What Conceptual Graph Workbenches Need for Natural Language Processing

Graham A. Mann

School of Computer Science & Engineering,
University of New South Wales
Sydney 2052, Australia
mann@cse.unsw.edu.au

Abstract. An important capability of the conceptual graph knowledge engineering tools now under development will be the transformation of natural language texts into graphs (conceptual parsing) and its reverse, the production of text from graphs (conceptual generation). Are the existing basic designs adequate for these tasks? Experience developing the BEELINE system's natural language capabilities suggests that good entry/editing tools, a generous but not unlimited storage capacity and efficient, bidirectional lexical access techniques are needed to support the supply of data structures at both the linguistic and conceptual knowledge levels. An active formalism capable of supporting declarative and procedural programs containing both linguistic and knowledge level terms is also important. If these requirements are satisfied, future text-readers can be included as part of a conceptual knowledge workbench without unexpected problems.

1 Introduction

From the large number of experimental software systems designed to realise the power of conceptual graph (CG) theory, a few have emerged as contenders for the 'standard' workbench. Three such workbenches will be discussed here: the Loughborough toolset [9], the PEIRCE project [5] and the object-oriented UNE-CG-KEE environment [11]. These projects are serious attempts to apply good software engineering methods to supply the reliable, efficient and expandable knowledge engineering tools that the CG community so badly needs; with the basic CG processing machinery in hand, future experimental work can get on with higher-level knowledge structures, operators and applications.

Of the potential benefits the standard workbench might offer, one of the most important will be natural language capability. The demand for direct text-to-knowledge and knowledge-to-text has never been greater. Without a language capability, CG workbenches are as subject to the knowledge acquisition bottleneck as other knowledge systems and so will also be 'I/O-bound' in this way. But CG representations are especially suited to natural language, and provided some care has been taken in design stage, the standard workbench could provide practical text-reading and generation modules, ready to be customised for particular domains. Although only the PEIRCE project has such capabilities as stated goals (and has begun a processes of specification [13]), the provision of such language modules could make an important difference of whether or not a given system would be widely adopted. Even a fairly simple text parser would be a good 'selling point'.

What special requirements do language modules have for workbench design? How can the engineer ensure that a system will at least be able to support future language modules? Experience with BEELINE [10], a CG-based agent capable of parsing paragraphs of real text, will be used to identify the key points at which design might be affected. Since modern, well-written software is fairly easy to modify, mistakes at these points would rarely be disasterous; they would more likely result in inelegant coding and inefficient performance. But given the high standards of quality and performance being demanded for the standard CG workbench, this would still present a problem.

The following basic model of language use will serve as a basis for the discussion. A compositional model in which canonical graphs encoding particular word senses are joined togther into structures representing the overall meanings of larger text units such as phrases and sentences should be

uncontroversial by now. The resources - data structures and operators - required for langauge can be conveniently organised into four levels ranging from surface linguistic features to deep knowledge structures (though this should not be taken as a commitment to any kind of processing order or priority).

- A supply of lexemes and methods for composing them into words. (morphology)
- A supply of linguistic information about words (lexicon) and methods for for composing them into phrases and sentences (syntactical grammar, semantic rules).
- A supply of canonical graphs representing word meanings (conceptual catalog) and methods for assembling them into compound graphs (conceptual parsing using formation rules).
- A supply of higher-level knowledge structures (schemata, scripts, plans, discourses) and methods for fitting compound graphs into them (pragmatics, schematic matching, metaphor).

Each resource consists of a supply of data objects and the methods needed to apply them. It will be argued that a workbench capable of providing these resources will need

- the storage capacity for an adequate number of data of each kind
- easy-to-use entry/editing tools for the data
- efficient, bidirectional access to each entry
- an active formalism to implement the methods at each level, which can work between levels.

The four resources are somewhat different, and it is possible to imagine specific provisions for each. More likely, the language-ready standard workbench will group the first two resources (linguistic level) and the last two (knowledge level) and include toolsets, storage, access and formalisms appropriate for each grouping. The following sections are divided up roughly along that line.

2 Linguistic Level Provisions.

These considerations based on the model of lexical access described in Figure 1. This shows the path from a word to a lexicon entry via a morphology component which handles plurals, tense markers, gerunds and other modifying suffixes. The lexicon entry contains information on one or more word senses, each associated with its linguistic features such transitive/intransitive, syntactic constituent, or other part of speech. Each word sense must then access a canonical graph representing the definition of the word in terms of other concept types and relations. These definitions are stored in the system's knowledgebase.

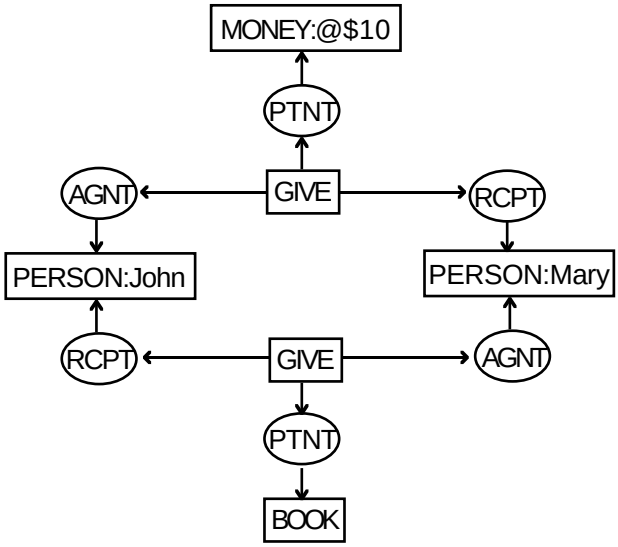
In English a small number of morphological transfromations captures regularities over the entire language. For example, only 6 rules account for spelling changes required to map stems and suffixes (e.g. try + s -> tries, strap + ing -> strapping, etc.). With such a small number of rules it is practical the encoding of such transformations as finite state automata for maximum efficiency [2], or as simple functions. Although some languages, such as Turkish, have many more of these rules, the numbers do not appear to be incompatible with such succinct and efficient coding schemes, and since many such systems have already been developed, they need not constrain the design of a CG workbench.

What are the strorage requirements for the lexicon? Williams [17], estimates that a working human vocabulary includes from 500 to 40,000 words, depending on the age, language aptitude, and education of the individual. The original lexicon employed in the BEELINE system contained approximately 32,000 entries; only a small fraction of which have since been actually defined with conceptual graphs. A language understanding system with 30,000 fully defined words would be a

considerable achievement. Swappable wordbooks of 5,000 to 10,000 entries may be acceptable for specialised applications.

Neglecting the conceptual graph definitions for the moment, data sets of this size and complexity present no great problems for modern editing tools. Nor should availability: a number of comprehensive lexicons, including Merriam-Webster's Concise Electronic Dictionary (80,000 entires) and the Longman Dictionary Of Contemporary English have been commercially available in machine readable form since the eighties [18]. For the lexicon it is convenient to have search and edit functions, and the ability to modify individual entries without re-reading the entire lexicon. The lexicon in BEELINE can find and update single word entries by selecting tools from pull-down menus; these have proven valuable for fast testing of the system.

It is clear that to be efficient, access from words to entries in a lexicon of reasonable size needs to employ some kind of hashing technique, in which each word is tagged with an address which can be used to retrieve the entry directly from memory. In LISP-based systems like Alvey and BEELINE, this can be achieved by making word strings access a hash table of lexical entries directly [14, 16]. What is not so clear is that the path from word to conceptual graph needs to be easily *reversible*. In generating well-formed language from conceptual graphs, methods based on the traversal of graphs have been suggested [1,4]. Such algorithms visit each node in the graph, building up conceptual and relational information which can be expressed through an appropriate grammatical form. In practice this works reasonably well for simple graphs, when the granularity of the graph is such that there is roughly a one-to-one correspondence between the concepts or relationships and words which express them. Where a cluster of concepts and relations could be more succinctly expressed with a single word, the output produced by simple traversal algorithms seems verbose and redundant. Such a method would express the following graph



as something like "John gives Mary $10 and Mary gives John a book.", whereas a person would recognise the pattern and say simply "Mary sells a book to John for $10".

While type contraction may sometimes be able to be used to simplify the graph by collapsing clusters, not all complex graphs would yield to this method. If a projection of some word-sense definition could be found in the complex graph, then the associated word would be available to account for the entire cluster [12], provided there was a path back from the definition to the word. Effectively, another type of contraction operation becomes available. Besides better quality of generation, such a definitional contraction might be useful for analogical reasoning, by exploiting the conceptual relationships between different senses of a given word. Suppose a given conceptual graph will not fit into an surrounding context or schema, but should. It needs to be modified in some way to enable the fit. The graph is contracted into its word. The definitions of different senses of this word are then expanded, making available related conceptual graphs to work with.

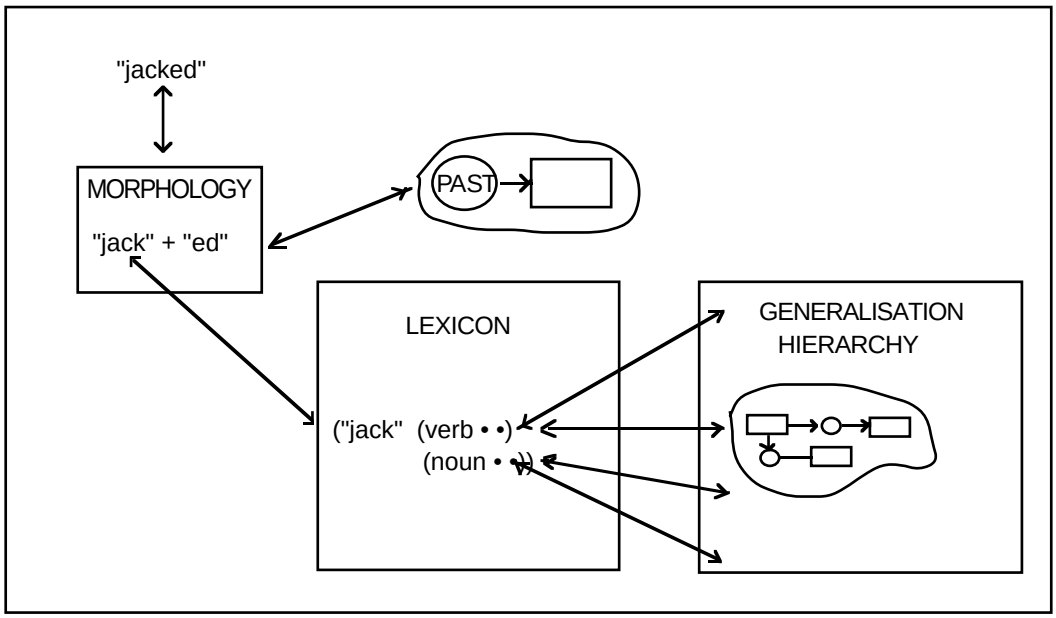


Fig. 1. A canonical graph representing the meaning of a word is accessed via a morphological analyser and a lexicon. Note the two-way pointers between all data structures.

Such operations require that the entire set of word definitions be efficiently searchable and that the word corresponding to each be reconstructable given the definition. In the Loughborough toolset, canonical graphs which would form word definitions are part of a complex set of interdependencies between other components of the knowledge base. [9] does not give details of how canonical graphs are represented or searched, so no indication of how to add a lexicon can be given. Presumably, pairs of pointers could bidirectionally link lexicon entries and particular graphs at some point within the complex. In PEIRCE, conceptual graphs can be associated with unique bit codes, which makes search very efficient. Words could be hashed into such bit codes, but the hashing method would need to be reversible. Note that the scheme described in [13], in which words are defined by packets of procedural information designed to create new structures and modify existing ones, is not simply or efficiently reversible for generation. In the UNE-CG-KEE system, lexical access could be accomplished by means of pointers from the lexicon entries to definition graphs, in the manner like that described in section 3.3 of [11] for schemas, prototypes and composite individuals. This access would be efficiently reversible if a second pointer from the graph back to the lexicon were added.

The software development language (C++ for all three workbenches under consideration here) could at a minimum be used as a basic active formalism. However, it would be an awkward and expensive way to deal with higher level abstractions, separate rules from interpreting mechanisms, and provide code for morphological and syntactical analysis. More practically, some of these linguistic level provisions could be supplied by an off-the-shelf NLP toolset, which could be adapted to the standard CG workbench. Good NLP toolsets come equipped with their own dictionaries, parsers and morphologies as well as formalisms for specifying lexical entries, morphological transformations, grammars and other types of information.

Sowa & Way [15] developed a conceptual graph parser using the PNLP system [7], which provided a dictionary of over 70,000 words, an English syntactical parser using a comprehensive augmented phrase structure grammar, frame-like record structures suitable for representing conceptual graphs and a high-level language capable of interpreting production rules or procedural programs. Another example is the Alvey Natural Language Tools project [3], which provides a standard, wide-coverage morphological and syntactical analyser for English. It includes a development

environment which can be used to build generalised phrase structure grammars (GPSGs), but with a 782-rule broad coverage grammar provided; a morphological analyser; a chart parser, an LALR parser, and a 63,000 entry lexicon. The system generates highly readable structures which could be easily be imported into a CG workbench. However, both these systems run in LISP, which is incompatible with the workbenches in question. For up-to-date details of existing tools, consult [8].

3 Knowledge Level Provisions

All practical CG systems need some way of entering and altering the graphs that constitute their knowledge bases. To enter graphs into the Loughborough toolset, for example, one types conceptual graphs in linear form into the terminal. The graphs are then extensively checked against the existing graphs in the knowledge base, ensuring consistency. According to Munday, Sobora & Lukose [11], this process is a thorough, but very slow way of building the knowledge base. This would present an obstacle to the entry of large numbers of word definitions. These authors have elaborate plans for their own editor, called CGE, for the UNE-CG-KEE system, which also includes integrity checking across the type, relation and generalisation hierarchies. It is not yet clear whether their design will overcome the performance problems identified in the Loughborough interface, since the it is not fully implemented. The CGE's specification also calls for "the capabilities to build the type hierarchy, build the relation hierarchy, define conformity relations, build graphs, define abstractions like type definitions, relation definitions, prototypes, schemata and composite individuals", which may take some time to deliver, but would appear adequate to support the natural language component. The PEIRCE interface seems a reasonable compromise between simplicity and integrity checking; it accepts graphs in the standard linear form, does less checking, and is consequently much faster. PEIRCE is also adopting GRIT, a sophisticated graphical I/O tool [6], which could solve many problems of finding, modifying and displaying complex graphs.

How many graphs will be needed for word definitions? As indicated in Figure 1, there is a one-to-many relationship between each lexicon entry and the conceptual graphs that represent the meanings of its word senses. The average fan-out - the number of meanings per word - is the crucial measure here. In BEELINE's lexicon there are an average of 1.19 linguistic categories/word; since there must be at least one word sense per linguistic category, this sets a minimum bound. However, there could be many different senses within a single linguistic category. For example, the word "jack" has at least 6 clearly distinct noun senses. But as a noun, "jack" is unusually ambiguous. As a verb it has 2 senses, which is probably more typical. If so, the average number of senses per word should be doubled to 2.38.

The mean senses per word overestimates the true number of graphs required because there is some redundancy in word meaning; some senses will use share their meaning-graphs with other senses. The number of graphs per word should therefore be less than 2.38. Allowing a conservative reuse level of 5%, the final estimated mean number of graphs per word is 2.26. For a vocabulary of 30,000 words, a storage for 67,800 graphs would therefore be required for their definitions. To estimate the size of these graphs, the largest example of a word sense definition in each of 16 papers from the CG natural language literature was examined. These graphs contained an average of 11.6 nodes (6.3 concepts plus 5.3 relation nodes; contexts were rare, numbering only one or two when they appeared). Because authors probably use unusually simple graphs for explanatory purposes, the true maximum size for meaning-graphs may be somewhat larger. Nevertheless, the figures hint at the size of definitions that developers are expecting to use.

In practical systems, still more conceptual graphs representing other forms of knowledge than word definitions, such as schema, prototypes, plans, etc. will need to be created, input and stored. It is difficult to generalise about the required number and sizes of such graphs across systems intended for different purposes, and using knowledge in different ways. Some kinds of schematic or prototypical knowledge might be organised alongside the word definitions to try to capture regularities in events, situations or plans in an economical way in graphs that are larger in extent but fewer in number. Just how many such graphs are needed for commonsense reasoning, or even

for reasoning within a specific domain, remains unknown. Strictly speaking, however, such graphs are outside the realm of language requirements, so such questions can be put aside here.

The basic operations common to all conceptual graph systems are the four canonical formation rules: copy, restrict, simplify and join, along with their associated service functions like conforms and subtype. These, along with I/O functions for entering and displaying graphs, are basic to any standard workbench and need not be discussed further here. One suggested additional function, not always considered, is a provision for *destroying* graphs (making the memory they occupy available for other uses). In language systems, large numbers of partial graphs, useful for only a short time, will be created. In order to minimise memory wastage, such graphs should be able to be destroyed once they have served their purpose. This feature is easy to implement in systems which dynamically assign memory to new instances of abstract data types.

The higher order functions maximal join and projection play a pivotal role in natural language. The join and maximal join serve as unification operators, enabling partial graphs to be merged into larger aggregations. The projection function can find partial matches between graphs, enabling a "smart join" when the starting concepts of two partial graphs are unknown. Most workbenches support (or plan to support) these operators. Consideration should also given to support for actors. Not only are actors part of traditional conceptual graph theory, but they provide a genuinely useful way of linking procedural processes to the otherwise passive declarative structures. In the BEELINE system, for instance, actors embodied as LISP functions are used to test hypotheses about possible joins betweens partial graphs collected during phrase processing, as well as to actualise the transition semantics of verbs.

All these operators need to be at the service of a control language which, like PNLP, should ideally allow the processing of both procedural code and production rules. In keeping with the cross-level policy, such a language must allow convenient reference to both entire graphs and their constituent nodes. It should also allow access to words, and phrasal groupings after syntactic parsing, with their associated linguistic features. The Loughborough toolset already meets these requirements to some degree. Its script language allows sequences of commands, IF-THEN and IF-ELSEIF rule structures and macros to be executed, but would require some modification to admit linguistic level terms. In PEIRCE, a Prolog-like “graphical object” language is being developed; such a language could use conceptual graphs as terms, but would again need extension to accord with the cross-level policy. The ability to refer to objects at both levels could occur naturally as the UNE-CG-KEE project unfolds. In Phase II, executable conceptual structures are promised to enable both declarative and procedural programming with graphs. In Phase III, customisable knowledge-based system shells would be added, encouraging knowledge engineers to develop specialised applications. A natural language system using the executable conceptual structures would no doubt be constructed at this level, once the necessary high level object-oriented data types were established.

4 Conclusions

One may wonder about the feasibility of hand-engineering comprehensive NL systems. It would be a difficult task to create, enter and maintain over 67,800 graphs by hand. Yet even by today's standards a manual entry of this number of entries would not be impossible, if the will was there. It is because of the cost and organisational difficulty of large projects that alternatives to simple manual knowledge entry must be developed. One advantage of a standard CG workbench could be the possibility of sharing this task out among developers. If the workbench's editing system made entering new word definitions easy, and enforced a generally acceptable semantics, word knowledge would naturally accumulate as ever-expanding versions of the workbench's knowledge bases were shared around the community. Similarity based search-copy-and-edit methods could form the basis of useful semi-automatic definition graph editors. The standards efforts could be expanded to encompass guidelines for the representation of basic patterns in language, beginning with a popular domain such as diagnostic medicine.

Ultimately, we could be forced to depend on automatic language acquisition, which would enable word definitions to be automatically generated or modified in the course of natural language dialogues with a teacher. Such learning would require both parsing and generating subsystems. The first step would be the manual development of a set of word definitions needed for expertise in the task of language learning by example dialogue. The second step would be the construction of a generator which could build a new definition up by arranging fragmentary components provided by the teacher around a newly-created unique stem linked to its appropriate entry in the lexicon. Finally, a method of altering faulty or exceptional graphs detected by the teacher during test dialogues would enable the quality of word meanings to be maintained.

5 References

1. Bell, J.R. & Joyce, R.C. (1989) Mapping conceptual graphs onto natural language. Technical Report #89/6, Dept. of Computer Science, James Cook University of North Queensland.

2. Berwick, R.C. (1987) Intelligent natural language processing: current trends and future prospects. In W.E.L. Grimson & Patil, R.S. (Ed.s) *AI in the 1980s and Beyond.* Cambridge, Mass, MIT Press.

3. Briscoe, E., Grover, C., Boguraev, B. & Carroll, J. (1987) A formalism and environment for the development of a large grammar of English, *Proceedings of the 10th International Joint Conference on Artificial Intelligence.* Milan, Italy, 1987, 703-708.

4. Dogru, S. & Slagle, J.R. (1992) A system that translates conceptual structures into English. *Proceedings 7th Annual Workshop on Conceptual Graphs*, Las Cruces, New Mexico State University, 167-176.

5. Ellis G. & Levinson, R. (1992) The birth of PEIRCE: A conceptual graphs workbench, *Proceedings 1st International Workshop on PEIRCE.* Las Cruces, New Mexico State University, July, 149-156.

6. Eklund, P.W., Leane, J. & Nowak, C. (1994) GRIT: An implementation of a graphical user interface for conceptual structures. Technical Report TR94-03, Dept. of Computer Science, University of Adelaide, February, 1994.

7. Jensen, K. & Heidorn, G.E. (1983) The fitted parse: 100% parsing capability in a syntactic grammar of English. *Proceedings of the Conference on Applied Natural Language Processing,* Santa Monica, California, ACL, 93-98.

8. Jung, C. (1994) The Natural Language Software Registry, WWW database. URL = http://www.dfki.uni-sb.de/cl/registry/draft.html, DFKI GmBH.

9. Kocura, P., Ho, K.K., Moorehouse, D. & Sharpe, G. (1991) Aspects of conceptual graphs processor design. *Proceedings 6th Annual Workshop on Conceptual Graphs.* Binghamton, New York, July, 1991, 317-329.

10. Mann, G.A. (1995) BEELINE - A Situated, Bounded Conceptual Knowledge System. *Systems Research and Information Science*, forthcoming issue.

11. Munday, C., Sobora, F. & Lukose, D. (1994) UNE-CG-KEE: next generation knowledge engineering environment. *Proceedings 1st Australian Conceptual Structures Workshop.* Armidale, Australia, November, 1994, 103-117.

12. Nogier, J-F. & Zock, M. (1990) Lexical choice as a process of matching word definitions on an utterance graph. *Proceedings 5th Annual Workshop on Conceptual Graphs.* Stockholm, Sweden, August, E.03.

13. Oh, J.C. et. al. (1992) NLP: Natural language parsers and generators. *Proceedings 1st International Workshop on PEIRCE.* Las Cruces, New Mexico State University, July, 41-49.

14. Russell, G., Pulman, S., Ritchie, G. and Black, A. (1986) A dictionary and morphological analyser for English, *Proceedings of the 11th International Conference on Computational Linguistics.* Bonn, Germany, 277-279.

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

16. Steel, G.C. (1990) *Common LISP: The language*. 2nd Edition. Reading, Mass: Digital Press, p. 435.

17. Williams, C.B. (1970) *Style and Vocabulary: Numerical Studies.* Bristol, Griffin & Co., pp. 66-67.

18. Wilks, Y., et. al. (1988) Machine tractable dictionaries as tools and resources for natural language processing. *Proceedings of the 12th International Conference on Computational Linguistics.* Budapest, Hungary, August, 1988, 750-755.