SCS&E Report 9311
November, 1993

Conceptual Graphs for Natural Language Representation

Graham A. Mann

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
THE UNIVERSITY OF NEW SOUTH WALES

Abstract

Conceptual graphs are abstract data structures that can form the basis of a knowledge representation system. Since their introduction by Sowa in 1984, they have formed the core of a flourishing research effort, with applications in databases and artificial intelligence. The basics of conceptual graph theory are outlined, including the organisation of existential, finite, bipartite directed graphs and their relationship to first order predicate logic. A definition of the basic functions defined over such graphs follows, including the canonical formation rules and some higher order functions intended to support reasoning. With carefully designed semantic knowledge in the form of catalogues of concepts and relations, a conceptual graph processing system can support knowledge representation. A new evaluation of the strengths and weaknesses of conceptual graphs with respect to a theoretical view of knowledge representation which enumerates five roles which the representation must play: provision of a theory of intelligent reasoning, ontological commitment, object surrogacy, provision of a practical computing medium, and human read/writeability, is made. Finally, existing methods of using the conceptual graphs for natural language comprehension are discussed, and a new theory of conceptual assembly is proposed. Access to an experimental conceptual graph processor written in Common Lisp is offered in an Appendix A.

1. What are Conceptual Graphs?

Conceptual graphs are a knowledge representation formalism invented by John Sowa and first described in his book 'Conceptual Structures' (Sowa, 1984). Sowa was dissatisfied by the shortcomings of the representation techniques of the day: he was particularly aware of the problems involved in trying to use data structures to model commonsense notions such as those required in natural language utterances, including nested contexts, control of scope and types of quantifiers, and problems of individual reference, such as representing definite articular, generic and particular concepts, sets, and anaphoric or corefential links between concepts. Existing approaches to this problem had tended to come to grief either a) because they were not expressive or flexible enough to easily capture some of the things humans commonly say (as with attempts to write first order predicate logic clauses corresponding to sentences (Genesereth & Nilsson 1987)) or b) because purpose-built conceptual systems usually failed to have sufficient clarity, consistency and/or extendability to be useful as dynamic representations (even the best efforts, among them Schank's Conceptual Dependency Theory (Schank, 1972; Schank, 1975), were eventually understood to be inadequate). Sowa wanted a system of knowledge representation that enjoyed both the conciseness and computational power of logic and expressive flexibility of a conceptual network of semantic types, without the disadvantages of either.

Drawing inspiration from the existential logic graphs of C.S. Peirce (Burks, 1960) and on existing semantic network theory (Masterman, 1960; Findler, 1979), Sowa created an entire conceptual representation system, describing it complete with philosophical and psychological motivations, a set of formal definitions of the basic components, a mathematical defence consisting of definitions and proofs showing a system of logical semantic operations (the four canonical formation rules, and some more complex operations), a guide to using the representations for formal reasoning, and practical examples of their use for dealing with natural language and other kinds of knowledge.

Practically speaking, conceptual graphs are an abstraction which is implemented by individual researchers in different ways, usually according to the computer language being used. The basic encoding of the graphs as data structures, together with code implementing at least the canonical formation rules, together with support for creating, destroying and displaying graphs, forms a conceptual graph platform, can be a basis for developing knowledge-based systems. This report offers access to an experimental conceptual graph code base written in Common LISP for Macintosh computers, together with some notes about how it can be used (see Appendix A).

Since the publication of 'Conceptual Structures', conceptual graphs have formed the basis of dozens of knowledge systems including document retrieval systems (e.g. Gardiner & Slagle, 1991; Dick, 1991), expert systems (Venk & Govind, 1990) and a medical advisor (Schroder, 1992) as well as natural language experiments (Pazienza & Verlardi, 1988; Bornerand & Sabah, 1990). A thriving international research effort has continued exploring and extending the formalism (see the recent collection of papers in Nagle, Nagle, Gerholz and Ekland, 1992). A workshop gathers annually to discuss developments in the field. Recently there has been a substantial move in the direction of certifying the formalism as an international standard, and the development of a standard, high accessibility, general purpose workbench to facilitate the building of conceptual graph based systems called PEIRCE, is under way (Ellis & Levinson, 1992)[1].

2. Fundamentals of Conceptual Graph Theory.

At its most basic, a conceptual graph asserts the existence of a particular set of ontological objects called *concepts* together with a set of relationships among those concepts called *relations.* Formally, this assertion is represented by a finite, bipartite, possibly cyclic, directed graph, in which the concepts and relations form two types of nodes, and in which connections between the nodes are represented by identical single-pointed arrows. More sophisticated than a simple semantic network, a conceptual graph depends for its meaning on connections to a set of knowledge resources and inferential operations. Fundamental to these are a hierarchy of conceptual types and a (possibly structured) collection of relational types, from which are chosen the possible nodes of a conceptual graph. A set of basic canonical operations enforce and execute the rules of conceptual graph evolution, allowing the creation of new graphs from old graphs.

A concept is a typed instance of some event, idea or object in the world. A concept in a conceptual graph is a particular example of the entire class, or type, of that object. The difference between a type class and a concept of that type in a conceptual graph is the type-token distinction familiar to philosophers. In the standard display form[2] for conceptual graphs, a concept is represented as a box containing a *type label*: - a single capitalised word identifying the type of this concept (Fig. 1).

The box may also contain a *referent.* This is a string of symbols that uniquely identifies this particular instance of the type. Concepts without referents are called *generic types*. Entries of the referent field can be complex, but in its simplest form, it consists of either a literal string representing a proper name such as "Mary" or in the case of types that are not usually distinguished with proper names, a number like #164, which may be understood as the one hundred and sixty fourth example of this type that the system has encountered. A # symbol without a number should be understood as a definite article e.g. [CAT:#] can be expressed in English as "the cat" (c.f. [CAT], which is only the indefinite "a cat"). In the display and linear forms, the type field and the referent fields of concepts are delimited by a single semicolon. Special syntax to support variables, quantities, sets, and coreference to other concepts are discussed in Sections 3.3 and 3.7 of Sowa (1984).

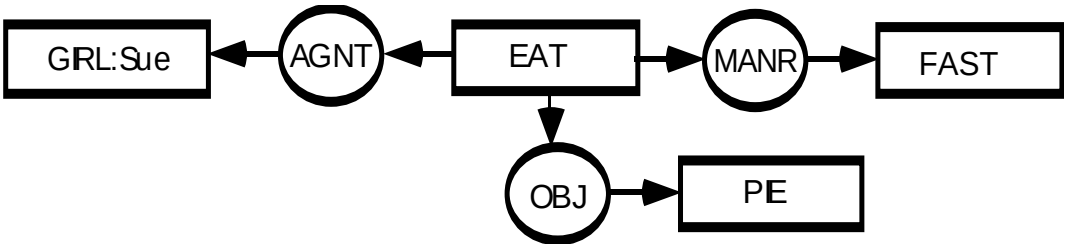

Figure 1. Conceptual graph representing a particular commonsense notion: "Sue is quickly eating a pie." Expressions using dyadic relations should be understood as follows: [CON1] -> (REL) -> [CON2] is to be read as the REL of a CON1 is CON2. Thus pie is the object of eat. Note the readability of the graph.

A relation is a generic form of a particular kind of relationship into which one or more concepts may enter. Monadic relations attach to only one concept. Examples of this kind of relation are tense markers such as (PAST), negation (NOT), and modal

operators such as possible (PSBL). Most relations are dyadic i.e. they connect two concepts. Examples of common dyadic relations are agent (AGNT), location (LOC) and instrument (INSTR). A few triadic relations exist, but the only common one is between (BETW).

In a standard semantic network, instance nodes may be explicitly linked to their type nodes by means of an IS-A relation. Much has been written about the exact meaning of IS-A relations (e.g. Brachman, 1983). Suffice it to say that IS-A links cannot be used in this way in conceptual graphs, because the relationship between a type and one of its instances is of a higher order than relations between one instance and another. Confusing these orders can lead to logical errors. Thus the type relation, as specified in Assumption 3.2.1, should not be confused with IS-A. Using type labels inside concept nodes enables a clear notation to describe a particular state of affairs without cluttering up the diagram with unnecessary semantic information. Sematic information is supplied in a hierarchy of conceptual types. Although no such ordering existed among the relation types in Sowa (1984), some have since advocated this (e.g. Pazienza & Velardi, 1987).

This hiding of higher order type relations also helps to make clearer the correct interpretation of a graph, as that of a particular episode, event or state of affairs, rather than some general law about relationships which hold among types. It is possible to represent general laws using conceptual graphs, but this matter will be addressed later. For now, consider the existence of the logic mapping operator ϕ which guarantees that a first order predicate formula can be derived from any conceptual graph. Assumption 3.3.2 in Sowa (1984) shows how this may be done for a graph *u*: :

- If *u* contains *k* generic concepts, assign a distinct variable symbol $x_1, x_2, \ldots x_k$ to each one.
- For each concept *c* of *u*, let *identifier(c)* be the variable assigned to *c* if *c* is generic or *referent(c)* if *c* is an individual.
- Represent each concept *c* as a monadic predicate whose name is the same as *type(c)* and whose argument is *identifier(c)* .
- Represent each *n*-adic conceptual relation *r* of *u* as an *n*-adic predicate whose name is the same as *type(r)*. For each *i* from 1 to *n*, let the *i* th argument of the predicate be the identifier of the concept linked to the *i* th arc of *r*..
- Then ϕu has a quantifier prefix $\exists x_1\ \exists x_2 \ldots \exists x_k$ consisting of the conjunction of all the predicates for the concepts and conceptual relations of *u*..

Using this method, the first order predicate formula for the graph in Figure 1 becomes

$$\exists x \exists y \exists z\ (\text{GIRL(Sue)} \wedge \text{AGNT (Sue,x)} \wedge \text{EAT(x)} \wedge \text{OBJ (x,y)} \wedge \text{PIE (y)} \wedge \text{MANR (x,z)} \wedge \text{FAST (z)}).$$

The scopes of the existential variables quantified by $\exists$ in the equation extend over the entire graph. Other quantifiers (e.g.universal or negation) may apply over the same or different parts of a graph. The scope of such quantifiers is specified by means of a sheet of assertion, which in conceptual graph theory means a context. A context enables conceptual graphs to be nested, making possible more complex assertions and making the representation of natural language easier. An example is the use of a monadic relation of time to act as a tense marker in a sentence. Since

everything that happened in the event “John went to London.” occurred in the context of time past, it would not be enough to link the monadic relation (PAST) to the concept representing the act of going. Instead, the entire assertion of “John goes to London” is enclosed in a context box, and that context box is then linked to (PAST):

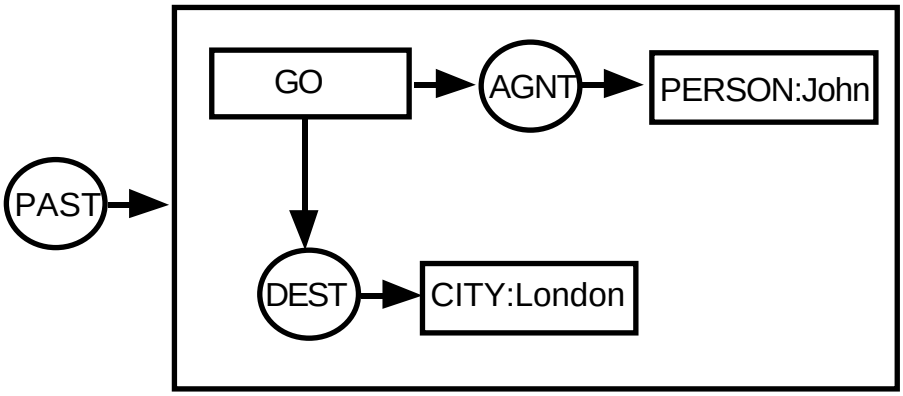

Figure 2. Conceptual graph representing the English sentence: “John went to London”. The monadic relation (PAST) dominates the entire assertion. Whatever implications follow from the idea that the action has already occurred apply to everything inside the context box.

If conceptual relations can be linked to contexts in this way, it follows that contexts can be nested within other contexts to an arbitrary depth. Nested contexts can be used to represent propositions, situations and beliefs. The inclusion of contexts in conceptual graph theory also allows for a variety of modal logic assertions, such as possibility and necessity, to be handled.

If conceptual graphs are able to be mapped into FOPC in this way, and it is the computational power of the logic we need, why use the graph notation at all? The answer is that it is much easier to read and write conceptual graphs than the corresponding logic form, particularly when the assertions become complex. Also, many of the operations on conceptual graphs (see Section 3) are much more readily understandable if they can be visualised as two-dimensional alterations to graph diagrams. In practical knowledge-based computing systems, particularly those involving the writing of many complex assertions, this fact becomes very important.

3. Operations on Conceptual Graphs.

3.1 Basic Support Functions

An elementary computer implementation of a conceptual graph system must begin with a method of representing conceptual graphs. Primitives of this nature are strongly dependent of the equipment and language being used, but all would include functions which can create, store and destroy the basic elements: concepts, relations

and contexts. Methods of representation vary widely. In the Common LISP implementation referred to in Appendix A, conceptual graphs are represented as lists of elements, each of which is the name of an association-list storing a set of entries needed to specify an element of that type. Other methods may be used in Prolog implementations, such as the use of tuples. Here I only assume the existence of functions that create, store and destroy conceptual graphs.

A practical conceptual graph system also needs some way of converting graphs between the display and/or linear form and the internal representation form. The methods by which this is done are highly implementation-dependent and need not concern us here. Suffice it to say that mapping a conceptual graph expressed in graphical display form into an internal representation and back again requires substantial programming effort.

The ability to quickly access the type fields of relations and the type and referent fields of concepts and contexts is clearly needed. Functions that do this, such as *type(c)* and *referent(c)*, were mentioned in the predicate logic mapping method described in Section 2. Also needed are functions such as *supertype(x,y)* and *subtype(x,y)* which check the type hierarchy for relationships between concepts.

An elementary function the conformity relation ::, which checks that the referent field in a concept matches the type field. GIRL::Sue means that it is true that the individual Sue is of type GIRL. This implies a mechanism that keeps track of which referents were originally assigned to which types. Note that a referent will conform to all supertypes of its original types. For example, the assertions PERSON::Sue, ANIMATE::Sue and THING::Sue are all true. The conformity operator is needed during certain operations which try to alter the type and referent fields of concepts, to make sure that invalid concepts like [GIRL:Spot] do not come into existence. It helps to enforce the semantics of graphs during transformation operations.

3.2 Canonical Formation Rules

Any collection of concepts and nodes connected by arcs in the proper way is a conceptual graph. But not all such graphs make sense. Graphs which make plausible assertions about the world, that is, graphs which represent clusters of relationships between concepts which we recognise as statements about the real world, are called *canonical* graphs. As an intelligent system gains experience about the world, it builds up a collection of canonical graphs. New graphs can become canonical graphs in three ways: *perception*, in which a canonical graph assembled to match a sensory icon from the outside world; *formation rules*, in which new canonical graphs are derived from existing canonical graphs by means of the *restrict*, *join* , *simplify* and *copy* rules; and *insight*, in which arbitrary conceptual graphs are designated as canonical, in respect of the creative process by which new ideas can form

Assumption 3.4.3 of Sowa (1984) states that the four canonical formation rules by which a conceptual graph *w* may be derived from conceptual graphs *u* and *v* (where *u* and *v* may be the same graph) :

Restrict

For any concept *c* in *u*, *type(c)* may be replaced by a subtype; if *c* is generic, its referent may be changed to an individual marker. These changes are permitted only if *referent(c)* conforms to *type(c)* before and after the change.

Figure 3. illustrates the effect of the restrict rule.

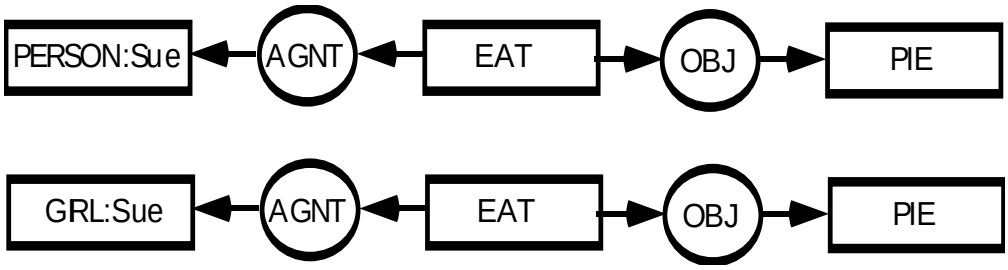

Figure 3. The concept [PERSON:Sue] in the top graph may be restricted to the concept [GIRL:Sue] because the type GIRL is a subtype of PERSON and the referent Sue is a girl. The result is the lower graph.

Join

If a concept *c* in *u* is identical to a concept *d* in *v* then let *w* be the graph obtained by deleting *d* and linking to *c* all arcs of conceptual relations that had been linked to *d*.

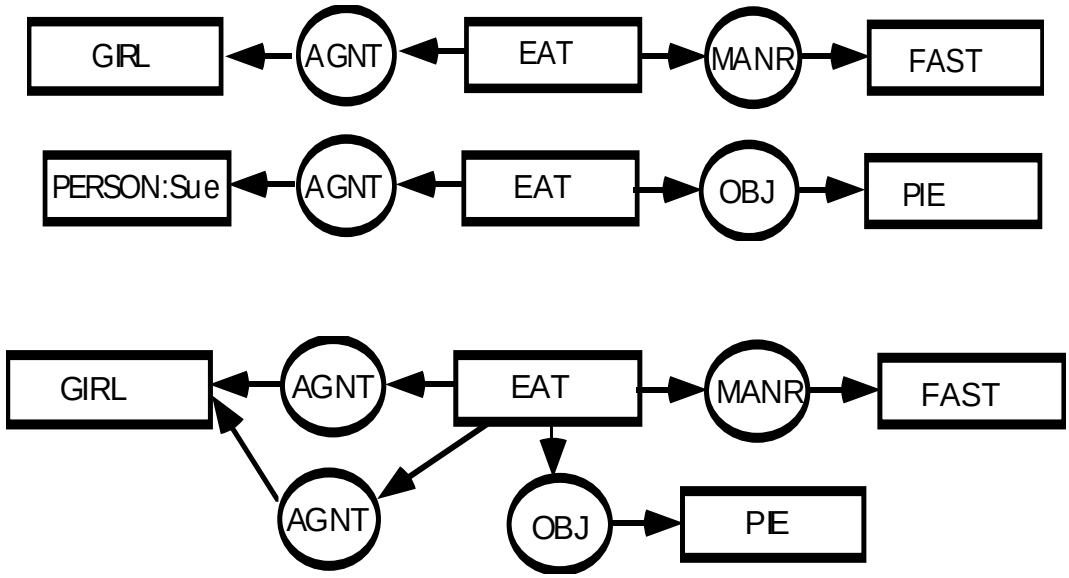

Figure 4. A join operation on the top two graphs results in the bottom graph. Note the extra (AGNT) relation in the final graph. Such redundant relations are normally removed using the simplify rule (see below).

Simplify

If conceptual relations r and s in the graph u are duplicates, then one of them may be deleted from u together with all its arcs. To be duplicates, the relation nodes must be linked to the same concepts.

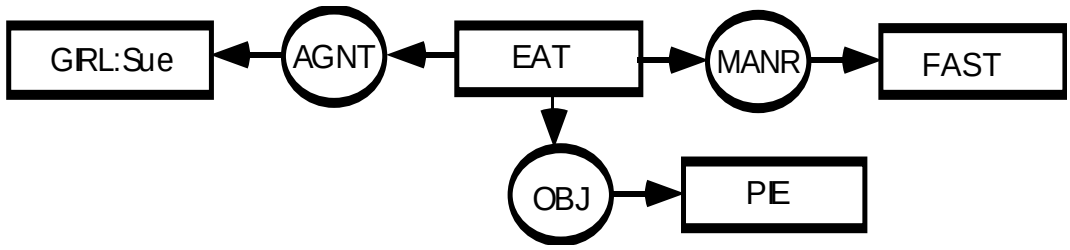

Figure 5. Simplify operation on the bottom graph of Figure 4 results in a graph with no redundant relations.

Copy

A new graph w is formed, which is an exact copy of u.. This means that another representation of the old graph is formed, which happens to have exactly the same concepts and relations, and structure connecting them. At the display level the graphs look identical (though the graphs must be distinguished at some level in the implementation).

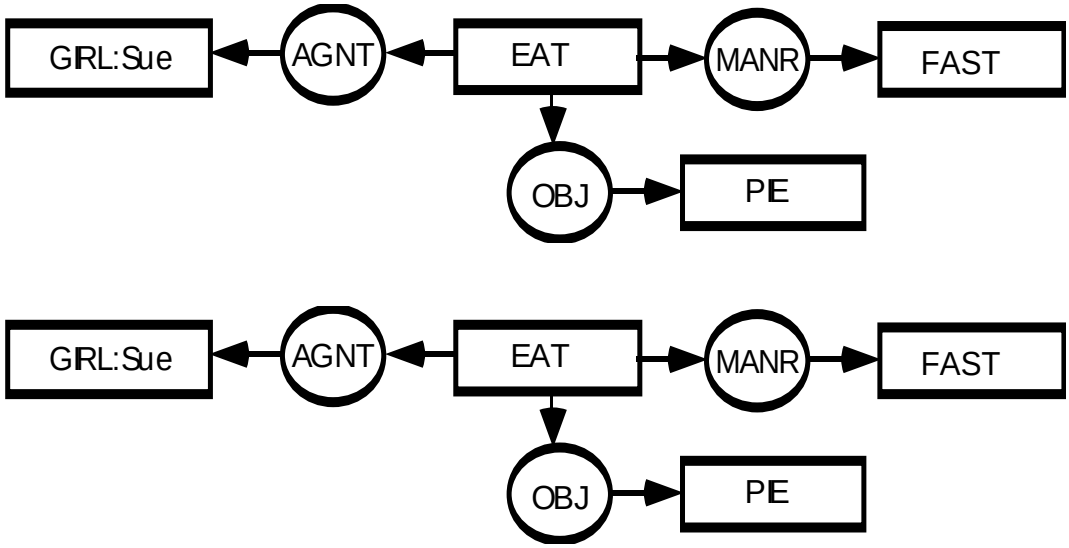

Figure 6. An exact copy of the graph in Figure 5 is made. Note that the display forms are identical, but the tokens implementing the two graphs are different.

3.3 Higher Level Operations.

The canonical formation rules of Section 3.2 specialise graphs. A number of generalization rules are also mentioned in Sowa (1984). The generalisation of a graph implies the removal of some parts of a graph or the replacement of some of the type

labels in concept-nodes with more general forms on the type hierachy. Referent fields may also be replaced with more general forms. Generalisation rules preseve the truth of a graph, but not selectional constraints. An important generalisation rule is projection.

Projection

If graph *u* is a specialisation of graph *v*, then there must be a subgraph *u'* embedded within *u* that represents the original *v* to which additional graphs were somehow joined during canonical formation. The subgraph *u'* is called the projection π of *v* in *u*.. Every relation in πv must be identical to its correspondent in *v*, but some of the concepts may have been restricted to supertypes or may have been converted from generic to individual. Since some of the concepts might have been removed as duplicates or joined to each other, the shape and concepts of the projection may be different. The mapping is not necessarily one-to-one, nor unique.

The operator π is proved to have these properties in Sowa (1984). Projection is useful for cognitive operations in which some probe graph is to be matched to a set of target graphs, for example when trying to identify a graph representing an appropriate word-sense in a given context (see Nogier & Zock, 1992). An effective projection algorithm is described by Lendaris (1992), a version of which is used in the conceptual graph processor described in Appendix A.

Lambda Abstractions

It is possible to use conceptual graphs as frames and to instantiate the values of variables into them by using graphs in *lambda abstractions*.. An n-adic lambda abstraction consists of a canonical graph *u* (the body) and a list of generic concepts $a_1,..,a_n$ in *u* (the list of *formal parameters)*. The parameter list distinguishes the formal parameters from other concepts in u. In effect, concepts that act as formal parameters have variables for referents. An instantiation function can assign values to these variables.

For example consider the lambda-abstraction

```
λx,y,z  [GO] -
                (AGNT) -> [ANIMATE:?x]
                (OBJ) -> [ANIMATE:?x]
                (SRCE) -> [PLACE:?y]
                (DEST) -> [PLACE:?z)
```

With the formal parameters of x, y and z bound to [PERSON:John], [CITY: Paris], and [CITY:London] respectively, then given that type restrictions and the conformity operator are respected, the instantiated graph would be

```
[GO] -
        (AGNT) -> [PERSON:John]
        (OBJ) -> [PERSON:John]
        (SRCE) -> [CITY:Paris]
        (DEST) -> [PLACE:London)
```

Maximal Join

According to Section 3.5 of Sowa (1984) two graphs may have a *common generalisation*; that is, the graphs have a common imbedded graph from which each graph could have been derived by canonical formation. The two graphs contain projections of the common generalisation. Given any two graphs, it is possible that the projections of each graph will be *compatible* , that is, capable of being merged by a series of joins, possibly with additional restrictions and simplifications, except if the merge is blocked by incompatible type labels or referents. A given merger of compatible projections of two graphs might be able to be extended if at the boundaries of the projection, other relations and concepts are found which could be accommodated in the merger. This process must have a maximum limit in finite graphs; the resulting compatible projections are maximally extended. A join on maximally extended common projections is called a maximal join.

A maximal join can be visualised as one graphs overlaying another, with the corresponding pairs of identical relations merging into one relation, and each compatible pair of concepts forming a new concept which contains the most specific information from the pair. Compatible subgraphs merge into a common core graph. Incomopatible subgraphs (differing relations and unrelated concepts) branch off from the compatible core in an additive manner (Figure 7).

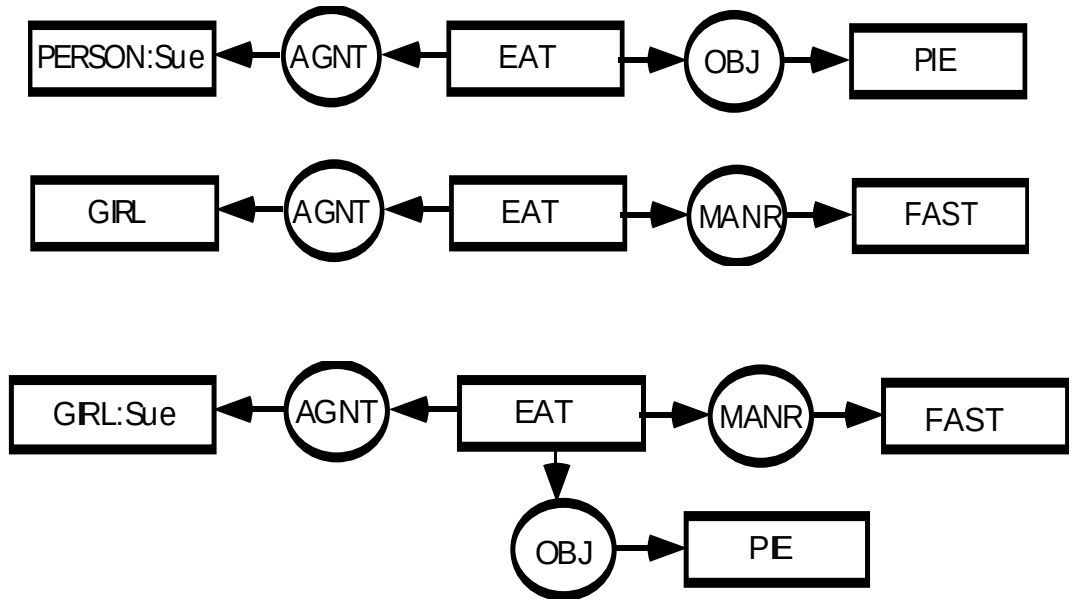

Figure 7. Maximal join of the top two graphs produces the lowest graph. Note the preservation of the most specific information in the merge.

A maximal join can be considered the most economical and informative collaboration of two graphs. Thus, it is an ideal way of accumulating information from a number of small component graphs into an integrated whole. For this reason, the maximal join is a most important function for building up representations of sentences from component words and phrases. Finding the global maximal join is a non-trivial combinatorial problem. Fortunately a number of useful algorithms for tackling the maximal-join problem are available (McGregor, 1982; Hartley, 1990; Myaeng & Lopez-Lopez, 1991).

4. Value of Conceptual Graphs for Knowledge Representation.

4.1 Evaluation Criteria

A knowledge representation system entails more than just a data structure. Representation is such a fundamental issue in intelligent system design that the choice of a representation method, or formalism, has major theoretical and practical implications for the project. Davis, Shrobe and Solovits (1993) have identified five distinct roles that need to be played by a representation method, and which may therefore be used as a framework for evaluating it. After a brief description of these roles, conceptual graphs will be discussed with respect to each.

First, the representation method should form the basis of a *theory of intelligent reasoning*. According to Davis, Shrobe and Solovits, such a theory rests on three foundations: a) the representation's fundamental concept of reasoning, b) the set of inference operations the representation permits and c) the set of inferences the representation recommends. Second, a knowledge representation implies a set of *ontological commitments.* Since no modelling system can capture the full richness of the natural world, the representation must make some simplifying assumptions about how to divide up the world. In identifying features of interest, and deciding what can usefully be ignored, the representation makes explicit an important theoretical claim about what exists, and thus about what can be dealt with by an intelligent system.

Third, the abstractions in a knowledge representation must act as *surrogates* for the objects, events and ideas that it represents; that is, they must stand in for the real objects, ideas or events in any interactions involving them by virtue of a resemblance, or at least correspondence, at some level. The relationship between real objects and their surrogates is a complex matter, but two important issues are those of how the surrogate is coupled to it's real counterpart (i.e. how the surrogate is generated from perceptions, and how alterations to the surrogate will affect the real thing) and of the fidelity of correspondence (i.e. how closely does the surrogate model the real thing: what features does it include, omit or emphasise). Fourth, the representation must be a pragmatic and efficient *medium for computation*. The data structures that stand for things and the operations that allow inferences on them must be supported by practical computer software. Memory usage and computational tractability are important matters, particularly in large scale projects (as intelligent systems tend to be). Finally, the representation must be an effective *method of human expression*, so that human software engineers may readily use it to communicate knowledge to their systems and to each other. A system of notation designed without human use in mind will be ineffective as a tool for knowledge engineering.

4.2 Theory of Intelligent Reasoning

How do conceptual graphs measure up against these criteria? The conceptual graph formalism views intelligent reasoning fundamentally as reality-preserving manipulation of reality-describing (i.e. canonical) graphs. A basic set of canonical graphs, originating as described in Section 3.2, may be transformed by some series of operations into other graphs. If the given canonical formation rules are used, the new graphs will also be canonical. The canonical formation rules enforce selectional constraints during graph transformation, so a certain kind of logical nonsense, such as ascribing an inanimate object as the agent of some act, will not propagate. In themselves, however, these rules do not provide an inference mechanism, because they do not necessarily preserve truth. Selectional constraints are not strong enough to

prevent fallacies. Further constraints, in the form of inferential algorithms or heuristics are needed for this. Other kinds of constraints on the nature of derivable graphs, such as plausibility, might also be demanded. Inference, in this view, consists of the development of ever more elaborate reality-modelling graphs through the application of truth-preserving and/or plausibility-preserving operations.

Since conceptual graphs are equivalent to FOPC, we can use all of its symbolic operations as a the beginnings of a formal semantics, allowing the notation to be backed up with an operational denotation. If the operations of lambda-calculus (e.g. as provided by links to LISP functions) are also allowed, a powerful toolbox of operations is clearly at hand[3]. The formalism thus permits the use of a very large number of potential inferential methods. Although research using conceptual graphs for AI systems has favoured the use and development of higher order operations like those described in Section 3.3, it generally stops short of making recommendations about inferential methods. Many of the conceptual graph community have backgrounds in formal logic representation, and this group is notoriously silent on the matter. Choices must still be made by the system builder. However, lessons have been learned about the value of some operators. In Section 5, I will describe how the maximal join can be used to aggregate conceptual structures representing the meanings of sentences from fragmentary word graphs.

4.3 Ontological Commitment

The ontological commitment of a conceptual graph system is explicitly available in its supply of concepts and relations. The concepts are usually organised on a generalisation lattice, in which more specific types occupy lower nodes than more general types, and types are connected by supertype-subtype relational links to the next highest level. Each type node is associated with a canonical graph that defines the selectional constraints which must be observed when an instance of this type is being manipulated. Relational types are provided in an arbitrarily organised catalogue, also with associated constraint graphs. No prescription is made about what types should be included, the choice being left to the system builder according to the needs of the system. However, concepts and relations from the elementary catalogues provided in Sowa (1984) are frequently seen in papers and electronic mail. There appears to some dispute about whether or not to distinguish natural types from role types in the conceptual catalogue; doing so complicates the organisation of the catalogue and the type operations that depend on it. Failing to do so can lead to problems of placing some common types on the simple generalisation hierarchy. Another apparent ontological lack in the formalism is of clear guidelines about the use of referential fields in concept nodes. The conformity operator :: is expected to forbid the formation of concepts with inappropriate type-referent combinations, but no details about how this might be done are offered. Nor is the matter of possible entries in the referent field a closed book. Another problem is a lack of good semantics for sets, though this problem has now been tackled (Kocura, 1992). Still another unresolved issue is that of whether relational types should be organised into a hierarchy. Here again, more work needs to be put in before the formalism can completely fulfil the ontological commitment role.

4.4 Adequacy of Symbolic Surrogates

Given that one accepts minimal Aristotelian, classical-AI assumptions about value of symbolic representations, the assessment of conceptual graphs as useful surrogates depends on a) the theoretical claim that concept-and-relation semantic networks can model reality b) goodness of fit between graphs (nodes and structure) and a particular domain and c) the quality of the method used to generate the canonical graphs. The claim in a) has been examined and defended thoroughly in Brachman (1985), and will

not be further discussed here. The question of goodness of fit to the domain is related to the ontological commitment role, and is likewise dependent on good choices being made by the system builder. Good choices carve the domain at its joints, selecting concepts and relations that capture exclusive and important features, and at an appropriate grain size. How does one decide on these features? The process should begin with a clear idea of the intended purpose of the system. With this in mind, an analysis of the domain should be performed to expose a set of aspects, discrete entities and relationships, which can be encoded in the conceptual and relational catalogues. Example graphs, making interesting assertions about the domain might be sketched out to assess the adequacy of the catalogues.

For AI systems, I claim that an important guide here is a clear notion of an intelligent agent situated within and clearly demarkated from the domain (however abstractly this is conceived) and this leads to point c). To be causally linked to the real world, the some concepts and relations must be able to be automatically selected or built up from the catalogues in response to patterns or events which are distinguishable to the agent's perceptual apparatus, while others will be useful only by virtue of their implications for eliciting or selecting actions from the agent's behavioural repertoire. In choosing concepts for representing objects, we are bound ultimately by the capabilities of the perceptual machinery, but for actions, we need to commit the agent to a given action, or sequence of actions. In natural language understanding, words in the corpus may be used as a starting point, provided one is careful to avoid the trap of mistaking the mnemonic word labels that are often used as type labels for the content of the concept underlying the type. In practice, many choices will be made intuitively, based on the system builder's own cognitive structure for the domain. Such choices may be difficult to defend. Ultimately, the acid test for graphs representing the domain is their value to the agent interacting with the domain - and that is an empirical matter.

4.4 Medium for Computation

Are conceptual graphs a effective computational medium? The existence of a number of practical graph processors (e.g. Kocura & Ho's (1991) modular CG processor; the X-windows toolkit GET (Wermelinger & Lopes, 1992); Pfeiffer & Hartley's CP (1993); and the PEIRCE project's core modules (Ellis & Levinson, 1992), to name a few), together with a much larger number of research programs implementing parts of the theory (including the software described in Appendix A), testifies to the practicality of the formalism for computational purposes. A large body of both mathematical proof and experimental experience shows that the existing theory performs as advertised. Whether this performance is what is needed to build intelligent systems is another matter.

Statements about efficiency must be restricted to individual implementations. Since conceptual graph papers are generally couched in abstract terms to keep them from becoming bogged down in implementation detail, data on actual efficiency is rarely offered. In the absence of appropriate benchmarks, and with attention focussed on just getting tools running at all, questions of efficiency tend to take a back seat. In keeping with good software engineering practice, some effort is generally made to ensure the efficiency of proffered algorithms. The PEIRCE project, like many others, is still under construction and it may be some time before it can be optimised. In the system described in Appendix A, no claim is made that the methods used are optimal, or even particularly efficient. In some of the more complex and important functions, such as the maximal join, this may create considerable time delays when performing experiments.

4.5 Medium for Human Expression

Perhaps conceptual graphs fulfil the fifth role, that of a medium for human expression and communication, best of all. Most people find the formalism very easy to learn and use. The very readable structures, employing wordlike type labels, together with the choice of display forms, are widely believed to help account for the popularity of the formalism. At conferences and in the electronic forum, graphs are the *lingua franca* , appearing routinely as ways of expressing complex ideas. However, while individuals can usually make sense of graphs written by another, there appears to be little agreement across individuals in the form of graphs expressing the same idea. When I informally asked a small number of conference attendees to make graphs from a list of simple sentences, I was surprised that no two graphs for a given sentence were alike. This is probably due to my not being explicit enough about the concept and relations allowed, but structural differences were apparent, too. For example, when asked to draw a graph representing the sentence "John and Henry gave gifts to Kate and Mary", one subject drew the following graph:

```
(PAST) -> [PROPOSITION:
            [PERSON: x (){John, Henry}] <- (AGNT) <- [GIVE:v (x y z)] -
                                                (OBJ) -> [GIFT:y{*}]
                                                (RCPT) -> [PERSON:z(){Kate, Mary}] ]
```

while another drew

```
(PAST) -> [SITUATION: [GIVE] -   (AGNT) -> [PERSON:{John, Henry}]
                                            (OBJ) -> [GIFT:y{*}]
                                            (BEN) -> [PERSON:{Kate, Mary}] ]
```

With longer graphs or more complex ideas the opportunities for divergence are much greater. Individuals clearly have different ideas about what constitutes the best canonical graph describing a given state of affairs. This need not be a problem for individual systems which can generate and use graphs in a consistant way, but it could cause trouble if the graphs were used to communicate between inconsistant systems.

5. Using Conceptual Graphs to Model Natural Language.

5.1 Existing systems

One of the most important uses of conceptual graphs is to represent natural language utterances. A number of systems now exist in which a graph forms a conceptual structure, or *conceptualisation* , assembled from and corresponding to one or more sentences. A good way to think about this assembly process is to consider the model-kit analogy (Noble, 1988). This model states that natural language is like the instructions in a construction kit in that it contains information useful for putting together a conceptual structure. Assembly information is of two basic kinds: information for identifying the parts, and information describing how the parts fit together. In natural language, *content* words (nouns, adjectives and verbs) access conceptual building blocks from the catalog of conceptual types, while *function* words (such as determiners, adverbs and prepositions) help clarify relationships

between the building blocks. The idea of conceptual parsing is to automate the process by contriving a procedure which selects the suitable component graphs from a supply, and connects them together so that the correct relationships are expressed.

A group working at IBM's Rome Scientific Centre (Velardi, Pazienza & De'Giovanetti, 1988) has a system capable of generating surface-level conceptual graph representations of Italian sentences from a corpus of press agency releases about economics (about 100,000 words). It uses comprehensive catalogues of concepts and relations (850-1000 abstract concepts in a type hierarchy and 50 relations in a simpler relational hierarchy) and a large dictionary of word-senses (2000 word-sense definitions appearing as terminal nodes attached to the abstract concepts). Word-sense definitions are sets of Prolog rules representing elementary graphs holding a) semantic expectations and b) "pragmatic" language usage information (Pazienza & Velardi, 1987). This information is represented as word-sense-relation-concept triples. A supertype field link each word-sense to an abstract concept on the hierarchy. Using a parse tree generated by syntactical and morphological analysis, a semantic verification algorithm finds suitable word-senses for each word or phrase, looks up plausible relations between them, and assembles complete conceptual structures using the information. The semantic verification algorithm works by replacing noun and verb markers in the parse tree with concepts selected from the conceptual catalogue and by matching word-sense definitions to concept-relation-concept triples in the resulting structure.

The IBM Paris Scientific Centre's KALIPSOS system (Berard-Dugourd, Fargues & Landau, 1988) is another large system, also implemented in Prolog, and capable of processing French sentences. KALIPSOS uses a lexicon of word-sense graphs stored in the form <NAME> is <CG>, where <NAME> is a functional term comprising a lexeme and generic syntactic graph representing the syntactical role the lexeme is playing, and CG is a word definition graph representing important concepts and relationships constraining the conceptual role of this sense of the lexeme. A variety of algorithms for conceptual processing are under development, but no detailed information on performance is yet available**.**

Another sizable accumulation of conceptual graph-based tools, including a natural language parser and semantic interpreter is the Extendable Graph Processor system at Deakin University (Garner, Lukose & Tsui, 1986). A variety of knowledge sources are available for the use of the langauge system. These include a word-sense lexicon, in which content words and function words are represented by short meaning-graphs; a supply of component graphs, from which complex conceptualisations may be assembled; default schemas, which allow elaboration of a concept or simple graph with defaults and background information; a set of type definitions represented as lambda abstractions; and tables representing the type hierarchy, the set of allowable type-referent conformities and frequency of association of one concept with another. A set of pattern correlation functions is used to limit search among the large sets of graphs. Functions computing various correlation measures between pairs of concepts and graphs include semantic distance, type frequency, and special conceptual relevance, and preference functions, designed expressly to allow plausible selection of a relevant graph to join on when more than one option is available. These measures are computed dynamically when needed from data which is automatically updated when the knowledge in the system is expanded. Together with a set of pattern modification functions, these correlation functions are used by the semantic interpreter, which is a bottom-up recursive algorithm similar to that of Sowa & Way (1986) (see below).

What lessons may be learned from these efforts? A common feature of successful systems is the large size of their knowledge bases. Obtaining a plentiful supply of component graphs appears to be a costly but unavoidable requirement, particularly if the system's coverage of the natural language is to be broad. A variety of approaches

to the representation and assembly process of the conceptual parser are used, but since data about the performance of these systems is almost never published, it is difficult to make comparisons between them. This regrettable tendency is not restricted to the conceptual graph community; it appears to afflict the much of the natural language field (a notable exception is the Message Understanding Conferences (Lenhert & Sundheim, 1991)), and to some extent the whole artificial intelligence discipline. Another lesson is the importance of good representational methods, since the large number of component graphs must at present be written by hand.

5.2 Requirements for Conceptual Parsing

There are four basic requirements for a working conceptual parser. The first requirement is a set of component graphs, the building blocks from which conceptualisations will be formed. Second, a method allowing single words or small phrases to elicit the appropriate component graph from the set is needed. Graphs in a conceptual catalogue may be pointed to from word-accessed entries in a lexicon. The problem of lexical ambiguity needs to be addressed in this method. The third requirement is for mechanisms which can combine the component graphs into more elaborate graphs. Finally, an organising principle, which can be used to guide the process of combining the component graphs into the final conceptualisation, is needed. The following subsections deal with each of these requirements in more detail.

5.2.1 Component Graph Set

In conceptual graph theory, semantic knowledge consists of a conceptual catalogue and a relational catalogue. Different ways to organise the information under each entry are possible. Pazienza & Velardi (1987) use extended case frames, with as many as 20 relational extensions on a stem representing the word-sense. The organisation of flexible case-frames for language representation has also been discussed in Sabah & Vilnat (1991). These extensions encode both semantic extensions and word usage patterns. Sowa & Way (1986) suggested that each entry in the conceptual catalogue have the following information: the type label itself; a definition for the concept, consisting of a lambda-abstraction (except for primitive concepts, which have no composition); a canonical graph, representing the selectional constraints on, or allowable attachments to, a concept of this type; and one or more schemata, representing defaults, expectations and other background knowledge. In their system, Sowa and Way use the canonical graphs as the building blocks in the assembly process, with the associated schemata assisting in some way.

Since primitive concepts have no graph describing them, there is the question of how they can be used by a language system. It is possible to assign single conceptual type and relation nodes to such primitive concepts. Sowa (1990) has described a set of heuristic rules by which mappings can be made between some expressive forms in natural language and conceptual graphs. Among these are examples of nouns and verbs which can often be mapped into a single concept, e.g. “lady” = [LADY] and “dance” = [DANCE] and modal auxiliaries like “can” and “must” map into relations (PSBL) and (OBLG).

In AI systems which have no perceptual, emotive or motor mechanisms, the value of such ungrounded, arbitrary symbols is often a mystery, and this can lead to criticism and misunderstanding about artificial knowledge. Primitive symbols (those which are not cognitive in nature) need not be meaningless if they can stand for raw perceptual, emotive or motor experiences. Only in AI systems which have subsystems involving these experiences can the idea of semantic primitives ever fully make sense. Primitive concepts such as “blue” occur frequently in human language, so this issue must be

properly addressed by a natural language processing theory. The pioneers of conceptual knowledge representation understand this, with conceptual dependency theory having, for example, "PPs" (picture-producers) as a class of perceptual primitives (Schank, 1972), while Sowa (1984, p.70) explicitly draws on “percepts” as source of graph components. However, the implications of this lesson have still not been fully appreciated. At the very least, keeping the sensory, emotive and motor implications of concepts firmly in mind can help with the creation of meaningful definition graphs for non-primitive concepts, as mentioned in Section 4.

5.2.2 Lexical Access to Conceptual Building Blocks

Each word of a text should access an entry in a lexicon, which gives information about one or more possible syntactic roles this word can play in sentences. Associated with each role is a link to one or more entries in the semantic catalogues. Thus, the meaning of a given word may be ambiguous, because several concepts may be indicated. For example, in the Sowa & Way (1986) system, each lexical record has a list of pointers to word-sense records, each of which accesses the type and canonical graph of a conceptual catalogue entry. Although access to more than one canonical graph per word is not an intractable problem (it is tackled in the Sowa and Way system by trial and error at the conceptual assembly stage), it is less than desirable for reasons of efficiency and time. Ideally, only one canonical graph, representing a particular sense in the current context should be elicited. Human beings can undoubtedly identify the relevant sense of a word for a given context. This can be thought of as a gating process operating at the time of lexical access: only contextually relevant word-senses should ever be considered as conceptual building blocks.

How can a current context be represented if the meaning of the sentence has not yet been ascertained? In the domain of navigational directions, it is possible to reliably identify the pragmatic class of a sentence from a superficial analysis of word patterns (Mann, 1992). Each class can have an associated conceptual graph representing a context for word-sense disambiguation. This graph can be compared to each of the candidate word-sense graphs using a semantic matching algorithm (Wuwongse & Niyomthai, 1991). This algorithm scores each pair of graphs according to an importance scalar for concepts and the semantic distances of matching concepts on the type hierarchy. The pair with the highest score will contain the most relevant word-sense graph for the given context. There is at least one other, probably cheaper, constraint which may be applied to eliminate candidate word-senses: the principle verb in a sentence constrains the possible senses of associated noun and adverbial phrases. A (large) table of purely linguistic information encoding these constraints can apparently be used to obtain such economies (Winston, 1984). A lexical access function combining these methods can be brought to bear on the lexicon to elicit a minimal set of component graphs in the fastest and most computationally inexpensive way.

5.2.3 Methods of Combining Graphs.

As new information becomes available to a conceptual parser (or other system that does reasoning by accretion of conceptual graphs) it must be combined into the existing graph in some systematic and truth-preserving way. Two processes based on the operations discussed in Section 3.3 are required to achieve this: *incorporation* and *instantiation*..

1. Incorporation is used when a graph or list of graphs needs to be combined into a whole, beginning with an initial, kernel graph. Each new graph must be maximally joined to the evolving graph, since this is the most economical, information-preserving join. The maximal join algorithm requires that a pair of start concepts, one

from each graph, be specified. The maximal joins must also be carried out in a particular order (so that, for instance, the subgraphs of each noun phrase and prepositional phrase are fully assembled before they are joined to the root verb phrase). The pairs of start concepts and the order are provided by an *organising principle* (see below).

2. Instantiation is used to resolve the values of variable referents in concepts in the evolving graph. The instantiation functions takes a graph and a variable binding list consisting of pairs. Each pair consists of a variable name and a concept. For each pair, the function searches the graph for the named variable and tries to substitute the concept, with its possibly more specific type and referent field, in order to maximise the specificity of the graph. Instantiation is also the process by which referents expressing indirect or anaphoric references may be resolved. A concept with # as its referent triggers a search for the more specific referent of a pre-existing concept of the same type. This information is kept in a globally-accessible table of current concepts, which is continuously updated as new graphs are built. If no specific information is found in this table, default values are used. The default information may come from non-linguistic sources, such as visual observations; so that the concept [ROAD:#] ("this road") can be correctly identified as a specific road [ROAD:#1248] from the set of world state observations. Because this operation may bring in defaults which could exclude other, more appropriate specifications, it should always be carried out last.

5.2.4 Organising Principle

To assemble graphs correctly and reliably , the conceptual parser needs to be informed by some kind of organising principle, which tells it how and when to choose an incorporation or instantiation function, how to parameterise them, including how to choose the appropriate subgraphs for each operation. In the conceptual parsing of natural language, this organising principle appears in two forms: a parse tree from a syntactical parser, and a theory of case role attachment. Particularly in the early stages of assembly, an unambiguous parse tree is a valuable guide to the order in which subgraph components should be joined.

A method like that used by Sowa & Way (1986) is used. It works as follows: for each terminal node of a parse tree, an appropriate component graph is chosen, using the lexical access method described above. Then following the structure of the tree, the components are pairwise incorporated into intermediate graphs representing the meaning of the phrasal constituents. Each intermediate graph is then joined to others, with control ascending the parse tree and unifying the partial meanings, until the root node is reached, and the process is complete. Where some problem, such as ambiguity in the selection of an appropriate place to join, is encountered, the rules identifying case role attachments in the component graphs and matching them to the case role requirements of the evolving graph are applied in a way that maximises satisfaction of those requirements, while accounting for as many component graphs as possible (Figure 8).

5.3 Theory of Conceptual Assembly

A general theory of assembly of conceptual graphs from diverse information sources would be useful not only for conceptual parsing, but also for other kinds of processing with conceptual graphs. Only the outlines of such a theory are given here. References will be made to the conceptual parsing process, but the overall principles apply to other non-linguistic assembly tasks as well.

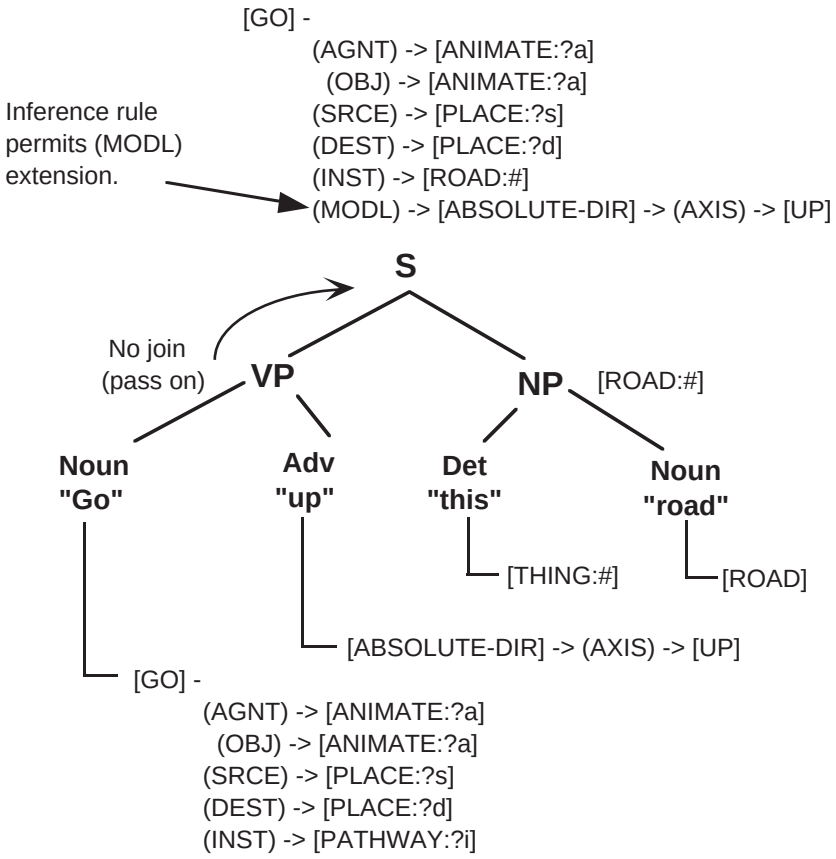

Figure 8. A recursive function climbs a parse tree, incorporating each component or intermediate conceptual graph into the evolving whole, to form a surface level representation of the meaning of the sentence. The presence of unspecified extensions to certain relations in the graph at the root node, or of component graphs which are still unaccounted for, triggers a search for rules that permit further incorporations to be made. For example, the graph for the adverb "up" cannot be incorporated during tree traversal, but [GO] may be extended with a (MODL) attachment, which can incorporate the graph. In the absence of specific information, later processing will access a default value for variable ?a (Self) and a current context value for both ?s and the indefinite reference in [ROAD:#], but no value for ?d which is at this point genuinely undefined.

The theory asserts that understanding means, in part, the ability to draw on multiple, diverse sources information in order to build a conceptual model (here a conceptual graph) of some state of affairs. In the case of language comprehension, information from a text stream, a lexicon, a conceptual knowledgebase, a pragmatic context, and a table of current concept values are combined to make a surface level model of the text. Further processing employs inferential rules to elaborate the model, and fill in any gaps that were left from analysis of the text. Ultimately the graph is passed on to other, different processors, depending on the role it is to play (see Section 5.4).

Any process of conceptual assembly requires a set of conceptual building blocks to be drawn from conceptual storage. Something of this sort is described in Sections 5.2.1 and 5.2.2. The building blocks could also be accessed by visual icons, sounds, or other raw sensory experience. These building blocks form a kind of evidence about some state of affairs in the world, and thus represent bottom-up input into the conceptual assembly process. Initially, however, the building blocks are fragmentary and disorganised, and so do not constitute a useful model of this state of affairs.

A pragmatic analysis of raw sensory experience activates one aspect of a conceptual graph called a pragmatic context (not to be confused with the data object called a context) which contains an overall model of the situation or task with which the system is dealing. The activated context represents a stored preconception, or microtheory, about one particular aspect of this situation or state of affairs which will be the focus of this conceptualisation, influencing the assembly process in a top-down direction. As well as constraining access to the conceptual knowledgebase (as in lexical disambiguation), the activated pragmatic context accesses a skeletal structure, or *kernel*, which will act as an armature around which the conceptual structure will aggregate. Though generalised enough to be accessed in response to a broad range of regularities in the sensory experience, the kernel is too vague and rudimentary to be a useful model.

The idea of conceptual assembly is to use one or more organising principles (Section 5.2.4) to successively attach the building blocks to the kernel in a way that respects the pragmatic context while accounting for the maximum number of building blocks. This collision of influences between top-down microtheories and bottom-up perceptual influences affords powerful, reality-preserving constraints to the assembly process and is a common feature of modern, psychologically informed cognitive processing models (e.g. Minsky, 1986; Grossberg, 1987).

One approach to conceptual assembly would be to assign parameters or "valencies" to all the concepts to be joined. The term valency is used by analogy with the processes of ionic or covalent bonding in the spontaneous formation of molecules in physical chemistry. Valency values would express the relative strengths of potential bonds between concepts of various types. To accrete a complete conceptual graph, a "molecular chemistry" algorithm would simply evaluate the potential joins between the evolving graph and all competing component graphs. Each graph would be attached using a maximal-join at the most optimal site (computed by a suitable energy-reducing function). No ordering of joins or extra information would be required at the assembly stage; the graphs would simply be mixed together like reagents in a solution, and a complete graph would naturally form, like a compound molecule. The chief problem with such an approach is, of course, that correctly attaching the correct valency to each concept is potentially a difficult task, because it would be difficult to foresee how the values would interact in an arbitrarily complex reaction. The problem of making appropriate joins has simply been transformed into another, possibly less tractable, form. A more structured approach, encouraging the use of general organisational principles about how to make good joins, and affording more control of details in the assembly process, is probably preferable.

Since incorporation and instantiation are the methods by which structured attachment is achieved, the organising principles must be expressed or realised in a way that provides i) a pairwise joining order and pairs of start concepts for the incorporation, ii) a variable binding list for the instantiation process. In practice, the organising principle is a procedure driven by a data structure such as a tree, or by inferential rules combining linguistic and case role semantic knowledge. This procedure will begin with the kernel and the set of building blocks, and continue incorporating them until as few as possible building blocks remain. The structure-following method is tried first. If any building blocks remain, inferential rules allow the process to continue.

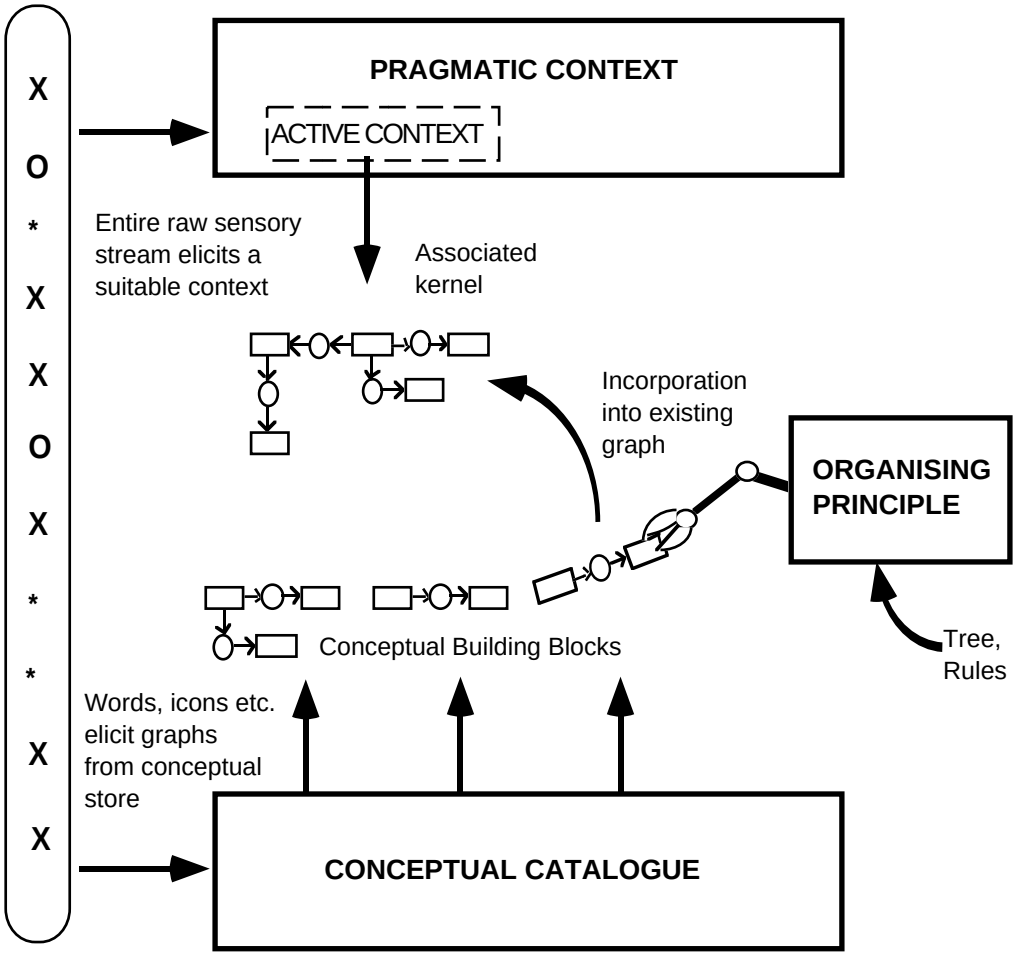

Figure 9. Assembly theory calls for an organising principle, such as an inference procedure, to successively incorporate component graphs into an evolving whole, beginning with a pragmatically selected kernel. The kernel is accessed from a context activated by the raw sensory data stream. The component graphs are each elicited by objects in the stream, under the influence of the active context. When all of the component graphs are accounted for, collected variable bindings are instantiated into the final graph, carrying defaults to fill any remaining gaps.

The evolving graph "bids" for attachments, while the building block graphs "offer" their services in particular roles. The goal is maximal satisfaction of both parties. Failure to account for many graphs can force another context to be tried. At the end of the process, a variable binding list, which has been opportunistically collected during the earlier stages, instantiates variables in the evolving graph with specific referents. Unresolved references are handled by anaphoric reference to the previously assembled graph, looked up in the table of current concept values or allowed to default to values associated with the active pragmatic context.

5.4 Playing the Language Game

Pragmatically speaking, it is possible to imagine how different kinds of speech acts should be handled using conceptual graphs: declarations, questions and commands are different kinds of language usage that must be treated differently, even though similar methods might be used to arrive at the assembled graphs. The use of the graphs in the larger system ultimately depends on the purpose of the system, but some general methods may apply.

Declarations could be handled by maintaining a database of graphs representing the state of the world. Each new declaration would be represented by a graph expressing some observation of or relationship among entities in the world. These graphs would be asserted onto the world-state database, subject to checks for contradictions or redundancy. Assertions would be made on the basis of sensory observation, valid inference or *a priori* realisation. The inference process will be non-trivial, since an indefinite number of inferences could follow from a given assertion. Negations would search for the relevant matching concepts and remove them. Changes in the world state might be handled either by altering the relevant matching concept (if the world state was handling only the current world state) or else by time-stamping assertions (if the world state database was keeping a history of all past states of the world) in the manner used by Vere & Bickmore's basic agent (1990).

Questions can be handled by representing them as existentially and universally quantified queries of a database. The queries can be thought of as graphs with certain parts represented by variables or The denotation operator δ enables appropriate boolean and sets to be returned, supplying the missing parts, which can form the basis of an answer (see Sowa (1984) Section 4.4). In the Conceptual Graph Processor described in Appendix A, queries of a database of fact graphs can be made using the function **query-db**, which searches a specified database for a fact which contains a projection of the graph representing a query. If found, the resulting projection can be interpreted as an answer to the question posed by the theory.

Commands: may be handled using conceptual graphs with [ACT]s from the behavioural repertoire as their head concepts and some concept referring to the system itself (such as [SELF]) in the agentive case. Verb transition semantics are useful in overcoming the limitations of primitive act theory for practical control of action (Cercone & Shubert, 1975; Allen, 1978; McKevitt & Wilks, 1978). A context expressing one or more preconditions attached to the central verb concept is checked to see if the implied action may be performed. Predictions about the consequences may be inferred from a context containing postconditions. Provided that the preconditions for an action are fulfilled and no adverse affects follow from the predicted consequences, the world state database is updated by an appropriate set of assertions and deletions, and then the system must perform the action in the real world by executing an appropriate parameterised call to a demon written in the base language. More indirectly, a command may be thought of as a goal (a conceptual graph representing a state of the world and placed on a goal-database

Some work has also been done on the generation of well-formed sentences expressing the meaning of conceptual graphs (e.g. Bell & Joyce, 1989; Dogru & Slagle, 1992). This process appears in principal to somewhat simpler than the converse, because a relatively simple graph traversal mechanism, with a simple lexical choice method can produce a humanly readable, if somewhat stilted, grammatical structure. Choosing appropriate words to express complex structures is more difficult, and making choices among the various expressive and strategic possibilities,

such as how to choose the focus, what information to include and exclude, and how to model the intentions and needs of a conversational partner remain important research issues.

Notes.

1. At the time of writing (October, 1993), the first version of PEIRCE was being released. It is accessible by anonymous ftp from a site at the Computer Science Department, University of Queensland using the following command sequrence:

```
ftp     ftp.cs.uq.oz.au
cd /pub/peirce
binary
get peirce0.1.tar.Z
```

The code is compressed in tar form. It can be converted into ASCII text using the following commands:

```
uncompress peirce0.1.tar.Z
tar -xf peirce0.1.tar
```

It is necessary to edit the file default.mk, altering the line

```
-DPEIRCE_LIB_PATH="\"/homes/ged/peirce/\"" $<
```

as appropriate to contain the name of directory where you are installing PEIRCE. Finally, issue the make command. The source code can apparently be compiled into a working system using most C++ compilers; the AT&T C++ compiler is recommended. For further information contact Gerard Ellis at the Computer Science Department, University of Queensland: ged@cs.uq.oz.au.

Other official ftp sites for PEIRCE source code are

ftp.cs.su.oz.au (129.78.8.208) in the directory /pub/cg.
crl.nmsu.edu (28.123.1.18) in the directory /pub/cg

A beta-test version of code has become available for the Apple Macintosh, called MacPeirce version 0.1. It, too, can be obtained by anonymous ftp.

The ftp host is camis.stanford.edu in the directory /e/ftp/pub/MacPeirce_0.1

The filename is: peirce_0.1m2.sit.Hqx. This file is a BinHexed Stuffit archive. If you use FETCH version 2.1b4 the conversion and decompression to a folder will be done for you automatically; the file manual.ps is a postscript manual explaining how PEIRCE works. If you use some other ftp method, you will need Stuffit Expander 3.0 or some other such decompressor/convertor. Log on to camis as anonymous, and use your e-mail address as your password. For further information contact Keith E. Campbell at Stanford University: campbell@camis.stanford.edu.

2. A diagrams containing boxes and circles, like Fig. 1, is said to be in the *display form*.. This form is easy to read, but difficult to type into a computer without an elaborate graphical interface. For this reason, graphs in many systems, including the

one provided here, are displayed in the *linear form*.. The display form represents concepts like this [CAT:#143] and relations like this (LOC). The sentence 'The cat sat on the mat.' would appear as

[CAT:#] -> (STAT) -> [SIT] -> (LOC) -> [MAT].

For more details about the linear form, see Sowa (1984), Section 3.2. Note that the linear form implemented in the graph processor described in Appendix A is a slightly modified version of the form described by Sowa. Most of the changes are needed due to the special uses of characters by the standard Lisp reader. The semicolon : separating the type and referent fields in concepts and contexts has been replaced by an exclamation mark ! .The # sign used to indicate particular referents is replaced by the equals sign =. To shorten graphs written in the linear form, the arrows are expressed as less than or greater than signs (< and >). No full stops or commas may appear. Nested clauses are terminated with a slash /, instead of a comma , . In the modified form, the above graph therefore looks like this:

[CAT!=] > (STAT) > [SIT] > (LOC) > [MAT]

3. The notation supports additional kinds of graph nodes, together with specialised links, called actors, which are procedural demons able to be triggered during graph manipulation, and which can read and write symbols to and from other nodes. Actors in such dataflow graphs are a way of making certain operations explicit in the graphs themselves. It is not clear that operations involving alterations in graph structure could easily be carried out in this way, however.

References.

Allen, J.F. *Natural Language Understanding*. Menlo Park, California: Benjamin/Cummings, 1987, 324-333.

Bell, J.R. & Joyce, R.C. Mapping Conceptual Graphs into Natural Language. *Technical Report #89/6* Department of Computer Science, James Cook University of North Queensland, January, 1989.

Berard-Dugourd,A., Fargues, J. & Landau, M.C. Natural Language Using Conceptual Graphs. *Proceedings of the International Computer Science Conference* '88 Hong Kong, 265-272.

Bornerand, S. & Sabah, G. Syntax-directed Joining for Language Understanding Processing. *Proceedings 5th Annual Workshop on Conceptual Graphs*, Boston, Mass. and Stockholm, Sweden, 1990, E.02.

Brachman, R.J. What is-a is and isn't: An Analysis of Taxonomic Links in Semantic Networks. *IEEE Computer,* 1983, 16,10, 30-36.

Brachman, R.J. On the Epistemological Status of Semantic Networks. In R.J. Brachman & H.J. Levesque (Eds.) *Readings in Knowledge Representation* Los Altos, California: Morgan Kaufmann, 1985, 191-215.

Cercone, N. & Scubert, L. Toward a State Based Conceptual Representation. *Proceedings 4th International Joint Conference on Artificial Intelligence,* Tbilisi, U.S.S.R.,1975, 88-90.

Davis, R. Shrobe, H. and Solovits, P. What is Knowledge Representation? *AI Magazine*, 1993, 14, 1, 17-33.

Dick, J.P. On the usefulness of Conceptual Graphs in Representing Knowledge for Intelligent Retrieval. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 153-167.

Dogru, S. & Slagle, J.R. A System that Translates Conceptual Structures into English. *Proceedings 7th Annual Workshop on Conceptual Graphs*, Las Cruces, New Mexico State University, 1992, 167-176.

Ellis G. & Levinson, R. The Birth of PEIRCE: A Conceptual Graphs Workbench, *Proceedings 1st International Workshop on PEIRCE*. Las Cruces, New Mexico State University, July 10, 1992, 149-156.

Findler, N. *Associative Networks: Representation and Use of Knowledge by Computers*. New York: Academic Press, 1979.

Gardiner, D.A. & Slagle, J.R. Extended Conceptual Structures for Intelligent Document Retrieval. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 169-185.

Garner, B.J., Lukose, D. & Tsui, E. Parsing Natural Language through Pattern Correlation and Modification. *Proceedings 7th International Workshop & Conference on Expert Systems and their Applications*, Avignon, France, 13-15 May, 1987, 1285-1299.

Genesereth, M.R. & Nilsson, N.J. *Logical Foundations of Artificial Intelligence*. Los Altos, California: Morgan Kaufmann, 1987.

Grossberg, S. Competative Learning: From Interactive Activation to Adaptive Resonance. *Cognitive Science*, 1987, 11, 23-63.

Hartley, R.T. Algorithms for the Substrate MGR. *Memoranda in Computer and Cognitive Science* #MCCS-90-197, Computing Research Laboratory, New Mexico State University, 1990.

Kocura, P. & Ho, K.K. Aspects of Conceptual Graphs Processor Design. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 317-329.

Kocura, P. Towards a Deep Knowledge Semantics for Conceptual Graphs. In T.E. Nagle et. al. (Ed.s) *Conceptual Structures: Current Research and Practice*. Chichester: Ellis Horwood, 1992, 109-126.

Lendaris, G.G. A Neural Network Approach to Implementing Conceptual Graphs. In T.E. Nagle et. al. (Ed.s) *Conceptual Structures: Current Research and Practice*. Chichester: Ellis Horwood, 1992, Section 8.3 pp 165-167.

Lenhert, W. & Sundheim, B. An Evaluation of Text Analysis Technologies, *AI Magazine*, 1991, 12, 3, 81-94.

Mann, G.A. Assembly of Conceptual Graphs from Natural Language by means of Multiple Knowledge Specialists, *Proceedings 7th Annual Workshop on Conceptual Graphs*, Las Cruces, New Mexico State University, 1992, 149-156.

Masterman, M. Semantic Message Detection for Machine Translation, using an Interlingua. *Proceedings of the 1961 International Conference on Machine Translation*, 1961, 438-457.

McGegor, J.J. Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software Practice & Experience*, 1982, 12, 23-34.

McKevitt, P. & Wilks, Y. Transfer Semantics in an Operating System Consultant: The Formalisation of Actions Involving Object Transfer. *Proceedings 10th International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, 569-575.

Minsky, M. The Society of Mind. New York: Simon & Schuster, 1986, p86.

Myaeng, S.H. & Lopez-Lopez, A. A Flexible Algorithm for Matching Conceptual Graphs. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 135-151.

Nagle, T.E. Nagle, J.A. Gerholz, L.L. and Ekland, P.W. *Conceptual Structures: Current Research and Practice*. Chichester: Ellis Horwood, 1992.

Noble, H.M. *Natural Language Processing*. Oxford: Blackwell Scientific Publications, 1988)

Nogier, J.F. & Zock, M. Lexical Choice as Pattern Matching. In T.E. Nagle et. al. (Ed.s) *Conceptual Structures: Current Research and Practice*. Chichester: Ellis Horwood, 1992, 413-435.

Pazienza, M.T. & Verlardi, P.V. A Structured Representation of Word Senses for Semantic Analysis. *3rd Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, April, 1987, 249-257.

Peirce, C. S. *Collected Papers of Charles Sanders Peirce*, Arthur W. Burks (Ed.). Cambridge, Mass: Harvard University Press, 1960.

Pfeiffer, H.D. & Hartley, R.T. The Conceptual Programming Environment, CP. In T.E. Nagle et. al. (Ed.s) *Conceptual Structures: Current Research and Practice*. Chichester: Ellis Horwood, 1992, 87-107.

Riesbeck, C.K. Some problems for conceptual analysers. In K. Jones & Y. Wilks (Ed.s) *Automatic Natural Language Parsing*. Chichester: Ellis Horwood, 1983, 178-181.

Riesbeck, C.K. Realistic Natural Language Processing. In A. M. Aitkinhead and J.M. Slack (Ed.s) *Issues in Cognitive Modelling*. Lawrence Earlbaum Associates, 1985, 193-206.

Sabah, G. & Vilnat, A. Flexible Case Structure Implemented into a Deterministic Parser. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 343-357.

Schank, R.C. Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 1972, 3, 552-631.

Schank, R.C. *Conceptual Information Processing*. Amsterdam: North Holland,1975.

Sowa, J.F. *Conceptual Structures*. Menlo Park, California: Addison-Wesley Publishing Company, 1984.

Sowa, J.F. Using a lexicon of canonical graphs in a sematic interpreter. In M.W. Evens (ed.): *Relational models of the lexicon.* Cambridge: Cambridge University Press, 1988, 113-137.

Schroder, M. Knowledge Based Analysis of Radiology Reports Using Conceptual Graphs, *Proceedings 7th Annual Workshop on Conceptual Graphs*, Las Cruces, New Mexico State University, 1992, 213-222..

Sowa, J.F. Toward the Expressive Power of Natural Language, *Proceedings 5th Annual Workshop on Conceptual Graphs*, Boston, Mass. and Stockholm, Sweden, 1990, E.06.

Sowa, J.F. & Way, E.C. Implementing a semantic interpreter using conceptual graphs. *IBM Journal of Research & Development* , 1986, 30, 1, 57-96.

Velardi, R.C. Pazienza, M.T. & De'Giovanetti, M. Conceptual graphs for the analysis and generation of sentences. *IBM Journal of Research & Development* , 1988, 32, 2, 251-266.

Vere, S. & Bickmore, T. A Basic Agent. *Computational Intelligence*, 1990, 6, 4, 41-60.

Venk, S. & Govind, R. Development of a Generalised Expert System for Production Grinding Systems using Conceptual Graphs. *Proceedings 5th Annual Workshop on Conceptual Graphs*, Boston, Mass. and Stockholm, Sweden, 1990, E.19.

Wermelinger, M. & Lopes, J.G. An X-Windows Toolkit for Knowledge Acquisition and Representation Based on Conceptual Structures. *Proceedings 7th Annual Workshop on Conceptual Graphs*, Las Cruces, New Mexico State University, 1992, 233-242.

Winston, P.H. *Artificial Intelligence* . 2nd Edition. Reading Mass.:Addison-Wesley, 1984, 314-325.

Wuwongse, V. & Nyomthai, S. Conceptual graphs as a framework for case-based reasoning. *Proceedings 6th Annual Workshop on Conceptual Graphs*, State University of New York at Binghamton, 1991, 119-133.

Appendix A: Quick Guide to Macintosh Common LISP Conceptual Graph Processor.

A.1 Basic representation of conceptual graphs in LISP

The basic constituent objects of conceptual graphs, concepts, relations and contexts, represented as association lists. The function **create** generates a unique token for each new object, and assigns an association list as its value. When a typical concept is evaluated, the result is an a-list like that shown below:

((TAG C127) (TYPE PERSON) (REFERENT John) (NOTE NIL) (ARCS (R87 R89)))

The concept with the token name C127 is of type PERSON, has the referent John, and is linked to the relations identified by tokens R87 and R89. The idea of representing links using a list of connected objects is a reasonably efficient way of navigating on graphs, but has the drawback of being difficult to use to encode directedness, that is, it cannot be used to indicate the direction of the arrowhead on the link. This has not proved to be a serious problem for my purposes; it could be a problem for some applications. An inconsistency in the use of fields is that non-nil single entries into them are stored as atoms. This implies the need for a nil-atom-or-list test every time a field is accessed; it would have been simpler and more efficient to allow only list entries. The NOTE field can be used to store useful information about the concept. For example, if after parsing a sentence a concept is assigned to some part of speech, the concept can be annotated with a marker such as VERB.

A relation object has a simpler structure, with no referent field. Relation R87 might be

((TAG R87) (TYPE AGNT) (NOTE NIL) (ARCS (C95 C128)))

The ARCS list for relation objects is a list of concept or context tokens.

Contexts are represented as a context object plus a set of concepts, relations or contexts forming graphs within the context. A typical context object is shown here:

((TAG CX150) (TYPE SITUATION) (REFERENT C108) (NOTE NIL) (ARCS (R122 R152)) (CATALOG (C108 R70 C109)) (ENVIRONMENT NIL))

Context are like concepts, except that they contain one or more graphs (possibly containing other contexts). These are represented within the context object as a list (or list of lists, if there is more than one included graph) of objects in the CATALOG field. The referent field contains the name of the most important concept in the included graph, called the *head* (or a list of these as appropriate). To enable contexts to be nested, an environment field is provided, in which the name of a dominating context may be stored.

A graph is represented as a simple list of the names (tokens) of linked concepts, relations and contexts. For example the graph G1 might be the list

(C140 R122 C145 R123 CX150 C108 R70 C109)

This set of objects has an implicit set of links. Note that the objects appearing within the CATALOG of context CX150 appear on the list. Generally, the order of objects in a graph is not significant, but for readability it may be wise to make the catalogued objects of a context appear after the context object containing them.

A.2 Functions supported

As explained in Section 3.1, a basic conceptual graph processor must include functions which can create, store and destroy the basic objects: concepts, relations and contexts. The **create** function generates a new token and sets the TAG field of a proforma association list to this token. To complete an object, values for the other fields must be provided. Exactly how this is accomplished depends on where and why the object is being made, but it has proved very convenient to enable graphs in the linear form (see Note #2) to be read and converted into the list form. The function **take-graph** enables this to be done. The LISP expression

(setq G1 (**take-graph** '([EAT] > (AGNT) > [CAT!=Fluff])))

reads the list containing graph expressed in a (modified version of the) linear form, and builds the corresponding list containing on relation and two concepts objects. This list then becomes the value of the variable G1. **destroy-graph** accepts a graph name like G1 and unbinds each token name within the graph from it's value, thus freeing up memory. Graphs may be displayed in the (modified) linear form using the function **display-graph**.

The contents of the various fields in an object are accessible using the function **fetch**. (fetch 'referent c127) returns the contents of the referent field of concept c127. The ability to quickly check concepts for position on the type hierarchy is clearly needed; the functions **supertype** and **subtype** decide if the named relationship applies to two type labels.

The function **instantiate** enables the values of variables to be included in graphs used as lambda-abstractions. All four of the canonical formation rules are implemented in the functions **copy-graph**, **restrict-graph**, **simplify graph and join-graph**.

The projection operator, π , is supported. The call (**projection** F Q) will return the projection of Q in F, if one exists, nil otherwise. Given a fact database consisting of list of graphs and a query graph (a general graph) the call (**query-db** database query-graph) will search for a fact graph which contains a projection of the query.

A **maximal-join** operator is provided. In addition to the names of a pair of graphs to be joined, it requires the evaluated forms of the pair of concepts, one from each graph, on which the join process is to begin. For example, the call

(setq J1 (**maximal-join** G1 G2 C58 C69))

generates a new graph J1 which is the result of a maximal join between graphs G1 and G2 beginning with the concept C58 from G1 and C69 from G2.

A.3 Getting started

To use the conceptual graph processor, you need:

a) a Macintosh computer with at least 5 M of RAM and running Macintosh Common LISP version 2.0.

b) a copy of the LISP Conceptual Graph Processor and Natural Language Support source code (in Disk 1 folder CG Processor and Disk 2 folder NLU, respectively). NLU contains a good deal of code written by Bill Wilson for experimenting with natural language, including a chart-parser, and a large English lexicon . NLU Experiments contains code written by Graham Mann for conceptual graph support.

The MCL environment supports documentation boxes which can be requested for all functions and these are complete for almost all functions. Good use of these can make getting used to the code easier.

Load the Macintosh Common Lisp, then, evaluate the file called loadup.lsp. This will attempt to load in the entire natural language system, including a complete 30,000 lexicon and chart-parser (file charts4.Lisp), a semantic knowledge base of about 200 concepts and relations (files make-semantic and make-relational), the set of canonical formation rules (file canonical-formation-rules.lsp) and the maximal join code (file maximal-join.lsp). Many of the basic support functions are found in the file object-support.lsp. It might be necessary to make some minor modifications to the files to get them loaded (typically, path names will need to be altered). Some files loaded by loadup.lsp may be unneeded, and may be removed from your copy of loadup.lsp.

The LISP source code is available on request from the software library in the Artificial Intelligence Laboratory, University of New South Wales. Note that it is experimental code written for experimental purposes and is not fully debugged and tested. Every attempt has been made to make the functions work well, but no claims are made about the reliability or efficiency of this software, and the user should take care when using the code for new work.