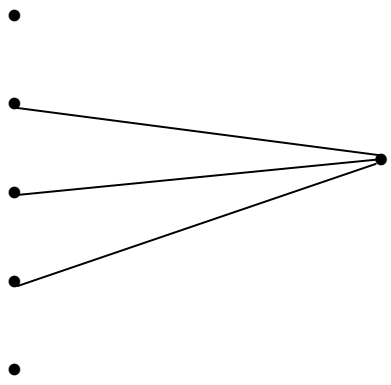
It is clear that a unit edge accommodation of a KH <A, E> in which all the members of the new edge are elements of A is the simplest form of accommodation. A unit vertex accommodation of a vertex $v \notin A$ will of course never be assimilated: We need to add in, as well, appropriate associations with members of A, in the form of new edges, to produce a context-hypernet for v that is assimilated by <A, E> if our objective is to construct KHs 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 (KH) CRKS isomorphisms in practical situations in which analogical modelling is used.

Next, let us point out that even though a hypercluster is, by definition, a (minimal) KH for a given edge, accommodating a hypercluster into a KH does not always lead to effective assimilation of that hypercluster. Certainly the join of the KH <A, E> and a hypercluster that is disjoint from <A, E> will yield a KH, so that hypercluster is assimilated by <A, E>, but this is a trivial situation of little importance: What we need to do is to 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 <∅, ∅>, the accommodation and assimilation are 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 KH and is such that the hypercluster introduced belongs to a component of <A', E'>. See section 4.3 for a simple example of (hyper)cluster accommodation.

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].

Before briefly discussing isomorphism, analogy, teaching by modelling and our models reasoning in KH theory, we should note the following. The notion of ***invariance*** is the basis of inductive reasoning in KH theory – see Chapter 9 of [GVS99]. It first arises in connection with the establishment of primitive concepts in the pre-linguistic and early linguistic stages of human learning. A ***primitive concept*** is one that is established by means of concrete examples only. At the lowest levels of learning, primitive concept-names must be treated as primaries – there is no other choice. Thus, for instance, a mother may initially partially establish the concept-name "red" by verbalizing to the child, in association with appropriate examples, "red roof", "red car", "red jersey" and so on. The invariant is the sound for "red". The rest of the words should be unconnected among the examples so as to provide verbal ***noise*** against which the invariant stands out. This kind of situation is represented by means of a KH that arises from the CRKS of a ***primitive binary relationship*** as follows:



A primitive binary relationship such as "x is red" for example, induces a ***primitive KH*** as depicted in the diagram above. The vertex on the right represents the invariant and is called an ***attention point.*** The vertices on the left represent a (growing) set of examples from which the invariant is ***inductively abstracted,*** from the noise of unassociated sound for example. From the left-to-right, each edge represents a statement of relationship of the form "is an example of". Read from left-to-right each edge in a primitive KH represents a trivial KH isomorphism, which is called an ***abstraction isomorphism*** in general. Such abstraction isomorphisms express the process of induction from examples that lead to an invariant. This serves as the basis of the CRKS/KH model of inductive reasoning and is called ***elementary induction.*** In general the examples on the left may be CRKS's/KHs and the vertex on the right an invariant (sub-) structure.

Once an attention point exists, and with it possibly a concept-name, the learner can start positively to seek new examples of the relevant attention point, thus inducing new edges. Such edges, read from right-to-left, are each associated with a statement of primitive (binary) relationship of the form "is a property of" for instance. Each such edge represents a trivial KH isomorphism, called an ***algorithmic isomorphism*** in the primitive KH that is under construction. Such algorithmic isomorphisms express the process of finding, or constructing, examples of an invariant. In general the invariant on the right may be a KH/CRKS for example, and the vertices on the left (sub-)KHs/CRKS's found, or constructed, all to be isomorphic with the KH/CRKS on the right.

While a primitive concept is established by means of examples all other concepts, called ***secondary concepts***, are established by relating them to primitive concepts and/or other

secondary concepts. In the case of a primitive concept, the more examples of it the better it is established. In the case of a secondary concept, the more statements of relationship that involve it, the better it is established. In both cases, the more statements of relationship, in a give CRKS/KH, that involve the given concept-name the more precise the meaning of that concept-name in that CRKS/KH, as gauged by its context-net in that CRKS/KH.

To see if two given CRKS's are isomorphic, we go via the two KHs subtracted from those CRKS's. If two KHs (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 KHs 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 KHs.

What, then, is the reason for introducing KHs 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 referring 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.

It could be useful, in defining search patterns in our KH <A, E> to find structurally analogous sub-hypernets of <A, E>.

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 KH <A, E> and the hypercluster KH <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 KH isomorphism is tested as follows. We define, at each stage of the "prediction" from <A, E>, an interpretation of the "predicted" KH, 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 KH 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 arises out of <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 KHs, 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 (- see also section 8.5 in [GVS99] -) is easily transcribable to KH representations of top-down algorithms. In fact, as pointed out in chapter 1, the entire treatment of problem solving in [GVS99] is best done in terms of KHs 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) KH. In the case of connectedness, which is clearly desirable, a fairly generous slice of the theory of hypernets is applicable in the analysis of the structure of the kind of hypernets referred to in chapter 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/KHs drawn up by the members of a class of learners.

CRKS models of reasoning were introduced in Chapter 9 of [GVS99], and all that is said there can be transcribed to KH 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 KH 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 KHs) 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 KH) is termed ***constructive*** reasoning. Some mainly heuristic guidelines for this task are set out in Chapter 9 of [GVS99].

We have proposed derivation as a model of deductive reasoning. More realistically we should refer to it as a model of ***inferential reasoning*** since every relationship/tuple is regarded as a rule of inference. Deductive reasoning, as one meets it in the formal languages of mathematical logic and computing, is much more restrictive than inferential reasoning. The "domain of descriptive application" of formal languages, both in mathematical logic and in computing, is severely limited by comparison with inferential reasoning, though the gross structure of all the formal languages involved is the same in every case.

4. A NET for use in Education

4.1 NETS in general

Consider a general NET; one in which the links between items are not necessarily associated with any relationship between items, but links which may be arbitrary. We introduce a structural model <A, E> as follows:

(1) <A, E> is a hypernet.
(2) A is a set of items.
(3) E is defined as follows. For every item $A_i \in A$ we introduce an edge set consisting of A_i together with all the items that are linked with A_i in <A, E>. From each such edge set $\{A_i, A_1, A_2, ..., A_{\ell(i)}\} \subseteq A$, for each $A_i \in A$, we introduce $\ell(i)$ edges $E_{i1}, E_{i2}, ..., E_{i\ell(i)}\}$in E where E_{ir}, r = 1,2,...., $\ell(i)$ is associated with a distinct arc between A_i and A_r, and that arc is labelled with set $\{A_i, A_1, A_2, ..., A_{\ell(i)}\}$. These are all the edges of <A, E>.
(4) <A, E> has no isolates.

Now suppose that we are given an initial sub-hypernet $<A^0, E^0> \angle$ <A, E>, where we may have $E^0 = \varnothing$, and wish to find all those items reachable with at least one member of A^0 in <A, E>. Run a fast access cascade from $<A^0, E^0>$ till that cascade terminates with a sub-hypernet $<A^n, E^n>$, and consider all items that belong to $A^n - A^{n-1} = \breve{A}$. It is quite easy to see that each item in $\breve{A}$ is reachable with at least one member of A^0, and that if we run a limited access cascade from $<\breve{A}, \varnothing>$ to automatic termination we will get $<A^n, E^n>$. Thus $<A^n, E^n>$ is a KH with primaries $\breve{A}$ and goals a subset of A^0. The derivation involved is, as for problem solution hypernets - see chapter 1 and [GVS99] - in a bottom-up direction.

Other methods of trimming <A, E> before a search are indicated in section 3.4 These would generally be implemented inside $<A^n, E^n>$, and in such trimming it would be preferable to preserve this bottom-up KH structure of $<A^n, E^n>$ in the trimming process, but that is of course not necessary. Further, it would also be appropriate to preserve connectedness in the (trimmed) components of $<A^n, E^n>$. We will not pursue the matter here but will end with the comment that hypernet theory, as applied to <A, E>, can be very useful in dealing with any NET such as <A, E>.

4.2 NETS in Education

In this section we briefly indicate how to set up, in principle, a NET <S,E>, called EDUNET, to represent curricula and study material. In setting up EDUNET we would follow the pattern of Part 1 of [GVS99], but convert all CRKS's there to KHs. This enables us to deal with isomorphism much more easily, and thence to free study material, at least partially, from the constraints of one particular teaching meta-language through the use of Language Equivalence (LE) – see section 1.4.

Nearly all the theory required for this endeavour appears in this work, so what remains here is descriptive, with reference to the appropriate terminology and theory.

Each item $s \in S$ of <S, E> arises from a CRKS of the form shown in Part 1 of [GVS99] by abstracting those CRKS's. All this entails is to replace each tuple by the edge that arises from i; $<A_1, A_2, ---, Aj, --- A_{n(k)}>$ with adjacency $<A_1, A_{n(k)}>$, in a CRKS, which becomes i; $\{M(A_1) = A_1, M(A_2) = A_2, ---, M(A_j) = A_j, ---, M(A_{m(k)}) = A_{m(k)}\}$ with adjacency $\{M(A_1), M(A_{m(k)})\} = \{A_1, A_{m(k)}\} = \{A_1, A_{n(k)}\}$, and $m(k) \leq n(k)$, where M is the (simple) abstraction used. Thus each such KH is represented by a vertex $s \in S$, i.e. an item in the NET <S, E>.

From each item we can use a trivial hypernet isomorphism to move to a corresponding edge j: $\{B_1, B_2, ---, B_k, ---, B_{m(j)}\}$ with B_1 the translation of concept-name A_1, $B_{m(j)}$ the translation of concept-name $A_{m(k)} = A_{n(k)}$, and B_k the translation of concept-name A_j for some k and j, and with m(k) = m(j). Then we can use j: $\{B_1, --- ,B_{m(j)}\}$ to define a corresponding statement of relationship, and hence a tuple $<B_1, ---, B_n, ---, B_{m(j)}>$, in the translation, where of course the arity of that tuple is at least $|\{B_1, ---, B_{m(j)}\}|$ and the adjacency is preserved, i.e. it is $<B_1 = I(B_1), B_{m(j)} = I(B_{m(j)})>$ where I indicates the (simple) interpretation used.

All the CRKS's defined, for example all those defined in [GVS99], can be incorporated in EDUNET. With appropriate coding of the KHs that arise from those CRKS's one can see EDUNET as a whole, with an item, i.e. a vertex of S, for each of the resulting KHs. Thus we can access any of these hypernets individually, or appropriate meets and joins of them. <S, E>, an EDUNET, thus has two "layers". The top layer, denoted by <S,E>, consists of a set S of items that are courses in a curriclum and a set E of edges. These are defined from prerequisite and parallel conditions between items/courses - we refer the reader back to the examples of this situation in section 1.5. As a result, <S,E >, ie.. the top or curriculum layer of an EDUNET, is a "prerequisite and parallel NET" that is itself a KH. The lower level of <S, E> is reached by "opening up" an item or items of S to reveal each item as a KH that arose from a CRKS. In using EDUNET, then, we can initialise with a concept-name or a set of concept-names, or an item or set of items, and follow relational and/or link edges, as the case may be, in any search. Following edges in E between items can establish a relevant sub-hypernet of EDUNET, for example in designing a degree course, after which a combination of edges between and in items can extract the course. Then we can use presentation strategies to guide teaching of that course, KH item by KH item. It is of course a CRKS interpretation of each KH item that is actually taught. (See also section 1.2 and section 1.4).

EDUNET can be defined in phases, and presentation strategies can be prepared, for selected courses, by subject experts working together or individually. With each accommodation of the growing KH EDUNET, one can use limited access cascades to check derivation and gauges to check "relational integratedness", i.e.integrity, of <S, E> and of study material, thus controlling the development of both edges in items and edges between items in different

circumstances, but using the same techniques.

Briefly stated, all of the theory covered thus far is applicable to EDUNET in both layers. How that theory is used in the representation, analysis and presentation of the study material section of EDUNET is clearly set out in [GVS99], there in terms of CRKS's but easily converted to KHs. The eight presentation strategies discussed in Chapter 5 of [GVS99] easily adapt to KHs. Chapter 8 of [GVS99] discusses the use of CRKS, now KH, isomorphism, which is much easier to deal with than the complex procedure set out in that chapter. Section 8.5 of [GVS99] tackles the vital question of problem solving, which is much improved with our switch to KHs for action diagrams. Sections 9.2, 9.3, 9.4, 9.5 and 9.6 of [GVS99] deal with the CRKS models of reasoning, and the whole discussion becomes even more relevant if one uses KHs in place of CRKS's. Finally, the whole of Chapter 10 of [GVS99] on the "potential uses of the CRKS model" is even more appropriate for EDUNET.

Of course one must ask "what is the reason for the existence of the notion of a CRKS now, in view of this approach via KHs"? The answer is that each of the KHs, i.e. of the $s \in S$, of the prospective EDUNET <S, E> arises from a CRKS in some teaching language, which is (very simply) abstracted to form that KH. Further, what is actually taught is not the KH, but an interpretation of that KH in the form of a CRKS in some teaching meta-language. The "bottom line" is that EDUNET starts and ends with statements of relationship among concept-names. The primary items of the KH <S, E> depend on the particular curriculum defined.

Finally we remind the reader that every feature of EDUNET, from setting it up to analysing and presenting the study material, is assisted by simple computer support that can be based upon the key constructional schemes in this work.

In summary, we have met two major structures, relation nets and hypernets, that are closely associated. The main focus is on CRKS's, prerequisite and parallel hypernets, and KHs, and these notions are brought together as facets of a prospective EDUNET which, then, is the principal application aimed for in the longer term. We close with some brief comments about an EDUNET.

The first thing to notice is that it is mandatory, in designing a CRKS - from which a KH can arise by means of a very simple abstraction - that a diagram be produced, step-by-step, in parallel with writing statements of relationships and construction of the appropriate tuples table. Such a diagram is an essential heuristic guide in the process of ensuring, a-priory, the existence of appropriate derivation paths and adequate integratedness as the design of the CRKS proceeds. It is also the visual key to presentation of a CRKS, both for learner and teacher, as can be seen from Chapters 5 and 10, for example, of [GVS99]. We are pointing out here, then, that it is virtually impossible to design a CRKS on the basis of simply writing down statements of relationship among chosen concept-names. We must be guided in our choice of statements at each stage by the developing diagram. The diagram can, for example, even persuade us to invent new concept-names in order to enforce, or to "smooth out", derivation paths or to provide extra relationships so as to ensure alternative derivation paths to support "weak" regions, or to temporarily avoid complex regions of a developing CRKS.

While computing can test, by limited access cascade and the use of gauges, a developing CRKS at each stage of development, and can assist in the production of a diagram at each stage, a computer cannot design an appropriate CRKS. Human designers are essential, and such designs by teachers will be subjective at least inasmuch as presentation strategies can vary from one teacher of that CRKS to another, and are also adjustable to suit a particular class of learners, or an individual learner. What computing can do as a basis is to store and

manipulate CRKS's by listing the statements of relationships and corresponding tuples, and to assist in the analysis of CRKS's and KHs, and thus of any EDUNET.

The presentation of a CRKS, in the steps revealed by a limited access cascade from the set of primaries of that CRKS, is a parallel mode presentation technique. It has been strongly suggested that the brain, as a learning machine, functions in a parallel mode. It is clear that limited access cascades provide a (formal) model for the implementation of this parallel mode of brain function.

4.3 Illustrative Applications

In this example, in terms of KH models, we start by assuming that the properties of addition of integers are discovered by induction from a number of examples such as the notion of a "number line" for instance, by the use of (partial) abstraction isomorphisms. Note 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-KHs of the KH 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 KHs, 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.

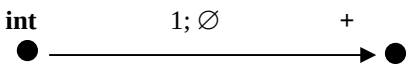
Definition 4.3.1: Given any cluster K, the ***cluster complexity*** of K is given by CCOM(K) = Σn_i where the sum is taken over all the n_i-tuples of K. Given a hypercluster M [K], the ***hypercluster complexity*** HCOM(M [K]) = $\Sigma \mid E_i \mid$ where the sum is taken over all the edges E_i of 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 **+.**

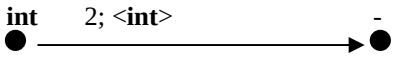
A cluster for 1 is



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.

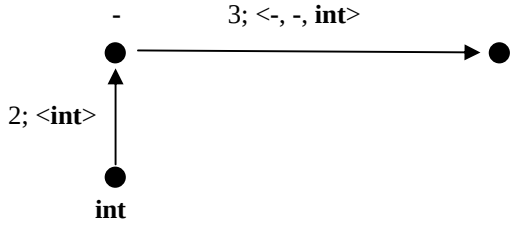
A cluster for 2 is



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.

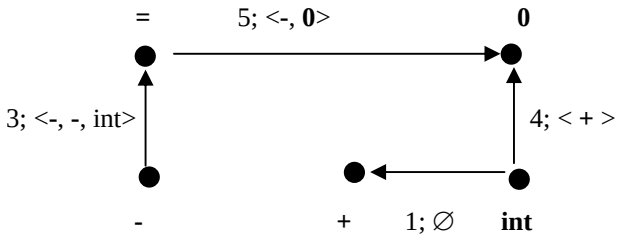
4. There is a special unique **integer**, for **+,** called zero and represented by the symbol **0**.

A cluster for 4 is

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

5. **=** holds between **- 0** and **0**.

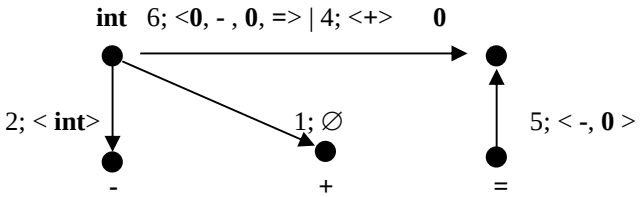
A cluster for 5 is



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**.

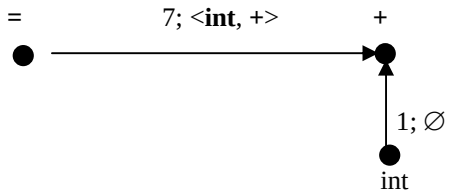
A cluster for 6 is



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 **+** y and y **+** x.

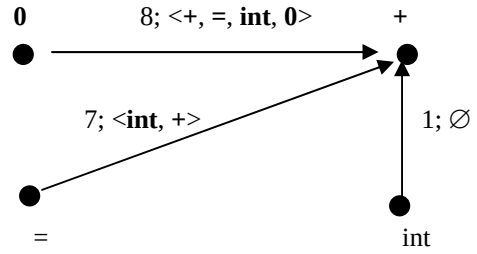
A cluster for 7 is



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

8. **0 +** x **=** x for every **integer** x with **0** under the operation **+.**

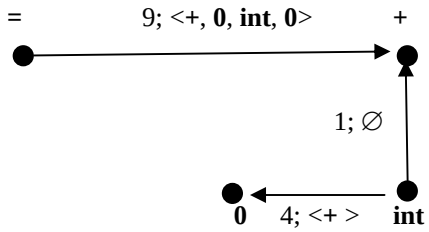
A cluster for 8 is



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

9. x **=** x **+ 0** for every **integer** x with **0** under the operation **+.**

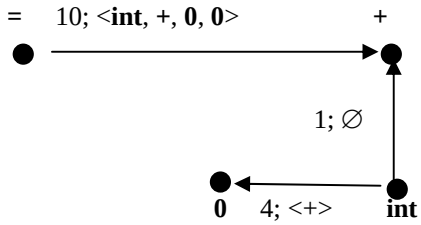
A cluster for 9 is



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 **+ 0** and **0 +** x, which conforms with statement 7.

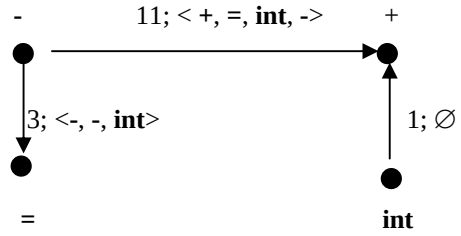
A cluster for 10 is



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

11. **-** x **+** x **=** zero for every **integer** x with **-** x under the operation **+.**

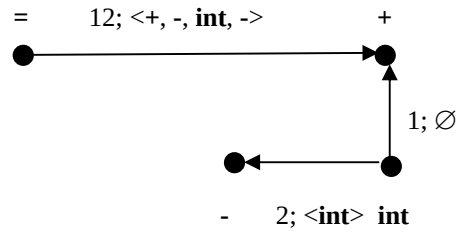
A cluster for 11 is



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

12. Zero **=** x **+** (**-** x) for every **integer** x with **-** x under the operation **+.**

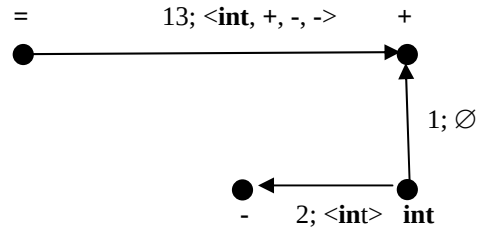
A cluster for 12 is



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

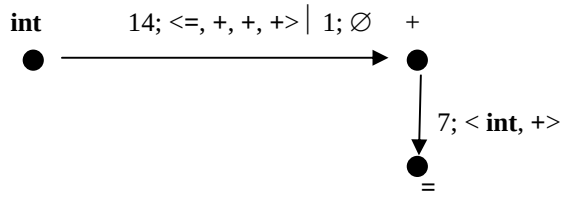
13. From statements 11 and 12 we have that **=** holds, for every **integer** x, between x **+ (-** x) and -x **+** x. This conforms with statement 7.

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 + (y + z) and (x + y) + 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, which displays all possible edge adjacencies between the cluster sets (the vertices), will be very complex. (The ***cluster set*** of a cluster is the set of all concept-names in that cluster.)

Next, we construct a CRKS 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. 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 4.3.1

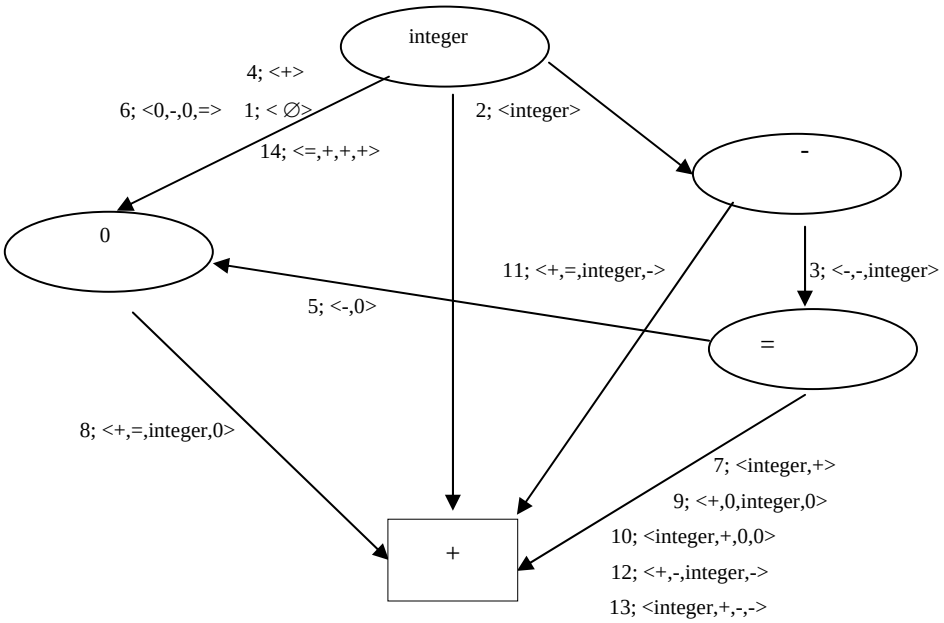
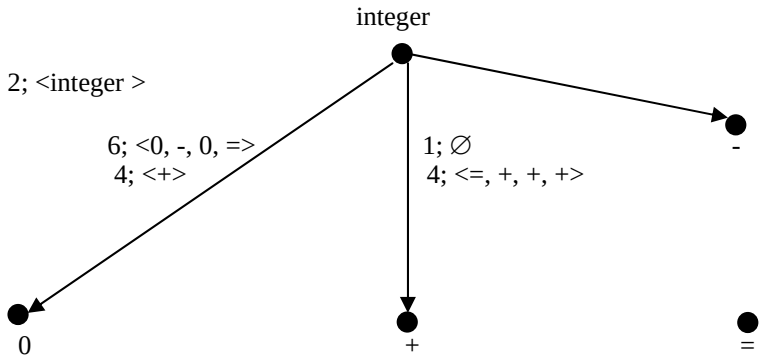


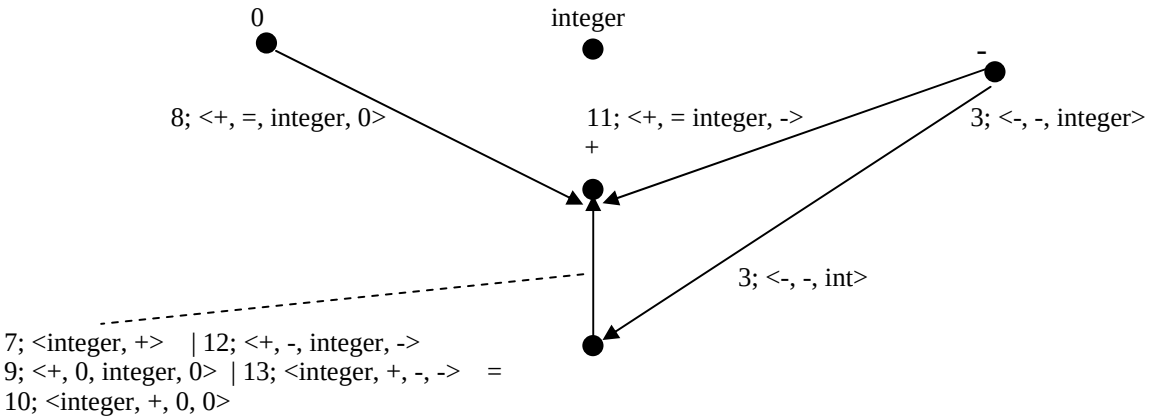
Figure 4.3.1 CRKS for the clusters.

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

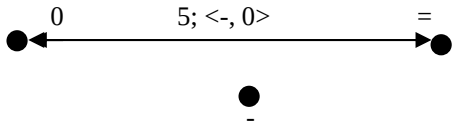
Step 1:



Step 2:

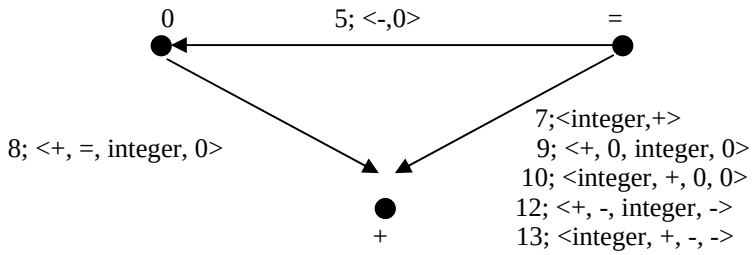


In the next step we "find" tuple 5, but no new vertices. After three steps the whole CRKS has been accessed. Suppose that after step 1 we decide to explore further only the concept-name "=". We start a new cascade with B^2_0 = {=}. T^2_0 = ∅, and for T^2_1 we have a choice of tuples that start with "=", i.e. tuples 5, 7, 9, 10, 12 and 13. If we choose only 5, then this step 2 yields:



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

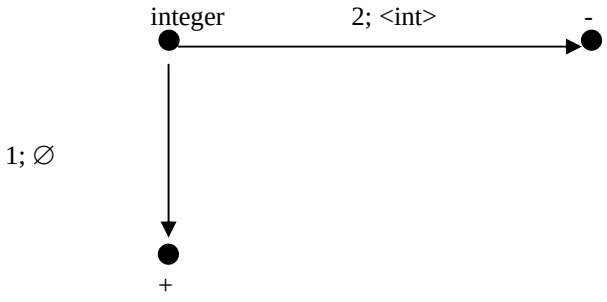
Step 3:



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.

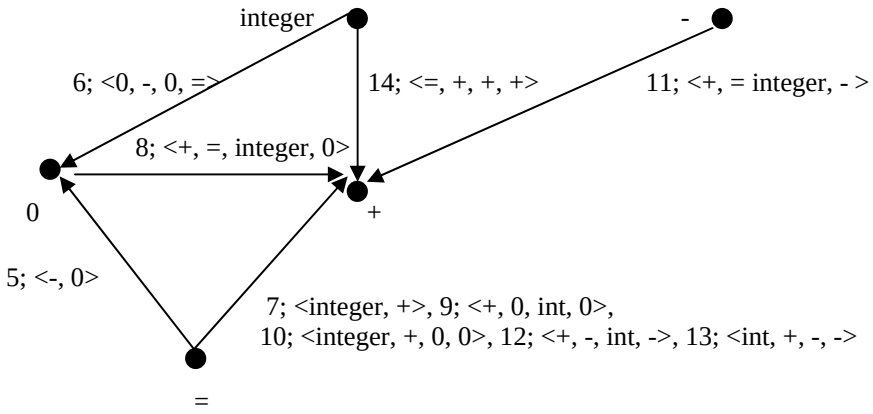
To illustrate our model of deductive reasoning in this CRKS we run a limited access cascade from its primary, i.e. B^1_0 = {integer}, in steps, showing what is newly derived in each step.

Step 1:

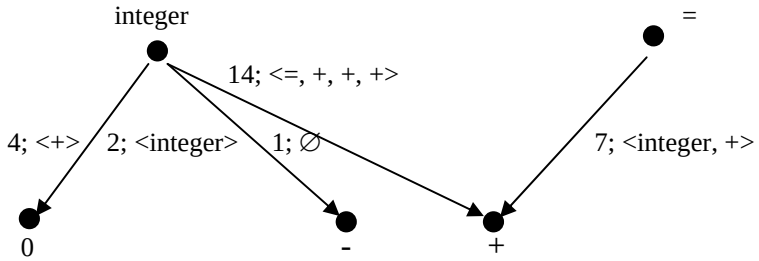


Step 2:

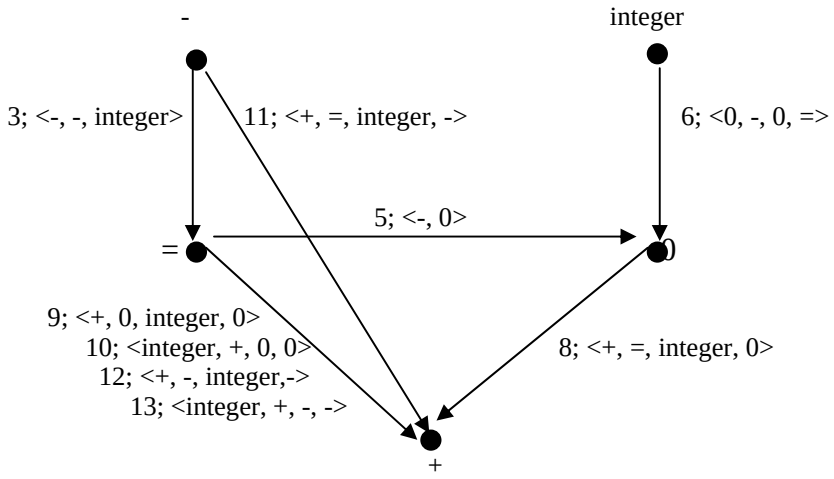
Step 3:



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 = {integer, +, =}. In the first step of this cascade we get

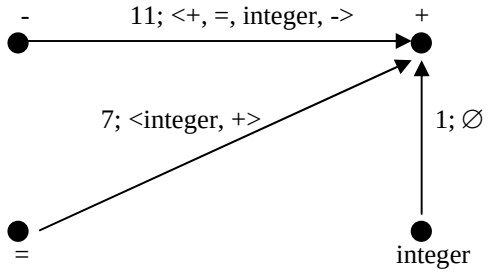


The next step, 2', of this second cascade yields



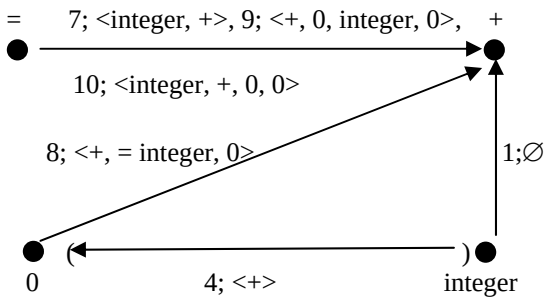
Joining $< B^1_1, T^1_1>$, $< B^2_1, T^2_1>$ and step 2' above yields the entire CRKS.

Next 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 <-, +, =, integer, -, +>, as shown below.

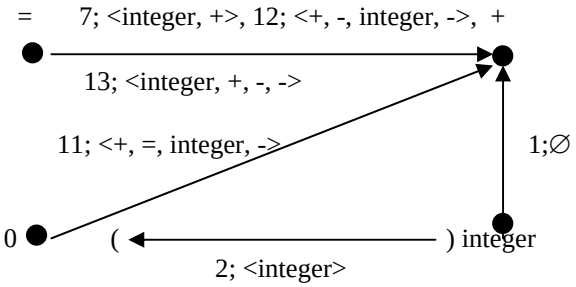


We have deleted 3 and added 7. This does not affect the construction of the CRKS from the clusters. This alternative cluster and that for statement 8 are isomorphic, where “-” and “0” are matched, so, for example we can use this structural analogy between the two clusters to teach/learn cluster 11 by 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, where 0 is identified with –, and we can go via the equivalent KHs. 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; <+> because 0 has been identified with -. In most cases isomorphic (sub-) CRKS's/KHs will share no vertices.

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.

Referring back to figure 4.3.1 we see a simple example of one presentation strategy type by spiralling to +. Thus, for example, we start with tuple 1 to introduce +. Then 4 introduces 0, 2 introduces -, followed by 3 to introduce =. Now we can use 14 to "support" +, and then 7.9.10.12 and 13. After this we can use 6, 11,5, and 8, completeing the CRKS for integer and +.

The context schema of + consists of all the vertices together with tuples 4, 14, 11, 8, 7, 9, 10, 12 and 13. The context schema of 0 is

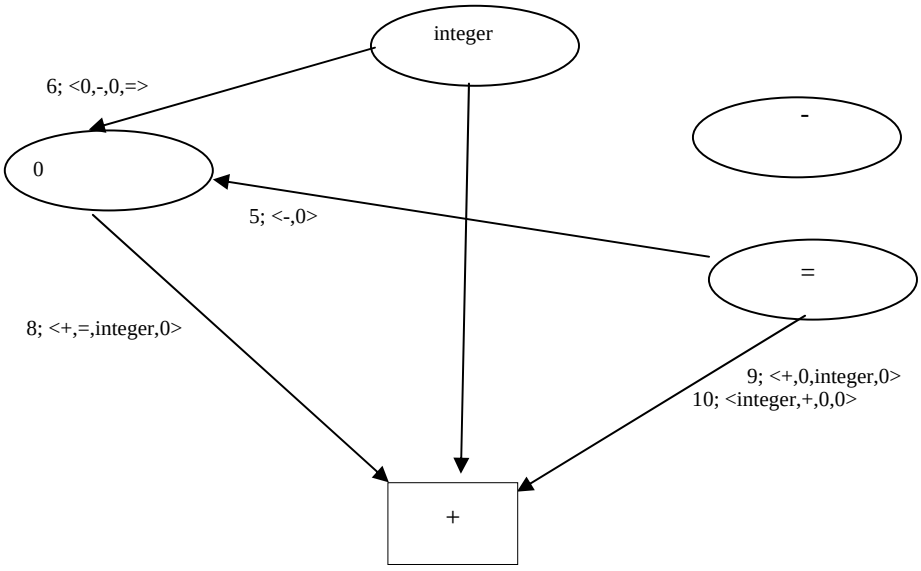


Figure 4.3.2. The context schema of 0.

We see that the vertex context number of 0 is 5, and its edge number is 5, regardless of whether we consider the context schema of 0 or the context hypernet of 0. As $|A|=5$ and $|T|=|E|=14$, 0 is quite "well integrated".

For an example of the construction of a path tree see section 5.4 of [GVS99].

Next 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 KH. Our illustration is the partial model of CRKS theory itself, as given in Appendix A of [GVS99], in the form of a CRKS.

The concept-names in the statements are those printed in bold. Here are the statements.

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. D**erivability** and **completeness** of a **formal schema** characterize 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. T**uples** in a **CRKS** are preserved by **CRKS isomorphism**.
27. **Isomorphism** of **CRKS's** expresses **structural analogy**.
28. S**tructural analogy** is expressed in terms of **isomorphic** (sub-) **CRKS's**.
29. I**somorphism** is used to express **structural analogy** among (sub-) **CRKS's**.
30. D**erivability** is realized in a **CRKS** by means of **derivation paths**.
31. D**erivation paths** express **derivability** in a **CRKS**.
32. D**erivability** 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.

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. <conceptmap, 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}

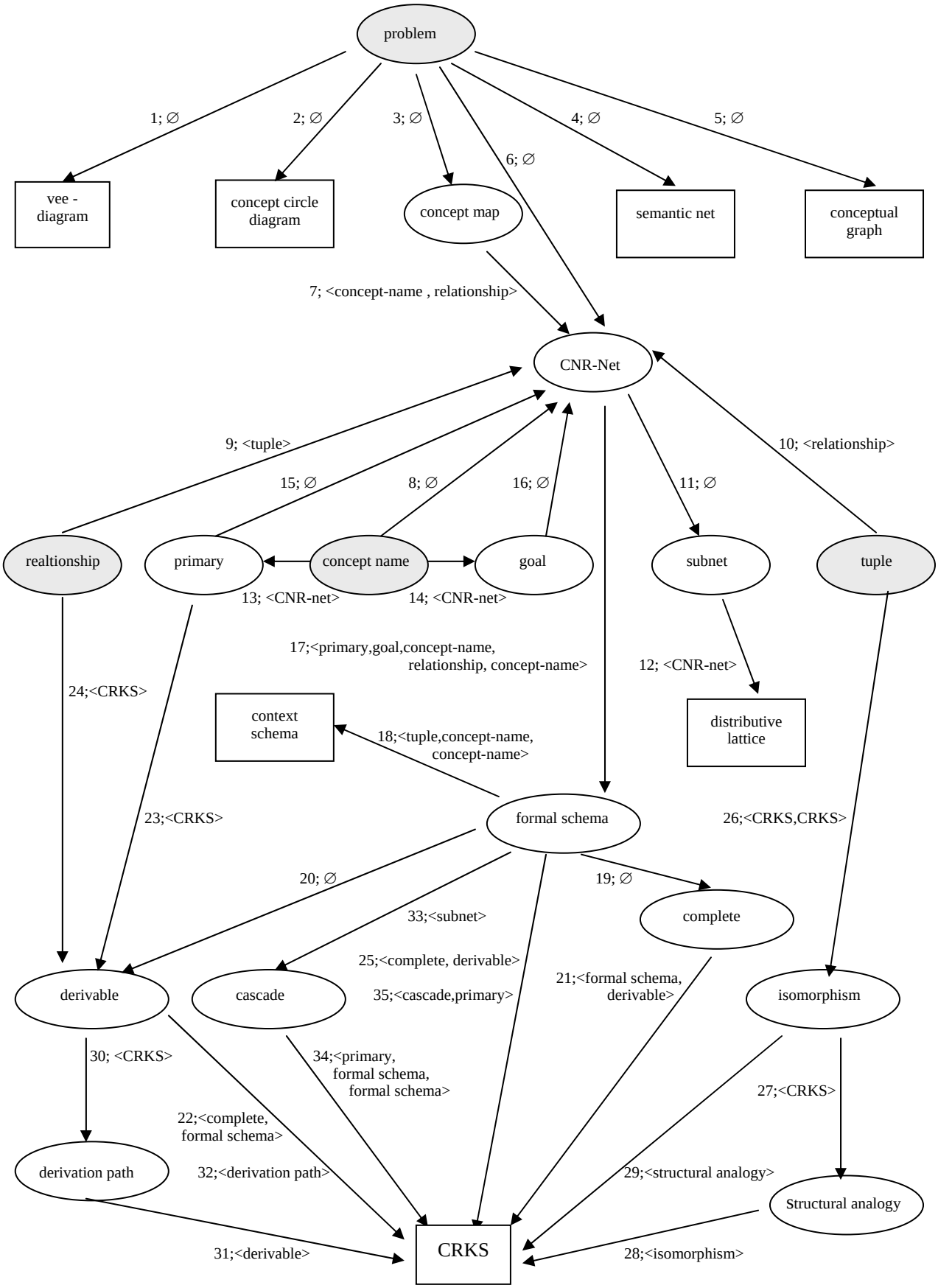
So far the difference is that the entries in the tuples are in a strict order, but those in the edges are unordered. Vertex adjacencies are preserved.

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}

The CRKS diagram is presented in figure 4.3.3. The abstracted KH is easy to construct from figure 4.3.3.

The concept-names involved in the CRKS can be translated to, or constructed in, another teaching meta-language, and from these we could build a KH that is isomorphic with the English language (in this case) KH represented by the second diagram. The new KH can now be interpreted as a CRKS in the “new” language in a number of ways, where we recall that if that “new” hypernet is a KH 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 KH 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, which will enforce redesign in order to preserve derivation paths.



Primaries are shaded. Rectangles represent goals.

Figure 4.3.3: A CRKS for the basis of CRKS's

Next we present some statements that constitute a partial model of KH theory and that constitute a CRKS representation of a small part of that theory.

Definition 4.3.2: By a ***formal hyperschema*** we mean a hypernet that is abstracted from a formal schema <A, T>.♦

A formal hyperschema <A,E> inherits the primaries and the goals of the formal schema <A,T> from which it is abstracted. <A, E> is such that, for every $E_i \in E$, $|E_i| \geq 2$, and every concept-name vertex $v \in A$ belongs to at least one $E_j \in E$, in <A, E>. Note that the "no-circuits" rule for <A, T> is not meaningful in <A, E>.

Here are the statements.
1. The abstraction of a **CRKS** is a **KH**.
2. The defining properties of a **CRKS** are inherited by the **KH** that is abstracted from it.
6'. The **problem** of devising a science of teaching has a potential solution in terms of **KHs.**
8'. **Concept-names** are represented by the vertices in a **KH**.
9'. **Relationships** are represented by the **edges** in an abstracted **KH**.
10'. **Edges** represent abstracted **relationships** in a **KH**.
11'. A **KH** has **sub-hypernets**.
12'. The set of all **sub-hypernets** of a **KH**, with meet and join defined on it, forms a **distributive lattice**.
17'. An abstracted **sub-hypernet** with at least one **primary**, at least one **goal**, and in which each c**oncept-name** is **related** to at least one other **concept-name**, is called a **formal hyperschema**.
18'. A **formal hyperschema** that consists of all the **edges** that involve a given **concept-name** constitutes, for that **concept-name**, its **context-hypernet**.
19'. A **formal hyperschema** in which every vertex has degree ≥ 1 is said to be **complete**.
20'. A **formal hyperschema** may have the property that every one of its vertices is **derivable**.
23'. A **primary** in a **KH** is trivially **derivable**.
24'. Every abstracted **relationship** in a **KH** is treated as an inference rule: This leads to the notion of **derivability**.
26'. **Vertex adjacencies** in a **KH** are preserved by **KH isomorphism**.
26a'. E**dges** in a **KH** are preserved by **KH isomorphism**.
27'. **Isomorphism** of **KHs** expresses **structural analogy**.
28'. **Structural analogy** is expressed in terms of **isomorphic** (sub-) **KHs**.
29'. **Isomorphism** is used to express **structural analogy** among (sub-) **KHs**.
30'. **Derivability** is realized in a **KH** by means of **derivation paths**.
31'. **Derivation paths** express **derivability** in a **KH**.
32'. **Derivability** is realized in terms of **derivation paths** in a **KH**.
33'. A **formal hyperschema** can be searched for relevant **sub-hypernets** using **cascades**.
34'. A **cascade** from the **primaries** of a **sub-hypernet** can be used to test a **formal hyperschema** for **KH** form.
35'. In a **formal hyperschema** we can use a **cascade** from the **primaries** to test for **KH** form.

Notice that the meaning of the term "structural analogy " applied to KHs is broader and more general than when applied to CRKS's.

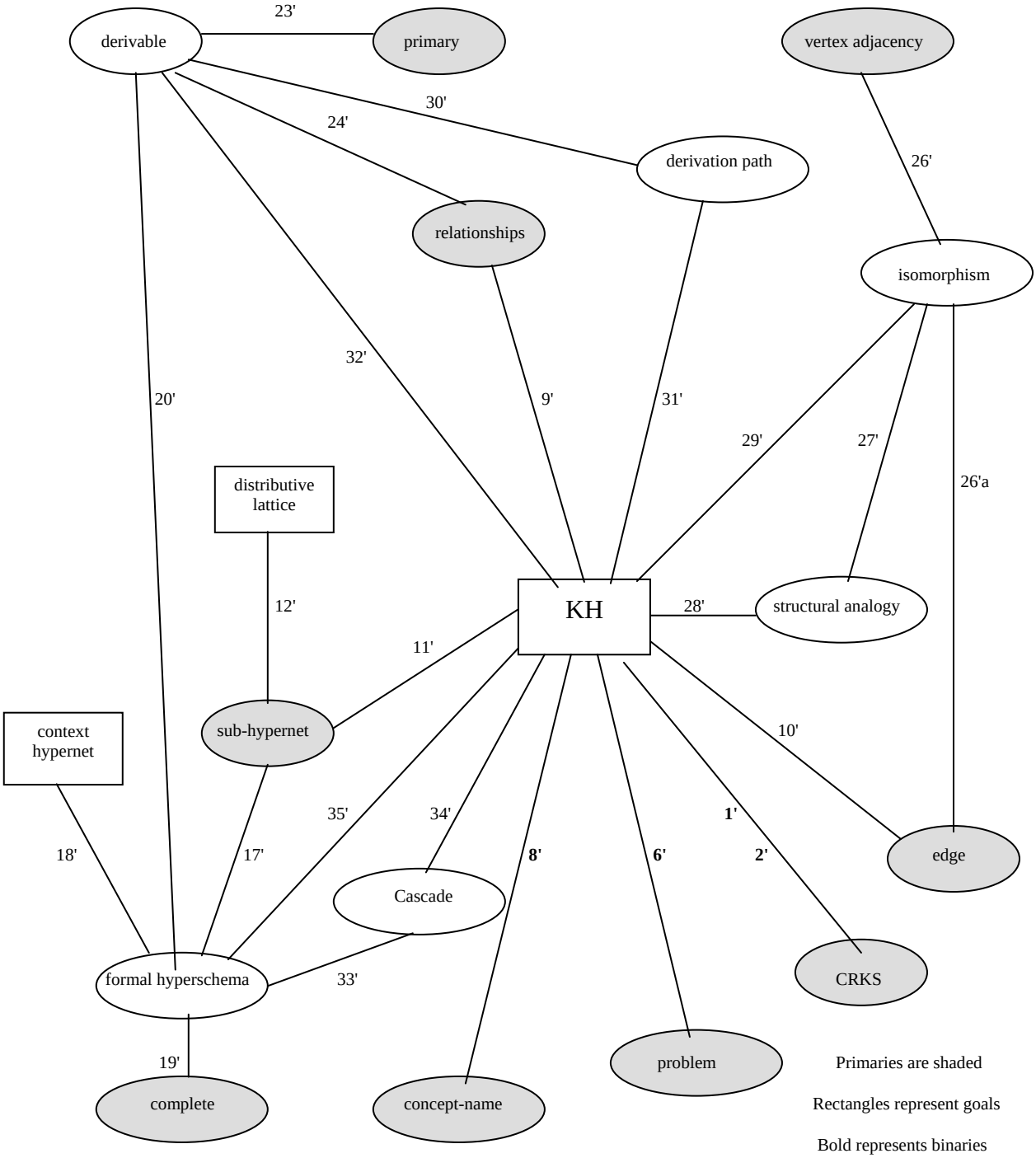


Figure 4.3.4 Diagram of a KH for some KH properties

The hypernet of figure 4.3.4 is indeed a KH, as can be verified by running a limited access cascade from its primaries, all of which are previously met concept-names. Some of the statements of relationship would have to be reworded to enforce derivability, primaries and goals of the CRKS from which the KH of figure 4.3.4 is abstracted. Notice that several more relationships could of course be added to the KH of figure 4.3.4. Such accommodations will generally require a change in the choice of primaries and goals in order to achieve assimilation, i.e. KH form.

4.4 Closing Comment

As we remarked in [GVS 99], the recursive definition of derivability is the key to all the properties and uses of a CRKS. The situation is directly analogous with formal deduction in mathematical logic, where the vertices represent well-formed formulas, the primaries represent instances of axioms, the relationships represent instances of rules of inference, and the derivation paths are equivalent to formal deductions. Just as axioms are a one line proof of themselves, primaries are trivially derivable by virtue of a derivation path of length zero. Every derived vertex is analogous with an axiom or a theorem. “Derivable“ implies a particular ordering of information, and we regard knowledge as “usefully ordered information”. This ordering appears to indicate that the information is learnable/teachable, so we take as a hypothesis that “derivable implies learnable/teachable”.

We have met two kinds of NET, relation nets and hypernets. As special cases we have CRKS’s and KHs and these two intimately linked systems constitute two different facets of the same structural model for acquisition, representation, retrieval, accommodation and assimilation, communication and management of knowledge. Indeed, the very definition of “knowledge” in this work is that it is data and information that fit the CRKS model. Briefly, we have the following synopsis:

- Knowledge acquisition/ learning consists of devising concept-names and discovering relationships among them, i.e. “perceiving” information items.

- Knowledge representation consists of constructing CRKS’s.

- Knowledge communication/teaching involves various presentation strategies for CRKS’s – see also [GVS99].

- Knowledge management entails the design, selection, retrieval, deletion, assimilation, manipulation and analysis of (sub-)CRKS’s using the techniques presented in this work and in [GVS99].

Appendix

A1. List of Constructional Schemes

1.2.1: To determine id(A_m), od(A_m) and d(A_m) for $A_m \in A$ in a relation net <A, T>.

1.2.2: To determine R(A_m), R[A_m], and hence |R(A_m)| and |R(A_m)|, for $A_m \in A$ in a relation net <A, T>, where |X| denotes the number of members of set X.

1.2.3: To check that <B,U> $\angle$ <A,T> for two given relation nets <A, T> and <B,U>.

1.2.4: To find the isolates and the complete isolates in a relation net <A, T>.

1.2.5: To check that relation net <B, U> is a spanning subnet of a relation net <A, T>.

1.2.6: To find the maximum subnet <B,T↑B> of a relation net <A, T>, where $B \subseteq A$.

1.2.7: To find all the $A_n \in A$ that are adjacent from $A_m \in A$ in a relation net <A, T>.

1.2.8: Find all the vertices that are vertex between A_r and A_s on a given path $A_r \rightarrow A_s$ in a relation net <A, T>.

1.2.9: Find all the vertices that are reachable from a given $A_r \in A$ in a relation net <A, T>.

1.2.10: Find the join of two relation nets <B, F> and <C, G>.

1.2.11: Find the meet of two relation nets <B, F> and <C, G>.

1.2.12: To "run" a fast access cascade from a given $B_0 \subseteq A$ in a relation net <A, T>.

1.2.13: To "run" a limited access cascade from a given $B_0 \subseteq A$ in a relation net <A, T>.

1.2.14: Find the context-net <A, T>[A_k] in a relation net <A, T>.

1.3.1: To find the primaries of a potential formal schema <A, T>.

1.3.2: To find the goals of a potential formal schema <A,T>.

1.3.3: To check that each $A_k \in A$ in a potential formal schema <A, T> is related to at least one $A_j \in A$, $A_j \neq A_k$, in <A, T>.

1.3.4: To determine whether or not there are circuits of any length in a relation net <A, T>.

1.3.5: To check that a given formal schema <A, T> is complete.

1.3.6: To find the context schema of a given vertex $A_k \in A$ in a formal schema <A, T>.

1.3.7: To construct a path tree, for a formal schema <A, T>, that displays and distinguishes every path from each primary in <A, T>.

1.3.8: To find all the paths of length ≥ 1 from $A_m \in A$ to $A_n \in A$ in a formal schema <A,T>.

1.3.9: Test to see if a formal schema <A, T> is connected.

1.3.10: To test a complete formal schema <A, T> for CRKS form.

1.5.1: Given two hypernets <A_1, E_1> and <A_2, E_2>, find an isomorphism between them.

2.3.1: To find an edge basis for a hypernet <A, E>.

2.6.1: To find an edge cut-set included in R where $R \subseteq E$ is any disconnecting set of edges of a connected hypernet <A, E>.

3.1.1: To construct a derivation path tree, for a KH <A, E>, displaying and distinguishing every derivation path from each primary of <A, E>.

3.1.2: Find all the derivation paths between vertex u and vertex v in a KH <A, E>.

3.6.1: To find a set of quasi-disjoint $\pi - \gamma$ paths in Z.

3.6.2: To find a set of quasi-edge-disjoint $\pi - \gamma$ paths in Z.

A2. Examples of CRKS's

All references in this section are to [GVS99]. They each consist of a section heading and page numbers.

1. Set, member , equal, empty set and some relationships among them. 2.2 pgs 29 to 35. (Corrections: pg 32 (+14) $\in$ in bold, 3 times, (+15) $\in$ in bold, pg 34 (-10) is figure 2.7 $\rightarrow$ is presented in figure 2.7)
2. The rotation symmetry group of the square. 5. 4 pages 75 to 78.
3. The path tree for 2. 5.4 pgs 80 to 86. (Corrections: pg 86 (lower diagram) insert arrow head at 0 and at fol–by, (-3) notation $\rightarrow$ notion.)
4. A formal language. 6.2 pgs 94 to 97.
5. The displacements in a plane. 7.2 pgs 103 to 105.
6. The triples: An example of isomorphism. 7.3 pgs 106 to 110. (Corrections: pg 106 (+10) triple–sum $\rightarrow$ bold).
7. Commutative group. 7.4 pgs 110 to 112.
8. CRKS's for theorem proofs. 7.4 pgs 112 to 118. (Corrections: pg 115 (diagram) 18 $\rightarrow$ 19, and insert arrow from g_k´ to uniqueness, label 18; <=,g_k, inv>; pg 117 (diagram) 21 $\rightarrow$ 22, and insert arrow from g_2 to cancellation law, with label 21; <=, g_1>).
9. A simple programming language. 11.1 pgs 173 to 181. (Corrections: pg 179 (+6) below$\rightarrow$ above.
10. Derivation path families. 11.2 pgs 182 to 187 (Corrections: pg 183 (+5) delete "indented,", pg 184 (-4) aexp$\rightarrow$sta

Note: Theorem 4.1 on page 55 of [GVS99] is incompletely stated. The corrected version is given as Theorem 1.3.2 on page 18 of this report.

References

Main references

[GVS99] **Geldenhuys, A. E., Van Rooyen, H. O., and Stetter, F**., 1999. *Knowledge Representation and Relation Nets*. Kluwer Academic Publishers, Boston.

References to Basics

[ANH78] **Ausubel, D.P., Novak, J.D., and Hanesian, H.,** 1978. *Educational psychology: a cognitive view*. NY: HoIt, Reinhart and Winston, second edition.

[Aus63] **Ausubel, D.P**., 1963. *The psychology of meaningful verbal learning.* NY: Grune and Stratton.

[Aus80] **Ausubel, D.P**., 1980. *Schemata, cognitive structure and advance organizers: A reply to Anderson, Spiro and Anderson*. American Educational Research Journal, 17(3): 400-404.

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

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

[Gel93] **Geldenhuys, A.E**., 1993. *Concept-Relationship Knowledge Structures: Computer supported applications in teaching and learning.* PhD Thesis, Unisa.

[Hes79] **Hestenes, D**., 1979. *Wherefore a science of teaching?* The Physics Teacher, April 1979: 235 - 242.

[HNC65] **Harary, F., Norman, R.Z., and Cartwright, D**., 1965. *Structural models: An introduction to the theory of directd graphs*. John Wiley.

[Len80] **Lendaris, G.G**., 1980. *Structural modelling – a tutorial guide;* IEEE Transactions of Systems, Man and Cybernetics, December 1980

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

[VR76] **Van Rooyen, H.O.**, 1976. *Binary networks in graph theory*. PhD Thesis, Unisa.

[VSG01] **Van Rooyen, H. O., Stetter, F**., and **Geldenhuys, A. E**., 2001. *Relation nets and hypernets*. Technical Report TR-01-020, Department for Mathematics and Computer Science, University of Mannheim, 2001.

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

[Wol82] **Wolvaardt, D.E.,** 1982: *Development of nebula theory with applications to syllabus databases.* PhD.Thesis, Unisa.

Index

Numbering: chapter/page, i.e. 1/33 means page 33 in chapter 1