Proceedings of the Fourth International Workshop on Peirce: A Conceptual Graph Workbench

Gerard Ellis and Robert Levinson (Ed.)

University of Maryland, Maryland, USA, August 19, 1994.

Contents

1

Visual Structure Representations and Conceptual Graphs.

Andrew L. Burrow[1], **and Peter W. Eklund**[2]

ABSTRACT When well laid out, conceptual graphs are an intuitive knowledge representation language that graphically depicts the relations between concepts. Their utility as a human-computer intermediary is conditional on a solution to the layout problem. However, the layout problem is non-trivial, both in the computational cost for general approaches and in the effort required to reason about the influence of the layout algorithm on the understandability of the resulting graph layouts.
The following paper suggests an approach to the problem of laying out conceptual graphs. The intention is to include reasoning about the understandability of the results of layout algorithms along with an assessment of computational cost.

1.1 Introduction

The problem of conceptual graph layout could be considered yet another case to be solved by a general graph layout algorithm. However, it is our opinion that the problem requires an approach that takes advantage of the specifics of the conceptual graph case. Such an approach avoids the computational difficulties encountered in unguided and general graph layout, and identifies the role of layout in the process of interpreting conceptual graphs.

If the pictorial representation of a conceptual graph is to act as a human-computer intermediary it must clearly express the graph. A metric for the quality of a conceptual graph should be its capacity to be understood. The key role of a layout algorithm is therefore to improve that capacity. In order to understand this process an analogy is drawn between conceptual graphs and natural language sentences. In this analogy both the spoken sentence and the pictorial representation of a conceptual graph are considered to be utterances. In order for a conceptual graph to be easily humanly understood the pictorial representation should bear a similar relationship to the unrepresented conceptual graph, as the utterance of a natural language sentences bears to its internal realisation.

Having stated a view of the goal of a layout algorithm and an interpretation of the nature of the result, there remains the task of analysing the decision points in designing a layout algorithm.

1.2 Source of Layout Rules

To understand the possibilities in a layout algorithm consider a possible source of the layout rules. An extreme but common case is to use simply expressed aesthetic rules to guide the layout algorithm, e.g., attempting to minimise edge crossings. The attraction of such an approach is its simplicity. This system should let the relations represented in the graph speak for themselves, improving the understandability of the graph simply as a result of a tidier drawing while avoiding the complications of attempting to present any human meaning in the layout of the graph. However this approach suffers from the following deficiencies:

- very little use is made of the specialisation of the problem for the case of conceptual graphs;

[1]Department of Computer Science, University of Adelaide, email: alburrow@cs.adelaide.edu.au
[2]Department of Computer Science, University of Adelaide, email: peter@cs.adelaide.edu.au

- the human perceptual talents for the interpretation of pictorial representations are wasted, since any interpretations suggested by the shape of the graph are unintentional and possibly misleading;
- the human capabilities for recognition are wasted for the following reasons:
  - the author has no method of achieving control over the layout of the graph, so is less likely to identify with the result;
  - the application of aesthetic rules is global and makes no attempt to preserve local structures which may alter the shape of a drawing for each small change to the graph. This makes it less likely that users will readily recognise substructures across a number of graphs, and hence that they will acquire a visual vocabulary for conceptual graphs;
  - given the algorithm must search an infeasibly large solution space, it will by necessity find sub-optimal solutions, and the result of the algorithm will be dependent on the starting configuration. The opportunity for recognition by shape will therefore be diminished.

Some of these deficiencies could be avoided by adding rules to partition large graphs into subgraphs. This approach is discussed in some detail by Paulisch [30]. In particular we might be able to preserve local structure in the presence of non-local changes by partitioning.

To perform the partitioning we must use either general graph metrics such as connectedness, or features specific to conceptual graphs. However, a graph metric for partitioning on connectedness is likely to suffer from the same "arbitrariness" criticism of the metric for in this general layout method. In the second case case we are no longer applying simple aesthetic rules but instead make some surface semantic interpretation of conceptual graphs. In either case there remains the unintentional nature of the layout of the subgraphs. It is a key criticism of general graph layout algorithms that they produce unintentional and possibly misleading layouts.

At fault in the approach of applying aesthetic rules is a misunderstanding of the requirements of a graph layout. A conceptual graph does not require attractive layout, but rather layout that enhances its understandability. A key observation is that during the process of layout the unrepresented graph is embellished with information to produce a pictorial representation. We wish this information to contribute directly to the understandability of the representation and so must consider its nature. In an ideal solution this information would correspond directly to the information that the user interprets in the graph's layout. Where aesthetic rules guide the layout algorithm, continuous positional information is manipulated. However, it is an observation of Sowa, in describing conceptual graphs, that humans manipulate information as the relations between discrete concepts rather than as continuously varying attributes. In this case, positional information seems a poor model for information intended to improve the understandability of the graph to a human.

In order to tackle this ideal of graph layout we need to consider how the algorithm captures the process by which users interpret the layout of conceptual graphs. The algorithm needs to manipulate a representation of the humanly understood information contained in sound graph layouts. To capture this information in rules for layout we could either attempt to analytically deduce the rules from a knowledge of the humanly recognised meanings of pictorial representations, or use empirical methods to attempt to capture the usage of layout in conceptual graphs.

While it seems plausible that a study could be made of the usage of layout to imply meaning in pictorial representations, the author suggests that this problem is AI-complete — equivalent to the problem of natural language recognition.

In contrast to the approach of creating the layout rules analytically, the analogy of layout as utterance can be extended to view empirically captured layout rules as rules defined by usage. This is intuitive, as we would expect that for any natural language the usage patterns would define the expression of the language. Furthermore, this approach is not required to explain each part of a representation convention so is more amenable to providing layout rules for the breadth of conventions used to denote meaning. A set of layout rules for a given body of usage is dependent on the existence of enough examples to apply the empirical method, rather than a complete understanding of each element of the convention.

To further this discussion the following two methods are suggested:

1. let each new conceptual graph be laid out by the human author, or if generated by a machine then from the conjunction of the graphs from which it was formed. In both cases the layout is stored with the graph for future use;
2. express the layout rules as a collection of primitive canonical graphs with layouts. Each new graph is parsed for matches with these generic structures and the graph laid out by the conjunction of the layouts extracted from the canon.

The second of these two proposals stresses substructure recognition over saving graph layout and recognition of a whole graph. It embodies the principle of a vocabulary for visual structures. In contrast, the first proposal stresses whole graph recognition by the user. For this reason the first proposal is less computationally expensive, incurring layout costs once only, and performing the layout either through the author, in the case of a graph being added to the canon, or through a single conjunction operation for each graph join. The second proposal requires some form of structural parser as well as multiple conjunctions and does not enforce a particular timing for the layout.

That these methods handle information corresponding directly to a human interpretation, is dependent on the language used to describe the layouts. However, the methods themselves make use of conceptual graph specifics, i.e.,

- they share a visual structure representation that can be operated on in conjunction with conceptual graph operations;
- they make use of the result that all canonical graphs are the result of joins of graphs;
- the idea of using layout joins seems sympathetic with the description of the operations of a conceptual graph processor by Sowa, in that graph joins embellish the original graph with detail.

This discussion of the language required to describe the layout in the two preceding proposals anticipates the central proposal of the paper: that the visual structure of conceptual graphs should be represented in the language of conceptual graphs. Still to be determined are the details of the representational canon, and the conceptual processor support required to operate on the these layout descriptions.

1.3 Design of a Visual Structure Representation Language

The decision to implement a visual structure language was pre-empted by the description of the two empirical layout systems proposed in the previous section. Each of the systems saves layout information at different scales and different points in the process. However, both systems save layout information for graphs in some form, and hence require a visual structure representation language. In both cases the motivations for saving layout information are the same: to allow the layout algorithm to manipulate information that corresponds to the interpretation of the layout by the user, and to encourage human recognition of conceptual graph structures. These themes are central to the design of the visual structure representation language.

The intention of exploiting the human capabilities for recognition is a reduction in the user's cognitive load. This is analogous to the observation of Robertson et al. [34] that,

> *Smooth interactive animation is particularly important because it can shift a user's task from a cognitive to perceptual activity, ...*

To do this there must be an observable consistency in the representation of conceptual graphs. The focus of the language must be: to capture the humanly recognised aspects of visual structure, and to ensure that these aspects are preserved by the operations performed on the representations described in this language. In the first case, this constraint ensures that the user will recognise structures that have been stored, in the second that the user will recognise the structure across transformations such as graph joins.

The proposed language depicts concepts manipulated by the viewer in the process of interpreting conceptual graphs. In this sense it is not an artificial language, but furthermore conforms to a natural language subset for describing the position and orientation of conceptual graphs. The language depicts concepts and relations understood by humans that are intended to be stored and manipulated by a machine, and it must be possible to reason with the language to implement the layout systems proposed. Conceptual graphs form a superset of this proposed language since they are intended to depict concepts manipulated by humans. For these reasons, a natural implementation of such a language would be a canon of conceptual graphs. Furthermore, a conceptual graph canon implementing a visual structure language would benefit from progress in the language of conceptual graphs that it depicts.

In creating this canon the assumptions made by Sowa [37] are directly applicable. In particular, the assertion that humanly understood concepts are discrete by nature suggests it is fair to generate the canon through a selection of natural language terms that describe conceptual graph layouts, i.e., concepts that describe the humanly recognised relations within a layout rather than the precise positioning. An initial empirical basis for the refinement of this canon would be published drawings of conceptual graphs. However, in the long term a conceptual graph interface might have recourse to use three dimensional techniques to improve information density. Therefore, in designing a canon it would be wise to attempt to provide for extensions for the notion of layouts in space, or rather to create a two dimensional subset of a more general three dimensional case.

The final design of the canon must also consider the computational factors of a particular canon. Two considerations stand out: firstly, the problem of providing a conceptual processor to produce new layouts by joining already existing layouts; secondly, the cost of the final translation from the device independent layout description to the displayed pictorial representation.

The first consideration is a strong basis for choosing conceptual graphs to represent layouts, but it must be supported by careful design so that the process of joining layouts gives results that are sensible to the user. If the separate layouts of two graphs joined to produce a new graph are distorted in a non-understandable way during the join, the advantages of recognition will be lost for the users.

The second consideration is critical in implementing the system in a conceptual graph editor, and is independent of the conceptual processor. An editor that is not able to respond in a timely manner to incremental changes to the layout, will loose the advantage of presenting a direct manipulation interface. Furthermore, the interaction of the canon with the final translation will decide the opportunities for the editor to provide editing features to facilitate the capture of these layout relations from the user.

1.3.1 Related Work

Kamada and Kawai [41] present a partial solution to the problem of visualising abstract information [20], and Takahashi et al. extend this work to translate in both directions. This work is particularly relevant as it specifies a Prolog based visual structure representation language and describes the process of translation to the final pictorial representation using constraints. The second case of translating from the pictorial representation back to the visual structure representation language is less useful as the implementation described is fairly naive involving a costly total structure parse to capture user edits.

Kamada et al. describe the basis of their layout specification as the following principles, viz:

(1) *Constrain the reference points of graphical objects to lie on a geometric curve (line, circle, ellipse, arc, spline) at equal intervals.*

(2) *Connect the reference points or graphical objects by a geometric curve (line, arc spline) with or without an arrowhead.*

Based on these principles the authors build a set of predicates that describe objects and their relations in pictorial representations of abstract data. These predicates are separated into several groups based on their role, i.e., graphical objects, geometric relations, drawing commands, and constructed relations. In this scheme the drawing commands that connect and label graphical objects do not effect the layout. This is a useful separation because it ensures that we can treat graphs described by these predicates as tree structured graphics. The constructed relations apply the other predicates to form recognisable elements of drawing, e.g., listings of graphical objects on curves, and above and below relations.

The principles of Kamada et al. are adequate to describe three dimensional space, and the constructed relations describe humanly recognisable arrangements. In these respects the system, as described, suits our needs. Furthermore, these predicates could be converted to conceptual graph concepts and relations.

The considerations here being the need to implement a Prolog-like list to specify orderings, and the need to devise rules for joining layout specifications.

The system described by Kamada et al. performs a translation from the abstract data to the final pictorial representation *in situ*. The expression of the representation in the visual structure language is therefore ephemeral, consequently there is no need to consider the problem of matching the graphical objects and their corresponding abstract objects. In contrast, the translation from conceptual graph to pictorial representation may take place in two steps at two different sites, and the expression of the visual representation will be stored for later use. This later use may include redisplay of the graph and the creation of new representations for descendant graphs. The conceptual case therefore requires a mechanism for capturing the correspondence between the elements of a graph and their pictorial representation.

This problem of specifying the correspondence between an abstract object and its graphical representation is discussed by Pineda [33]. Pineda attempts to formalise the concept of a meaningful drawing and provide a foundation for interactive graphics in AI. In doing so he makes use of the grammatical notion of a *deitic expression* [28], a language feature concerned with the expression of orientation relative to the utterance. Pineda is concerned with *deixis* in the process of interaction. In contrast the problem here is the correspondence between the pictorial representation as generated utterance and the abstract form of a conceptual graph.

The projection operation of conceptual graph theory can be used to bind a description of the layout to a conceptual graph. Thus, for the case of a conceptual graph editor the *deixis* in interaction is specified by the initial binding of the layout description to the abstract graph being represented. The layout graph is manipulated directly by the graphical representation editor, and the subset of manipulations that deal with changes to the abstract graph are passed on to the data structure representing the conceptual graph.

There is a large body of work relating to the interactive editing of graphics with constraints. Kamada et al. [20] present a survey of the literature on constraints and graphics, as do the majority of papers discussing user interface tools. Li [27] describes the use of constraints in interactive editing. Vlissides' work on graphical object editing [43], implemented as the Unidraw library [44], is a framework for creating domain specific graphical editors that contain abstractions and code to support tree structured graphics laid out with constraints.

1.4 The Canon

To create a canon implementing the principles of layout specification described by Kamada et al. [20] several concept and relation types must be introduced. These types capture the layout principles at an abstract level, expressing the notions of graphical objects, reference points, curves, and listings. Further types are required to deal with nesting-layouts in compound graphs, and a method must be identified by which instances of these types refer to the elements of the conceptual graph they represent. These details will be outlined in the following section.

It is important to separate the composition of a structure specifying layout from that of a structure representing compound graphical objects. In the following discussion it is assumed that the conceptual graph editor will provide the translation from graphical object to image. This simplifies the description of the structure capturing layout.

In this discussion the basic type definitions of Sowa[3] are assumed. The following basic type definitions form a naive representation of 2-D space.

plane `PLANE` < `PLACE`

A plane is a stationary surface upon which images are located. It contains an infinite collection of points.

***x*-dimension** `XDIM` < `DIMENSION`

An x-dimension is a characteristic of a measure on a plane along the x parameter.

[3] See [37], Appendix B, pp405–424.

***y*-dimension** `YDIM < DIMENSION`
A *y*-dimension is a characteristic of a measure on a plane along the *y* parameter.

point `POINT < PLACE`
A point is a location on a plane, described by Cartesian coordinates.

```
concept POINT(x) is
    [PLACE:*x]-
        (LOC)→[PLANE]
        (CHRC)→[XDIM]
        (CHRC)→[YDIM].
```

image `IMAGE < ENTITY, INFORMATION`
An image is information that takes the form of an entity that can be communicated by being viewed.

A graphic object is an image contained within a rectangle described by the x and y bounds and a single centered reference point. The internal structure of graphical objects does not effect the process of layout, hence the type definition describes only the size of the object and its reference point, i.e.,

```
concept GRAPHIC(x) is
    [IMAGE:*x]-
        (PART)→[POINT:"Center"*c]
        (CHRC)→[XDIM:"Xmin"]
        (CHRC)→[XDIM:"Xmax"]
        (CHRC)→[YDIM:"Ymin"]
        (CHRC)→[YDIM:"Ymax"]
```

A curve is a geometric curve located on a plane. It is described by a mapping from a single parameter to a path on the plane. The layout algorithm distributes values of this parameter to the members of listings placed on the curve as well as to single objects placed on the curve. The members of a listing receive values ordered according to the order in the listing, so that each curve has a definite direction. The process by which these values are distributed is not covered here as it is part of the constraint solving process. Note that the curve is a separate concept from the listings of graphical objects placed on it. Indeed, several listings may be placed on the same curve, in which case they will be placed along the curve so that they do not overlap.

A listing is an ordered sequence of places each involved in a relationship with the preceding element and another relationship with the proceeding element. Unlike the lists in LISP, listings may not be nested, nor is a listing of 0 or 1 elements sensible. The relationships in a listing are fully captured by a single *successor* relation, with the restriction that each place may be involved in a single listing, i.e., the places in a listing are owned by that listing. While each place in a listing is a member of a single listing, places in separate listings may be the location of the same concept allowing listings to intersect. A type definition with these properties is:

```
concept LISTING(x) is
    [PLACE:*x]→(SUCC)→[PLACE].
```

For example, consider the following graph describing four graphical objects on a curve, one alone, the other three in a listing:

```
[CURVE]-
    (LOC)←[PLACE]←(LOC)←[GRAPHIC]
    (LOC)←[PLACE]-
        (LOC)←[GRAPHIC]
        (SUCC)→[PLACE]-
            (LOC)←[GRAPHIC]
            (SUCC)→[PLACE]←(LOC)←[GRAPHIC]
```

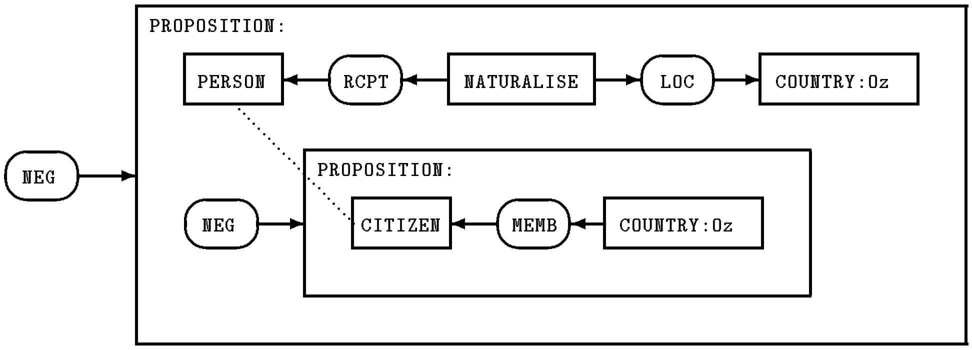

FIGURE 1.1. Subpictures in conceptual graphs.

Note that a listing is terminated by restricting the type label of the `PLACE` concept to some type other than `LISTING`, and that a location may become a listing simply by adding a successor.

Having introduced concepts and relations to describe graphical objects, curves, listings, and the location of a listing or reference point on a curve, a technique for binding a layout description to a conceptual graph is required. In the previous section the conceptual graph join was recommended for this task. To describe the join of a layout graph to the conceptual graph the correspondence between layout graph and the conceptual graph that it represents must be clearly defined. Two features require definition: the extent of the layout graphs, and the method of identifying correspondence between a graphical object and the conceptual graph feature that it represents.

The extent of the layout graphs is a measure of the coverage of a compound conceptual graph by a single layout graph: either the entire graph, or the the topmost simple graph. This decision point affords the opportunity for encouraging composition and modularity. If each layout graph describes the representation of a simple graph and each layout graph is contained within a single graphical object, a compound graph will be composed of layouts in principle analogous to the subpictures of Kamada et al. [20]. To understand this composition of layouts consider the case where each layout graph describes a simple graph. Graphs in the same context are represented by separate graphical objects and their internal layout is calculated separately. Referent graphs are hidden within the graphical object representing the concept of which they are a referent and their internal layout is not effected by the placement of their containing concepts.

This composition property simplifies the process of binding the layout description to the graph, and ensures that the graphics are tree structured. Tree structured graphics may be incrementally laid out: beginning with the damaged graph and passing the new size up to the enclosing graphical object, each enclosing object can lay out its components without effecting their internal layouts. For example, consider the compound graph in figure 1.1. This graph is composed of four simple graphs, i.e.,

```
(NEG)→[PROPOSITION]
[PERSON]←(RCPT)←[NATURALISE]→(LOC)→[COUNTRY]
(NEG)→[PROPOSITION]
[CITIZEN]←(MEMB)←[COUNTRY]
```

Consider the case where the second graph is edited to contain the fact that the ‘Wizard’ carries out the naturalisation, i.e.,

```
[PERSON]←(RCPT)←[NATURALISE]-
    →(LOC)→[COUNTRY]
    →(AGNT)→[PERSON:Wizard of Oz]
```

After this edit the the graphical object representing the second graph will have a new size requiring a new layout of the listing of graphs contained in the outer `PROPOSITION`. However, no changes will be required to the third and fourth graphs. The resulting change to the size of the outer `PROPOSITION` will cause a new layout for the first graph. Once the changes have propagated to the highest level graph the new positions of the components can be propagated down the structure. This final step does not alter the individual layouts, only their absolute position.

1.5 Conclusion

A script language has been proposed to capture the human details of conceptual graph layout, thus enabling a user interface to admit the human skills of recognition to the task of interaction. It is to be defined as a conceptual graph canon, taking advantage of the strengths of conceptual graphs in the representation of natural language, the principle of composition in conceptual graph context boxes, and the conceptual graph operations to manipulate these layout descriptions.

The work of Kamada et al. [20, 41] is presented as a strong basis for a definition of the language. This paper introduces a conceptual graph notation for the description of the layout frame via curves, reference points, and listings.

Three immediate directions for future work are identified: a data flow extension to this notation capturing the interaction of a constraint solver with the layout graph, an ontological basis for the binding of a layout graph to the graph that it represents, and a conceptual processor to implement the sensible joining of layout graphs.

2

A Role for Conceptual Graphs in Schema Integration

Peter Creasy[1]. Work done while the author was on leave at Department of Computer Studies, Loughborough University of Technology.

2.1 Introduction

Schema integration has become an important part of the information systems design process. The added impetus of database interoperability has led to the development of many theories and methodologies in the area and the necessity to automate the integration process. The topic of schema integration is a very broad area and encompasses federated systems, multidatabase systems at the database level to view and schema integration at the conceptual level.

The focus of this paper is on the conceptual schema integration level. In this area most work has been done using the relational and entity relationship (ER) [5] data models [3]. The claim for ER is that one can represent more semantics. However weaknesses in its representation power have prompted proposals using logic-based models. We have previously proposed conceptual graphs as a basis for schema integration [8] and also shown that conceptual graphs satisfy certain criteria as a canonical data model [7]. Earlier work which contributed towards constraint integration simply took functional dependencies and formed a minimal set of functional dependencies. While this work was significant in the area of database relations, functional dependencies do not by any means constitute all constraints. Significant work was done during the 1980s to systematise the approach, e.g. [2], [22].

Later efforts have focussed on the at least partially automating the integration process, generally with ER or ER enhanced, although many of the ideas are generic [39], [40]. With the realisation of some that ER is not semantically strong enough, some have proposed logic models with the idea of translating to these, generally still from ER. However we propose that with the right (canonical) model it is possible to integrate schemas developed in a number of different languages. In particular we have proposed that "off the shelf" schemas will make the use of canonical data models, such as conceptual graphs, an important part of the design process.

In this paper we shall initially discuss the integration problem and show what attempts have been made to structure and formalise it. In the next section we examine the role of canonical data models and then propose how conceptual graphs fills the role of a canonical data model.

2.2 Schema Integration

Typically, in a large organisation, the process of constructing a single conceptual model (conceptual schema) of the particular enterprise is either too complex or time consuming for a single analyst. Thus a number of analysts are involved in designing a number of subschemas for various parts of the enterprise. These must then be integrated into a single conceptual schema.

If approached properly with a data dictionary designed at the outset, this latter process should be relatively straightforward. However there are circumstances where a number of conflicts may occur. For example, in integrating a number databases (say, after the amalgamation of two companies), it may be desirable to integrate the databases by first integrating the schemas.

---

[1]Department of Computer Science, The University of Queensland, QLD Australia 4072
peter@cs.uq.au.o

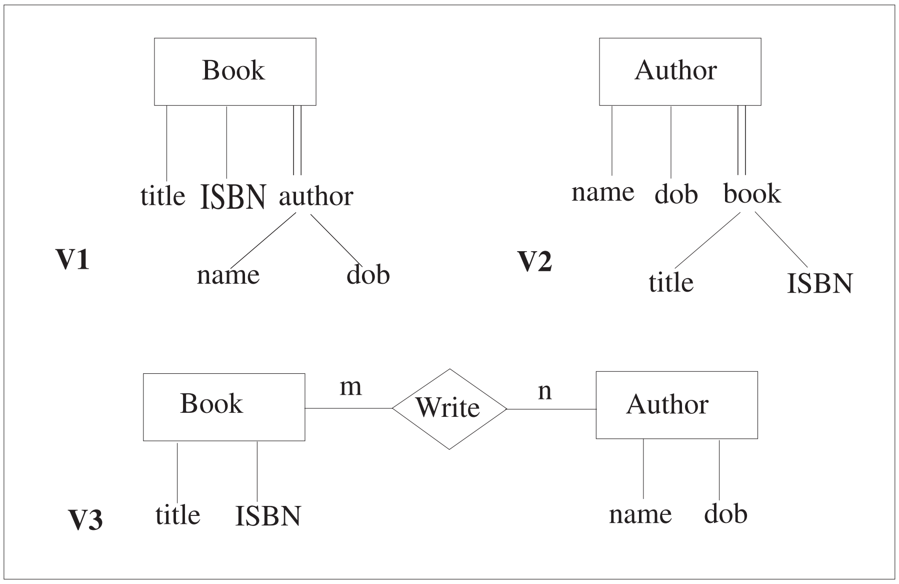

FIGURE 2.1. Three ways of modelling a UoD

An example of the problems which may occur are shown in figure 2.1. Three ways to model a relationship between AUTHORs and BOOKs is shown using ER. In one case AUTHOR is an attribute (multivalued) of BOOK, in the second BOOK is an attribute of AUTHOR and in the third there is a relationship between them. These representations possible arise because different designers have different perceptions of reality. This multiplicity of representations is referred to as *semantic relativism*.

It is very difficult to design systems to recognise that each of these models the same piece of reality. (In some ways this is due to ER's lack of uniformity, in that one distinguishes relationships and attributes, whereas in fact-based languages, such as NIAM [42], this does not arise. There would be only one way to map them. Of course the other side of the argument is that analysts/designers want to distinguish relationships and attributes, as, by removing attributes, an abstracted view of the design can be seen. So, semantic relativism is not a bad thing. In fact there should be this flexibility to permit a designer to model an enterprise as the designer sees it, within the framework of good design guidelines)

The first stage of the integration is to determine the conflicts which exist, *conflict analysis*. This involves a number of levels of checking:

1. Name conflicts

   (a) synonyms - same objects are represented by different names.

   (b) homonyms - different objects are represented by the same name.
   These are determined by:

   (c) concept similarity - differently named objects have similar properties - the generalisation concept is used to solve such conflicts.

   (d) concept mismatch.

   For example in figures 2.2 and 2.3 AUTHOR and CONTRIBUTOR are synonyms and one of them is renamed. BOOK and ARTICLE have concept similarity, but they are not identical; in this case they have a common parent - PUBLICATION. WRITTEN-BY and CONTRIB-BY are synonyms and one is renamed to give name unification. The difficulty here is identifying the conflict, i.e. how to detect that there is one. The resolution follows from the identification.

2. Structural conflict analysis
   In this phase concepts with the same names are checked for possible merging. We use the categories:

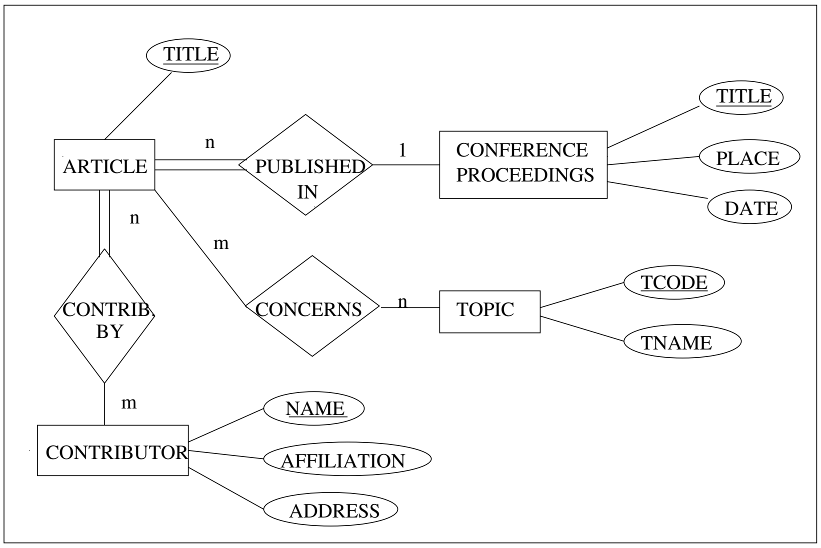

FIGURE 2.2. View 1 of the UoD as an ER diagram

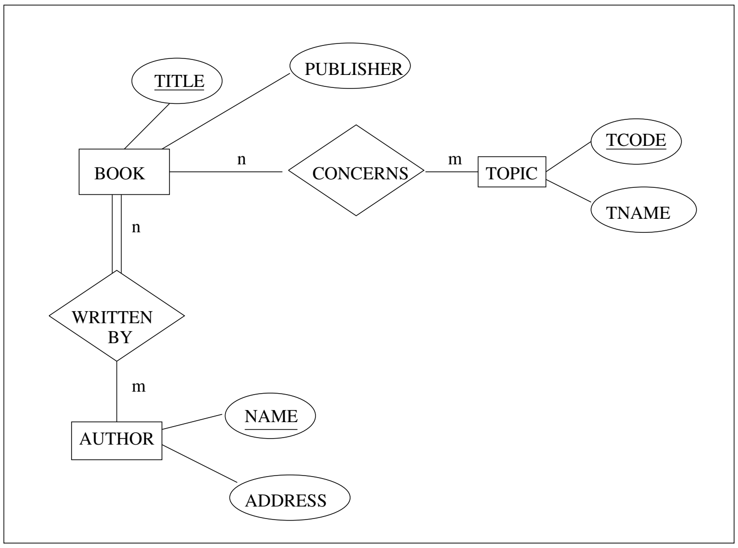

FIGURE 2.3. View 2 of the same UoD

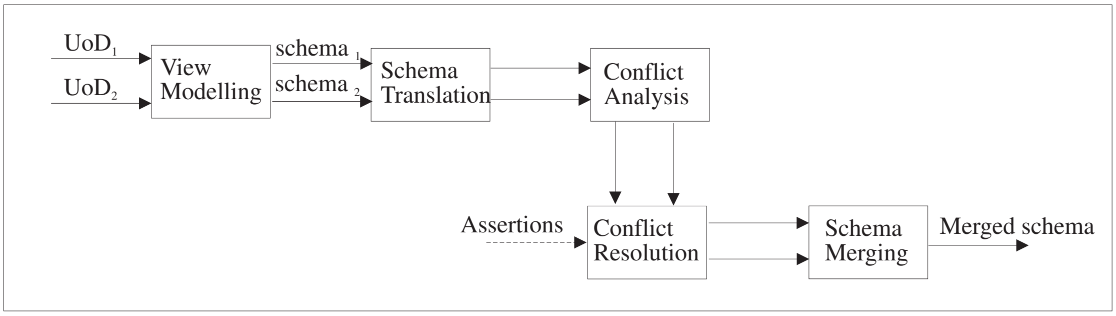

FIGURE 2.4. A schematic view of the integration process

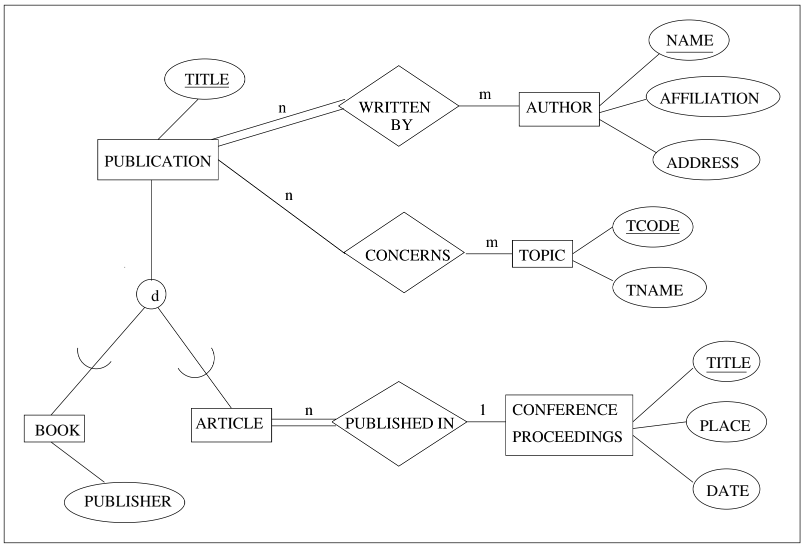

FIGURE 2.5. The integrated schema

(a) identical concepts
(b) compatible concepts
(c) incompatible concepts.

For example the concepts BOOK and AUTHOR are compatible in the view 1 and view 2 schemas in figure 2.1. Incomplete concepts would include those where the mappings are different. For example, one schema may have a relationship as N to 1, while another may have the "same" relationship as M to N.

Another important conflict is the data model conflict - which is handled by the canonical data model. We shall discuss this in more detail later. An integration of the schemas of figures 2.2 and 2.3 is shown in figure 2.5.

2.3 Canonical Data Models

While most work has been done in schema integration using a single data model, in areas where data models have already been developed (e.g. federated databases), it is important to have a canonical data model common to the whole enterprise. Saltor et al [36] have proposed a framework for such a canonical data model based on the representation factors of expressiveness and semantic relativism. We shall examine these as they appear to have, to a certain extent, become the de facto standard. This framework is very similar in many respects to that proposed by ANSI for the IRDS Conceptual Schema [32].

Expressiveness does not mean that one is necessarily looking for the "best" data model. A canonical data model may not be one that would be used in the ordinary course of modelling, however it would be desirable that it have readable and, preferably, graphical qualities. Expressiveness of a data model means the amount of conceptualisation that can be expressed. This is often referred to as semantic expressiveness and semantic power. It includes both structural and behavioural expressiveness. Its expressiveness should be at least as great as each of the data models involved in the integration. Greater expressiveness permits semantic enrichment which can simplify the integration process.

Structural expressiveness is the power of the structures of the model to represent concepts. For example, such a model should support generalisation, specialisation, between super and subclasses and aggregation and decomposition between complex and component part data objects. Most ER models currently considered in integration have been extended with this power. Behavioural expressiveness is not supported in the classical data models. However the power to represent the behaviour of objects is an important aspect of modelling.

Conceptual modelling is relative. It depends upon users' views of the world. Orthogonality is not the strong point of data modelling languages that it once was. While we need a single representation, we must allow for any number of external schemas. Semantic relativism is the degree to which all the different conceptualisations of the same real world can be accommodated.

2.4 Using Conceptual Graphs

In [7] we have pointed out that conceptual graphs satisfied the criteria for a canonical data model. In this section we shall examine some of the mechanisms that realises this satisfaction.

In [8] we demonstrated that ER could be translated to conceptual graphs (a bijection). In [6] we further demonstrated that the same could be achieved for NIAM, thus effectively permitting integration with schema conflict. If we examine the mapping from ER to conceptual graphs and some of the subsequent conflict resolution we can show the usefulness of conceptual graphs in the integration process.

In the translation ER concepts are mapped to concepts, attributes and relationships to conceptual relations. We refer to the resulting graphs as conceptual schema graphs. The integration of concepts is done in essentially the same way as integration in ER. We see an advantage of using conceptual graphs in a particular environment. We believe that the design of conceptual schemas will become so costly that designers will use well-defined "off-the- shelf" schema modules. Well defined meaning that each of the concepts and conceptual relations (assuming conceptual graphs are used) will be defined type structures. The generalisations necessary to sort out concept similarity will be available. Thus the conflict between ARTICLE and BOOK in figures 2.2 and 2.3 can be resolved. There will be a well-defined ontology for each environment. This will overcome problems of naming conflicts. Without such an ontology the conflicts will be manually sorted out. We have also previously noted that the richness and semantic power is ideal for defining, say, the relationships of ER. Which is essential if integration of these relationships is to take place.

The semantics of the ER relationships (i.e. the constraints) are carried by the relations. For example figure 2.6 is the translation of figure 2.2. In figure 2.2 there is an N to 1 relationship PUBLISHED-IN between ARTICLE and CONFERENCE-PROCEEDINGS, with a mandatory role on ARTICLE (i.e. every article must be published in a conference proceedings). The conceptual relation PUBLISHED-IN will then be a MAN-N-1 type. Each conceptual relation is classified according to its constraints.

In cases of concept incompatibility we use can our generalisation rule to provide a solution. For example if we have a mandatory 1 to N mandatory constraint on a relationship in one schema and 1 to 1 mandatory in another for a corresponding relationship we can use the constraints lattice of figure 2.7 to resolve the problem and integrate them as 1 to N mandatory. Other examples of the integration provided by the conceptual graphs path are given in [8].

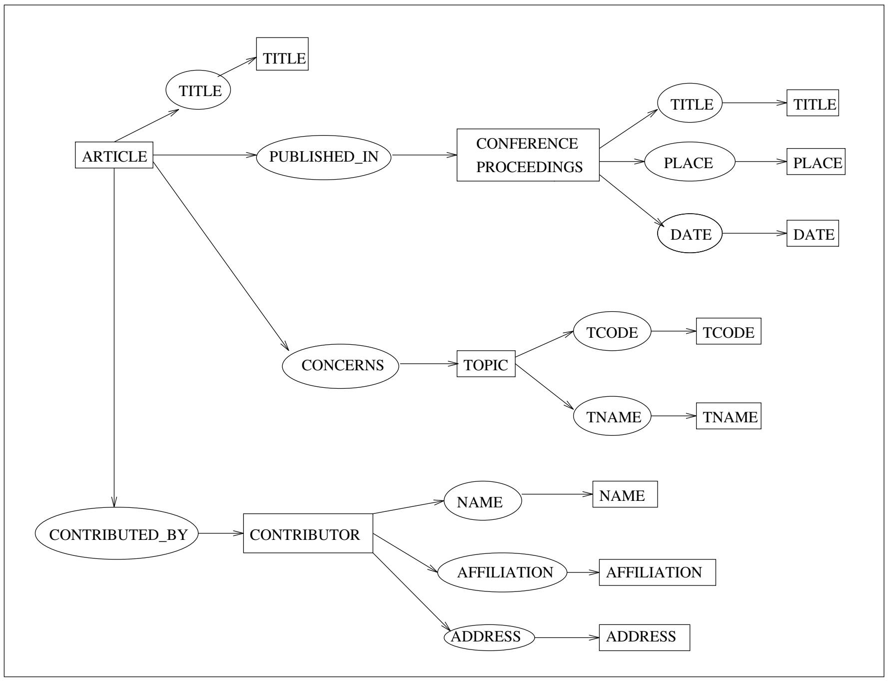

FIGURE 2.6. The equivalent conceptual schema graph of figure 2.2

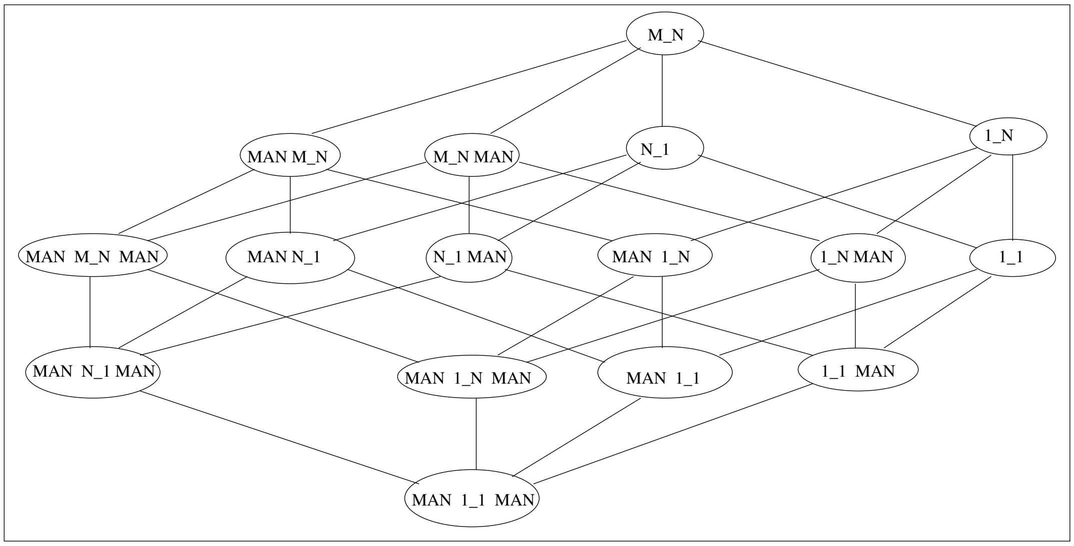

FIGURE 2.7. The ER constraints lattice

2.5 Summary and Conclusions

We have discussed the schema integration problem and noted some of the formalising that has been attempted in order to solve the problems. The framework for a canonical data model is close to that of the IRDS conceptual schema, for which a proposal has been made to use conceptual graphs. We have shown that conceptual graphs can be used in the integration process, particularly where an ontology is provided.

3

CCAT: The Current Status of the Conceptual Catalogue (Ontology) Group, with Proposals

Fritz Lehmann[1]


```
%================================================
%    P E I R C E  W o r k b e n c h  U n i t
%
% NAME: CCAT.workshop94
% TITLE: CCAT: The Current Status of the Conceptual
%        Catalogue (Ontology) Group, with Proposals
% TYPE: Chapter in PEIRCE Workshop Proceedings at
%       Int. Conf. on Conceptual Structures - 1994
% SUBSYSTEM: Documentation
% PURPOSE: To report on current status (desuetude)
%          of CCAT and to propose future direction.
%          Appendix has a list of concept-systems.
% AUTHOR: Fritz Lehmann (fritz@rodin.wustl.edu)
% DATE: July 28, 1994
% LANGUAGE: LaTeX
% REFERENCES: in this text at end
%=================================================
```

3.1 What CCAT Is Supposed To Be

The CCAT group in the PEIRCE Project is charged with creating an **ontological repository** — a taxonomic catalogue of useful concepts formally defined in conceptual graphs. This amounts to creating a concept-system for the real world. The CCAT group is different from other groups in the PEIRCE project because CCAT is not about representation or implementation, rather it is about the actual content that needs to be represented.

Sowa's theory of Conceptual Graphs (CG) [Sowa 84] is more than a variant of First Order Logic. It includes the idea that the logical predicates (concepts and relations) should have expanded conceptual definitions based on a set of semantically simple near-primitives (which relate to natural language); in particular it requires the explicit use of a *starter-set* of conceptual "deep case relations" like AGENT, INSTRUMENT, and PATIENT existing between an event and its participants. **Appendix B** of [Sowa 84] had lists of potentially useful basic concepts and relations. A logically acceptable assertion like `[BRUTUS]-->(KILLED)-->[CAESAR]` is not acceptable as a fully expanded CG; it would be expanded at least to:

```
[SITUATION: [KILL]-->(AGT)-->[PERSON: Brutus]
                  -->(PTNT)-->[PERSON: Caesar] ]-->(PTIME)-->[TIME: Past]
```


[1] *GRANDAI* Software, 4282 Sandburg Way, Irvine, CA 92715, USA
Tel: (314) 733-0566   FAX: (714) 733-0506
Email: `fritz@rodin.wustl.edu` or `flehmann@orion.oac.uci.edu`

CG theory generally eschews "hyphen-based ontology" — say, a relation like `Date-Of-Manufacture` which makes sense as English words without having a genuine conceptual definition of what it is to be a `Date`, to be `Manufactured` on a certain `Date`, and so on. CCAT definitions are to be at a deeper level, and will include a *canonical basis* constraining the types of participants in relations. [Sowa 84]

The job of CCAT is thus enormous at the start and never-ending. CCAT needs to build not only a generic ontology for the common-sense world, but also some more specific concepts as targets for certain practical projects now underway by various PEIRCE participants. The results of CCAT will be needed for: natural language, semantic database integration, intelligent agent communications, semantics for international standards like PDES/STEP and EDI, deep-knowledge rule-based expert systems, relevance-based information retrieval, semantic modelling of control systems, and others.

Fortunately this CCAT work will not occur in a vacuum. There are many other efforts now under way to develop the kind of concept-systems needed for CCAT. A critical issue for CCAT will be the means of importing ontologies created by others into ours. This involves the crucial but as yet little-explored field of knowledge integration. (See below.) Conversely, the CCAT core ontology will be available to users and researchers electronically by FTP, World Wide Web, etc.

3.2 Where We Are Now

It is not possible to report on the progress of the CCAT group, because there has been no progress. In fact I would almost say that there has been no CCAT group. I am speaking informally for CCAT only because none of the designated members (including me) has become the CCAT coordinator. At last year's ICCS-93 conference in Quebec, I convened an informal after-dinner meeting on ontologies, without an agenda. About 15 people attended. In the discussion, the only CCAT "decisions" I remember were the following, by vague consensus: 1. Put **Walling Cyre's system of time** in the "core ontology". 2. Put near-equivalents of the case-relations and basic concepts from **Appendix B** of [Sowa 1984] in the "core". 3. Begin giving the **PDES/STEP** (manufactured objects standards) people the **target-concepts** they need. 4. Coordinate with Tom Gruber's **ARPA KSE/SRKB/Ontolingua ontologies** (frames, bibliography, measurements, [elevators?] and others). 5. Coordinate with **IRDS Conceptual Schema** (for databases) standard. 6. Try to accommodate the **Penman Upper Model** categories for English. 7. See if **CYC** is interested. (These are listed in decreasing order of my certainty of memory). As far as I know, nothing has happened with CCAT since that meeting.

No PEIRCE subgroup can function without a coordinator willing and able to spend some time on it. CCAT more than other groups will require a prodigious amount of work before it has a "core" of concepts useful to others. The future of CCAT is in doubt.

3.3 One Ontology Or Many Ontologies?

An early issue on the email list was whether CCAT should be a single, monolithic ontology for all users, or a collection of different concept-systems suited to different purposes. **Doug Skuce** at the University of Ottawa has worked for years on a generic ontology for the world and he at one point seemed to favor the single-ontology approach. I countered this with a call to admit several different concept systems needed for different purposes. My slogan is *"Purpose Dictates Ontology."* Different users need different kinds of concept system, although there are widely common purposes and therefore large areas for ontological consensus. At the meeting last year we discussed the compromise approach of having two kinds: **A. a "core" ontology** for the common world, to which **B. various other special-purpose ontologies** can be attached. No one will be required to use the core (competing *general ontologies* could be maintained outside the core.) No non-core concept systems will be rejected from the CCAT repository, so long as they are syntactically correct. We will need to make official decisions as to what gets in the core, particularly to minimize logical inconsistency within the core. The Appendix below is a list of 157 concept-systems I've encountered so far.

3.4 Consensus About The Core Ontology

The consensus was that the core ontology should include at the start **Walling Cyre's time system** and near-equivalents of the case-relations and basic concepts from **Appendix B** of [Sowa 1984].

Walling Cyre's time system is described in "Acquiring Temporal Knowledge of Schedules" in [Mineau et al. 1993], p.328. This system, although originally designed for circuit switching analysis, is quite comprehensive and includes a hierarchy of several notions of time (intervals, points, relations, etc.) used in other work and it is applicable to common sense time as well. I recommend implementing (a basic subset of) existing international **standards for time and dates** in terms of Cyre's system, thus providing those standards with a formal temporal semantics. This will require some concepts of physical objects, days and years, different calendars, Greenwich Mean Time, etc.

In **John Sowa's Appendix B**, concepts and relations were listed as suggestions rather than as a comprehensive set of required elements of a conceptual graph system. Some of them have close analogues in natural language grammar (such as POSS for possession) and assigning a particularized conceptual definition to these would destroy the clear connection. I suggest using these as common subsumers of families of more precisely defined concepts and relations, rather than attempting to define them all with precision directly. Allowance can be made for metaphoric meaning change, and for "family resemblances" à la late Wittgenstein. Examples of concepts are ACT, AGE, ANIMATE, BELIEVE, COLOR, MAKE, PHYSOBJ, RECEIVE, WEAR-OUT. Examples of relations are AGENT, CAUSE, CHILD, EXPERIENCER, INSTRUMENT, LOCATION, PART, POSSESSION, RESULT, SUPPORT.

Other candidates for early addition to the core would be **systems for measurement units**. Two currently implemented systems are those of Tom Gruber at Stanford and Nicholaas Mars at the University of Twente.

The generic high-level ontology of **Doug Skuce** is probably fairly compatible with these systems, and could be the platform on which other structure is built. I don't know whether this is compatible with the **situational ontology of Bill Tepfenhart** [Tepfenhart 1992]; Sowa has included the situation approach in CG theory. An underlying **physical ontology** could be based on that of Roger Hartley, Heather Pfeiffer and Dan Eshner [Pfeiffer et al. 1992, Hartley 1992].

Beyond these near-primitives lies the area of event types, including the standard ordinary event types which are known to occur in real world histories and stories. There has been work on classifying these. Gian Piero Zarri's RESEDA [Zarri 1992] is one example, and similar work is now being done in efforts to provide intelligence to interactive fiction programs.

Although CCAT is not reponsible for natural language interfacing, I would recommend that the sophisticated language-based **Penman Upper Model** be used to tag each concept with its most specific Penman class. [Bateman et al. 1990] The Penman Upper Model is a conceptual taxonomy which seeks to distinguish every semantic difference which affects English grammar. It has over 200 concept classes.

3.5 Alliances With Other Ontological Knowledge Systems

Building concept systems is laborious. In the common-sense realm, **the CYC project** was to cost $35 million and take ten years for a dozen or so knowledge enterers. Other major conceptual work is being done at Stanford and USC/ISI in connection with the **ARPA Knowledge Sharing Effort (KSE)**. The PEIRCE project is not in a position to duplicate these efforts. I suggest that a top priority for CCAT is to link with outside sources of machine ontologies as soon as possible. One form of Conceptual Graphs will soon be standardized so as to map directly to the KIF language (Knowledge Interchange Format) of the KSE. This would allow knowledge to migrate to CCAT from Ontolingua ontologies, from knowlege bases expressed in LOOM and CLASSIC, and from other compatible systems; it would also make CCAT knowledge developed by PEIRCE participants available to the other systems. The CYC "Epistemological Language" (EL) is not too far from KIF and CG's, and it may be possible later to connect to CYC and translate to and from CGs automatically. This depends on whether Doug Lenat at CYC will ever be free about releasing its ontology (to date, access has been very limited, with much attention given to security and piracy). Tom Gruber at Stanford is amassing a very promising set of specialized, freely-available ontologies expressed in KIF and Ontolingua. Maybe some of these could be added initially to the CCAT core, while CCAT contributions could be reqired to conform to Gruber's rules and tests for Ontolingua ontologies.

The **PANGLOSS** system at ISI under Eduard Hovy has an "ontology base" of about 50,000 concepts organized under the Penman Upper model. It is to be used for natural language generation and translation. The creators of this base also exploited the ONTOS ontology of Nirenburg, the Wordnet meanings list at Princeton, the semantic markers in the Longman's Dictionary, and other sources. Kevin Knight of ISI

stated publicly at his AAAI-94 talk that the ontology base is freely available except for the versions which include information from Longman's. This is an IS-A hierarchy, without explicit conceptual definitions.

Ontek, Inc. is developing the ontology necessary to describe information and data systems. That work is closely tied to the ANSI **IRDS** (Information Resource Dictionary System) **Conceptual Schema standard**. Ontek's approach is close to CG's in that Ontek troubles to build its concepts on a deep philosophical foundation. At present it is not clear what part of Ontek's work is proprietary, but it may be that the main ontology will be freely shared, or that Ontek might agree to work with the Peirce project CCAT group directly.

There are **other companies with ontologies** that are presumably proprietary, such as G.E., Intelligent Text Processing Inc., Siemens-Nixdorf, Oracle, EIT, et al., but they too may be willing to share pars of their knowledge bases (or even to fund some specialized CCAT projects).

There is an enormous investment of mental labor in hundreds of existing taxonomies and thesauri for different specialized fields. These are studied by **ISKO (International Society for Knowledge Organization)** in Germany . Most of these systems have agreed-upon, reliable taxonomies of classes, but without conceptual definitions. In medicine one of the biggest efforts is the UMLS Metathesaurus developed at the National Library of Medicine. This has over a hundred thousand classes and terms in dozens of interacting hierarchies. It should be possible to interface such specialized thesauri to the "core" of CCAT.

National and international **data standards** are continually appearing throughout the world; most have carefully defined syntax but merely informal semantics based on English descriptions of data fields. They contain vast amounts of knowledge but are exceedingly "brittle". Programs using these standards cannot adjust to communicate with systems using different versions or subsets because there is no machine notion of what the fields and structures actually mean. **EDI** (Electronic Data Interchange) is one important example; it is now being used by 70,000 businesses for electronic commerce (purchasing, etc.) but it is held back by the need for human beings to set up each transaction protocol required by a different trading partner (most "E"DI is reportedly carried on by mailed paper and Fax!). Such standards need conceptual definitions of the structured components. There are enormous amounts of money involved in these fields. In a slight advance on this, the **PDES/STEP** standards for defining specifications of manufactured products is based on the EXPRESS language. Alex Bejan and Michael Wermeliger have worked on providing Conceptual Graph semantics for EXPRESS and STEP. Jeff Van Baalen is working to translate EXPRESS to KIF. What the EXPRESS/PDES/STEP workers need now are the target-concepts to translate into. This dictates a rich and varied ontology motivated by STEP alone.

Similarly, the key to the field of "**Enterprise Integration**" is an ontology appropriate for modelling business entities and transactions. At present, business does not really know which proposed enterprise model to use.

Another possible alliance would be with the French RESEDA project at CERTAL and perhaps with the interactive fiction theorists on the WISDOM email list.

3.6 Integration of Disparate Systems

The missing piece of this picture is the method for *ontological integration.* If CCAT is to bring in all kinds of different knowledge sources, the different formats, languages and definitions will need to be mapped to and from one another. This is among the most challenging research areas today. It is known to be crucial, and hard. There two levels of this: first, there is ontological integration of different databases and knowledge bases. This uses a common ontology as the tool to integrate data systems. Second, there is integration of different ontologies. That is the problem for CCAT if it is to go the "many ontologies" route. The current situation is that there are very few tools or theories for integrating two different ontologies. There are some theories for computer-assisted database schema integration (by hand), and I have been working on some promising lattice-based techniques ("EGG/YOLK Theory" and "Tricept Lattices") for integrating ontologies having formal definitions of classes, using incidence relations between classes and exemplars.

Logical inconsistency is a constant threat; even if two systems are each internally consistent, they may contradict each other in subtle ways, and no way is now known to detect this. Even if it is detected, there is no consensus on what to do about it. Lots has been proposed, but nothing proved working in the field as far as I know. This is a ripe area for research, at the heart of the *knowledge sharing* problem. For CCAT it will have to be done somehow. I expect that the CCAT participants will use a combination of machine assistance and human judgement.

3.7 Theoretical Issues

There are many theoretical issues for CCAT in addition to those just mentioned. The large-scale hierarchic structure of the ontology is unknown. An important question is when an ontology is "faceted" or decomposable into separate *ontological dimensions* like height, color, strength, etc. Issues include mereology (the nature of the PART-WHOLE relationship), the preservations of qualities along relation links (like the `TransfersThrough` rules in CYC), higher-order qualities of qualities, "slot inheritance", the graph-grammar structure of dictionary expansions, classification theory, causality, conceptual approximation, metaphor, salience, relevance, formal concept analysis, category-level theory, granularity and more. These are ripe areas for research on hot topics just now emerging in importance.

Other important theoretical issues concern the methodology of taxonomy building and knowledge acquisition; there is a workshop on this at this conference.

3.8 The future of CCAT in the PEIRCE Project

Without an active coordinator and better commitment by interested participants, CCAT is going nowhere.

It would be possible to simply drop CCAT from PEIRCE and let PEIRCE users develop their concept-systems completely independently; this would still provide the "workbench" for the syntax and (logic-equivalent) semantics of conceptual graphs, without a common canonical basis or the notion of appropriate level of conceptualization.

There are reasons not to drop CCAT, however.

- Knowledge sharing without ontology sharing is simply impossible. This means that the PEIRCE workbench would not act as a conduit for the flow of knowledge without CCAT. Instead, people might develop in PEIRCE and use other formalisms (coupled to generic ontologies) to share their knowledge.

- CCAT would provide an opportunity for meaningful research in concept-systems. Universities now have little use for the concept-system builder; the work is not deemed theoretically important in computer science departments. (It may be appreciated in some Library Science departments for classification systems.) CCAT would provide a niche and a sophisticated interested audience for this kind of work.

- Times are changing. The field of Knowledge Representation has matured somewhat, and top people are shifting attention to what is represented, rather than how it is represented. When CGs, KIF, CYC's EL, SNePS, LOOM, CLASSIC, HAT, HOL and other languages can intercommunicate, the interesting questions will be about the structure and contents of what is represented. Very few people out there are now proposing new, general knowledge representation languages.

Aside from natural languge and inference structures, the move is on to where the future action will be, and that is ontologies.

- Having CCAT will greatly enlarge the number of people interested in PEIRCE. People are perpetually curious about the CYC project — this approach inspires a lot of general interest (outside of CS faculty advisors' circles). For many people, CCAT is the *raison d'etre* for a project like PEIRCE. A good CCAT is what would make PEIRCE directly useful to the industries and standards efforts mentioned earlier.
- A conceptual catalogue with a certain level of conceptualization of relations (conceptual quasi-primitives) was part of Sowa's original idea of Conceptual Structures.

3.9 Recommendations

I think it's fair to say that CCAT is not a primary research interest of PEIRCE leader Gerard Ellis nor (probably) of Bob Levinson at the moment. Work on a voluntary project like PEIRCE depends on where the interest lies, and rises and falls accordingly. I recommend that either:
A. Someone with a genuine interest in CCAT step forward who has the institutional connection and time to devote to it, to act as coordinator and quickly get it in shape and moving forward, or
B. Officially drop CCAT from the PEIRCE project or put it on indefinite hold.

If PEIRCE has a conceptual catalogue at all, many people will judge PEIRCE on that, rather than on its workbench implementation, so it is important that it be a respectable effort.

3.10 Appendix: List of Known Concept-Systems

This is the latest version of a list of concept-systems which I promulgate by email every now and then. This exhibits the wide variety of potentially useful sources of knowledge structures. Not all of these entries are likely to be useful — but many of the later entries on the list are.

```
     [Version 4: Thanks to Piet-Hein Speel, Paul Doudna, Nick Youd,
Jim Fulton, and especially to Dan Fass for COPIOUS new material. Changes:
RICHENS/MASTERMAN/WILKS is now divided into: RICHENS 100 MINIMALS (he
invented "semantic nets" in 1956), MASTERMAN'S SEMANTIC LATTICES,
PREFERENCE SEMANTICS PRIMITIVES (Wilks), and FASS'S GENUS CLASSIFICATION.
The RELATIONAL LEXICON HIERARCHY becomes EVENS/NUTTER LEXICAL RELATION
HIERARCHY.  New total= 158 concept-systems.      FL, Jul 11 94]
=================================================================

                    CONCEPT-SYSTEMS CATALOGUE

Fritz Lehmann
4282 Sandburg Way, Irvine, CA 92715 USA (714)733-0566
[occasionally accessed email: fritz@rodin.wustl.edu]

November 1993

This is to be an informal catalogue of existing concept
catalogues, taxonomies and hierarchies (including high level
"ontologies") for possible use in knowledge representation,
artificial intelligence, simulation, and database integration.
Anybody can contribute (and be acknowledged).  Each concept system
is to be described (in a page or less) with some references and other
information.  I hope to be inclusive, with emphasis on potentially
machine-readable/usable concept (and relation) hierarchies.
```

Some people think there is ONE concept system for the true structure of the world. Others like me think pragmatic concerns (subjective, mission-determined, or socially agreed-upon) may dictate different structures. Most "ontologies" have large areas of near-agreement on concepts like time, space, individuals, properties, etc. Technical thesauri deal with more specific subject areas like accounting, subfields of medicine, or plumbing fixtures. Philosophical concepts are necessary but controversial; some concepts like "check-stub" are quite uncontroversial.

The list is now intentionally a grab-bag. It ranges from universal to fairly problem-specific, informal to formal. Formalized or not, two aspects of every system are: its purely mathematical (order) structure, and the meanings of its components. Notation or language is incidental to both. A concept in one system may differ entirely from a concept with the same name in another system.

The page ordering, for now, is vaguely chronological.
Please let me know of ANY OTHER concept-systems you know about.

[I started this list Nov. 20, 1992 for the "PEIRCE project" (a cooperative international implementation of a Conceptual Graphs inferential database processing sytem, initiated by Gerard Ellis and Robert Levinson), with a list of 84 systems beginning with Aristotle's. CODES: (i+)= I now have a little information; (i-)= I have almost no information; (I)= I have enough information; (p+)= Page written; (p-)= Page not written; (nr)= Need references; (nd)= Need descriptive documents; (c=)= Contributed information, besides me.]

ARISTOTLE'S CATEGORIES	(i+,p-,nr)
LLULL'S ARS MAGNA	(I,p+)
LEIBNIZ'S ARS COMBINATORIA & CHAR. UNIVERSALIS	(I,p+)
LODWYCK'S COMMON WRITING	(i+,p-,nd)
DALGARNO'S ARS SIGNORUM	(i-,p-,nr,nd)
WILKINS'S PHILOSOPHICAL LANGUAGE	(I,p-)
LINNAEUS BIOLOGICAL TAXONOMY	(i+,p-,nr)
(ANONYMOUS, c. 1830) UNIVERSAL CHARACTER	(I,p-)
CAVE BECK	(i-,p-,nr,nd)
KANT'S A PRIORI SYNTHETIC CATEGORIES	(i+,p-,nr,nd)
ROGET'S THESAURUS	(I,p-)
PEIRCE'S CATEGORIES (1,2,3; 66 SIGNS, ETC.)	(I,p-,nr)
BOLZANO	(i+,p-,nr,nd)
MEINONG	(i-,p-,nr,nd)
BRADLEY	(i-,p-,nr,nd)
DEWEY DECIMAL, BLISS, UDC & LIBRARY OF CONGRESS	(I,p-)
POLTI'S 36 DRAMATIC SITUATIONS	(i-,p-,nr,nd,c=Barger)
HUSSERL'S ONTOLOGY	(i+,p-,nr,nd)
STOPP'S FOLKTALE ENCODINGS	(i+,p-,nd)
PRINCIPIA MATHEMATICA	(I,p-,nd)
WHITEHEAD'S PROCESS THEORY	(i+,p-,nd)
LIESNIEWSKI'S MEREOLOGY & SO-CALLED ONTOLOGY	(i+,p-,nr,nd)
OGDEN & RICHARDS' BASIC ENGLISH	(I,p-,c=Fass,Thorson)
SEMANTOGRAPHY/BLISSYMBOLICS SYMBOLS	(I,p-)
RICHENS'S 100 "SEMANTIC NET" MINIMALS	(i+,p-,nr,nd,c=Fass)
CECCATO'S CORRELATION NET PRIMITIVES	(i-,p-,nr,nd)

```
MASTERMAN'S SEMANTIC LATTICES                        (i+,p-,nr,nd)
LINCOS INTERPLANETARY LANGUAGE                     (I,p-,c=Godden)
R.M. MARTIN'S SEMIOTIC PRIMITIVES                    (i+,p-,nr,nd)
COLON FACETED LIBRARY CLASSIFICATION                 (i+,p-,nd)
THE SYNOPTICON (FOR ENCYC. BRIT. GT BOOKS)    (I,p-,c=Salsman)
DEEP CASE SYSTEMS                                    (i+,p-,nd)
LOGLAN/LOJBAN SEMANTIC PRIMITIVE WORD ROOTS          (I,p-)
INGARDEN'S ARISTOTLE REVISION                        (i-,p-,nd)
LAFFAL'S CONCEPT DICTIONARY                          (I,p-)
LEECH'S SEMANTICS                             (i-,p-,nr,nd,c=Fass)
SCHANK'S CONCEPTUAL DEPENDENCY THEORY                (I,p-)
STITH-THOMPSON FOLK MOTIFS                  (i-,p-,nr,nd,c=Barger)
ACM COMPUTER SCIENCE CLASSIFICATION                  (i-,p-,nr,nd)
SHUM "SPIRITUAL" NETWORKS                            (i-,p-,nr,nd)
ANTHROPOLOGICAL CLASSIFICATIONS                      (i-,p-,nr,nd)
PROPAEDIA OF ENCYLOPAEDIA BRITANNICA              (I,p-,c=Van Roy)
WEBER RUSSELL'S CATEGORIES OF NOMINALS          (i-,p-,nd,c=Fass)
PARKER-RHODES' INFERENTIAL SEMANTICS LATTICES        (I,p-)
WIERZBICKA'S LINGUA MENTALIS                         (i+,p-,nd)
SCHEELE'S ORDNUNG DES WISSENS                 (i-,p-,nr,nd,c=ISKO)
PATENT CLASSIFICATION SYSTEMS                 (i-,p-,nr,nd,c=ISKO)
NTIS/DOD/COSATI CLASSIFICATION SCHEME         (i-,p-,nr,nd,c=ISKO)
UNESCO THESAURUS                              (i-,p-,nr,nd,c=ISKO)
BROAD SYSTEM OF ORDERING                      (i-,p-,nr,nd,c=ISKO)
RUSSIAN MISON "RUBRICATOR" CLASS. CODES       (i-,p-,nr,nd,c=ISKO)
BHATTACHARYA'S CLASSAURUS                            (i-,p-,nr,nd)
AUSTIN'S PRECIS CONCEPT ANALYSIS & INDEXING          (i-,p-,nr,nd)
KAMP'S DISCOURSE REPRESENTATION STRUCTURES           (i-,p-,nr,nd)
PREFERENCE SEMANTICS PRIMITIVES                 (I,p-,c=Wilks,Fass)
MILLER/JOHNSON-LAIRD PRIMITIVES               (i+,p-,nr,nd,c=Fass)
HAYES'S NAIVE PHYSICS                                (i+,p-,nr,nd)
LEHNERT'S OBJECT PRIMITIVES                     (i+,p-,nd,c=Fass)
SCHAN/CARBONELL SOCIAL/POLITICAL ACTS          (i+,p-,nd,c=Fass)
MEL'CHUK'S EXPLANATORY-COMBINATORY DICTIONARY        (i+,p-,nd)
MeSH - MEDICAL SUBJECT HEADINGS THESAURUS            (i-,p-,nd)
JOLLEY'S HOLOTHEME                                   (I,p-)
ZARRI'S RESEDA ONTOLOGY                              (I,p-)
LENAT'S AM/EURISKO MATH CATEGORIES                   (I,p-)
SOWA'S CONCEPTUAL GRAPHS PRIMITIVES                  (I,p-)
BARWISE/PERRY/DEVLIN SITUATION SEMANTICS             (I,p-)
SCHUBERTIAN ("ECO") SUBHIERARCHIES                   (I,p-)
FICTION CLASSIFICATION SCHEMES                (i+,p-,nr,nd,c=ISKO)
WAHLIN'S T.I.M, MANUFACTURING FACETS            (i+,p+,nd,c=ISKO)
CITIZENS ADVICE BUREAU CLASSIFICATION UK             (I,p-,c=ISKO)
QUALITATIVE PHYSICS PRIMITIVES                       (i-,p-,nr,nd)
SMITH-MULLIGAN ONTOLOGY                              (i+,p-,nr,nd)
SIMONS' PART SYSTEM                                  (i-,p-,nr,nd)
SMALLTALK DATA TYPE TREE                             (i-,p-,nr,nd)
OBJECTIVE-C (NeXTSTEP) DATA TYPE TREE                (i-,p-,nr,nd)
BOOCH/RATIONAL OBJECTS HIERARCHIES                   (i-,p-,nr,nd)
WUESTER'S GENERAL THEORY OF TERMINOLOGY         (i+,p+,nd,c=ISKO)
KAB GERMAN LIBRARY CLASSES (K.LEHMANN)          (i-,p-,nd,c=ISKO)
GRAESSER'S MULTIPLE CONCEPT HIERARCHIES              (i+,p-,nd)
DAHLGREN/McDOWELL NAIVE SEMANTICS                  (i+,p-,c=Youd)
LONGMAN DICTIONARY CODINGS (INCL. SLATOR)            (i+,p-,nr,nd)
```

```
LONGMAN LEXICON (THESAURUS)                       (i-,p-,nr,nd)
BURGER'S THE WORDTREE                             (I,p-)
FASS'S GENUS CLASSIFICATION                    (I,p-,nd,c=Fass)
PENMAN UPPER MODEL                                (I,p-,c=Hovy)
UNIFIED MEDICAL. LANG. SYS.(UMLS) METATHESAURUS (i-,p-,nr,nd)
ICONCLASS ART SUBJECT CLASSIFICATION              (i+,p-)
SPARCK JONES/BOGURAEV DEEP CASE LIST              (I,p-)
COOK ONTOLOGY                                     (i-,p-,nr,nd)
DIXON ONTOLOGY                                    (i-,p-,nr,nd)
VARIOUS WILLE-STYLE FORMAL CONCEPT LATTICES       (i+,p-,nr,nd)
RUSSIAN MERONOMY/TAXONOMY (SHREIDER ET AL.)       (i-,p-,nr,nd)
SOMERS'S CASE GRID                                (i+,p-,nd)
CHAFFIN'S RELATION HIERARCHY                      (i+,p-,nd)
LENAT/GUHA CYC PROJECT                            (i+,p-,nd)
EPSTEIN AM-BASED GRAPH THEORY HIERARCHY           (i-,p-,nd)
MARTY'S SEMIOTIC LATTICES                         (i-,p-,nr,nd)
GEOGRAPHIC DATABASE CONCEPTS                      (i-,p-,nr,nd)
MAPSEE GEOGRAPHIC MAP AXIOMS (MACKWORTH/REITER) (i+.p-,nd)
IRDS DATABASE CATEGORIES                          (i-,p-,nr,nd)
HOBBS' COMMONSENSE ONTOLOGY                       (I,p-)
GORANSON SYMMETRIES                               (i-,p-,nr,nd)
PANSYSTEMS PHILOSOPHICAL LOGIC (CHINESE)          (i+,p-,nd)
GIUNCHILIA'S ITALIAN PREPOSITIONS                 (i-,p-,nr,nd)
LILOG ONTOLOGY                              (i-,p-,nr,nd,c=Speel)
VELARDI'S SEMANTIC LEXICON FOR ITALIAN            (i-,p-,nr,nd)
ONTEK, INC.'S ONTOLOGY                            (i-,p-,nd)
LAKOFF'S CATEGORIES                              (i+,p-,nd)
HUHNS/STEPHENS RELATION FEATURES                  (i+,p-,nd)
WORDNET                                     (I,p-,c=Consortium)
EDR CONCEPT DICTIONARY (JAPAN)                    (i-,p-,nr,nd)
ONTOS CONCEPT HIERARCHY   (Nirenburg)             (i-,p-,nr,nd)
DOUDNA QUANTIFIER RHOMBIDODECAHEDRON            (I,p-,c=Doudna)
SCRAMBLED ROGET (5th Ed.)                         (i-,p-,nr,nd)
NIRENBURG'S DIONYSUS ONTOLOGY                     (i-,p-,nr,nd)
SCHUBERT/HWANG EPISODIC LOGIC CATEGORIES          (i+,p-,nd)
UNITRAN-LCS (DORR Et AL.)                         (i-,p-,nr,nd)
EVENS/NUTTER LEXICAL RELATION HIERARCHY           (i+,p-,nd)
PUSTEJOVSKY'S QUALIA/EVENTS                  (i-,p-,nr,nd,c=Fass)
FULTON'S SEMANTIC UNIFICATION METAMODEL        (i-,p-,c=Fulton)
RANDOM HOUSE WORD MENU CATEGORIES                 (i-,p-)
PANGLOSS ONTOLOGY BASE                            (i-,p-,nr,nd)
SNePS DICTIONARY OF CASE FRAMES(INCL.YUHAN)(I,p-,c=Chalupsky)
YALE'S ESPERANTO THESAURUS                        (i-,p-,nr,nd)
DARPA/ROME PLANNING ONTOLOGY                      (i-,p-,nr,nd)
CIMOSA BUSINESS ENTERPRISE ARCHITECTURE MODEL   (I,p-)
ICAM MANUFACTURING REFERENCE MODEL                (i-,p-,nr,nd)
IWI (GERMAN) MANUFACTURING REFERENCE MODEL        (i-,p-,nr,nd)
EDI (ELEC. DATA INTERCH.) BUSINESS COMM STANDARD(i-,p-,nr,nd)
GUARINO CONCEPT/RELATION ONTOLOGY                 (i+,p-,nd)
SKUCE ONTOLOGY                                  (I,p-,c=Sarris)
PETRIE ONTOLOGIES                                 (i-,p-,nr,nd)
TEPFENHART ONTOLOGY                               (i-,p-,nd)
G.E. SEMANTIC HIERARCHY & LEXICON (RAU ET AL.)  (i-,p-,nr,nd)
PLINUS CERAMICS ONTOLOGY                        (I,p-,c=Speel)
MARS' KELVIN MEASUREMENT HIERARCHY          (i-,p-,nr,nd,c=Speel)
```

```
    RANDELL & COHN'S SPATIOTEMPORAL LATTICES        (I,p-)
    PDES/STEP PRODUCT DESCRIPTION STANDARDS         (i+,p-,nr,nd)
    HARTLEY'S TIME AND SPACE WORLD                  (i+,p-,nd)
    ONTOLINGUA-KIF FRAME ONTOLOGY             (i+,p-,nd,c=Gruber)
    GRUBER'S QUANTITIES/UNITS ONTOLOGY        (i-,p-,nd,c=Gruber)
    GENERIC BIBLIOGRAPHY CONCEPT SYSTEM       (i-,p-,nd,c=Gruber)
    DICK'S CASE-RELATION SYSTEM FOR LEGAL ANALYSIS  (I,p-,c=Dick)
    HORN'S SUBJECT MATTER CATEGORIES             (I,p-,c=Horn,nd)
    ARNOPOULOS' SYSTEM UNIFICATION MODEL (SUM)      (I,p-,c=ISKO)
    TENENBAUM/EIT MANUFACTURING ONTOLOGY            (i-,p-,nr,nd)
    INTELL. TEXT PROCESSING INC. SEMANTIC LEXICON   (i-,p-,nr,nd)
    TADS - TEXT ADVENTURE DESCRIPTION LANGUAGE      (i-,p-,nr,nd)
    BARGER'S CONCEPTUAL FRACTAL THICKET THEORY      (i-,p-,nr,nd)
    ORACLE "CONTEXT" CONCEPT TREE                   (i-,p-,nr,nd)
    NEXTSTEP ENTERPRISE OBJECTS MODEL               (i-,p-,nr,nd)
    BONGALON ONTOLOGY                               (i-,p-,nr,nd)
    DOUDNA'S TOTO (THE ONE TRUE ONTOLOGY)                 (i-,p-)

[Hundreds of special-area thesauri are listed in Int. Classification
 and Indexing Bibliography (ICIB) published by Indeks-Verlag, at the Int.
 Soc. for Knowledge Organization, Woogstr. 36a, D-6000 Frankfurt 50, Germany]

SUGGESTED FORMAT: Brief description, Examples, Formalized?,
Abstract hierarchy structure, Maximum depth, Necessary/sufficient?,
Current authorities/enthusiasts, Machine-readable text?, Machine-
usable structure?, Source, FTP site?, Implemented? References.

=========================================================================
```

3.11 References

[Bateman et al. 90] John A. Bateman, Robert T. Kasper, Johanna D. Moore and Richard A. Whitney, "A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model," Tech. Report, ISI, 4676 Admiralty Way, Marina del Rey, cal. 90292-6695, March, 1990.

[Hartley 92] Roger T. Hartley, "A Uniform Representation for Time and Space and Their Mutual Constraints," in [Lehmann 92]

[Lehmann 92] Fritz Lehmann, Ed., *Semantic Networks in Artificial Intelligence,* Pergamon Press, Oxford, 1992; also in *Computers and Mathematics with Applications,* v.23, no.s 2-9, 1992.

[Mineau et al. 93] Guy Mineau, Bernard Moulin and John Sowa, Ed.s, *ICCS'93 Conceptual Graphs for Knowledge Representation,* Springer LNAI 699, Berlin 1993.

[Nagle et al. 92] Timothy E. Nagle, Janice A. Nagle, Laurie Gerholz and peter W. Eklund, Ed.s,*Conceptual Structures Current Research and Practice,* Ellis Horwood, New York, 1992.

[Pfeiffer & Hartley 92] Heather D. Pfeiffer & Roger T. Hartley, "The Conceptual programming Environment, CP," in [Nagle et al. 92]

[Sowa 84] John F. Sowa, *Conceptual Structures,* Addison-Wesley, Reading, Mass., 1994.

[Tepfenhart 92] William Tepfenhart, "Using the Situation Data Model to Construct a Conceptual Basis Set," in [Nagle et al. 92]

[Zarri 1992] Gian Piero Zarri, "The 'Descriptive' Component of A Hybrid knowledge Representation System," in [Lehmann 92].

4

LEARN:Exploiting the Physics of State-Space Search with MorphII

Robert Levinson[1]

ABSTRACT This chapter is a blueprint for the development of a fully domain-independent single agent and multi-agent heuristic search system. The paper gives a graph-theoretic representation of search problems based on conceptual graphs, and outlines two different learning systems. One, an "informed learner" makes use of the the graph-theoretic definition of a search problem or game in playing and adapting to a game in the given environment. The other a "blind learner" is not given access to the rules of a domain, but must discover and then exploit the underlying mathematical structure of a given domain. These learning agents are based on generalizing the understanding obtained with the Morph chess system to all games involving the interactions of abstract mathematical relations. An effective monitor for such domains is presented, along with implementations of blind and informed learners. Performance results are very encouraging. **Keywords:** games, mathematical structure, heuristic search, machine learning, hypergraphs, neural networks, analogical reasoning, RETE, relational patterns, hierarchical reinforcement learning

Introduction

We suggest the following tenets or guiding principles for developing machine intelligence:

1. Intelligence is optimal problem-solving in pursuit of specific goals under resource constraints. (Note that no mention is made of human cognition or consciousness).
2. Given this definition, domain-independence and adaptability are fundamental aspects of intelligence. (To not adapt, if economical, is sub-optimal)
3. Single and multi-agent state space search problems are a large and important class of problems in which to develop and study machine intelligence
4. The task before us is to develop fully domain-independent methods for working in state-space search domains.
5. The knowledge required to perform well in these domains is embodied in the definition of the domain and the mathematical structure of the state space. *I.e.* the degree that a heuristic is good is explainable within the framework of the mathematics of the space.
6. Experience in a state space reveals mathematical structure that can be exploited by an optimal problem solver (what is not revealed, need not be exploited).
7. For many practical problems, experience alone reveals enough structure to lead to efficient problem-solving.
8. The underlying mathematical structure in these domains is independent of the labels given to the conditions and operators in the state-space definition.
9. This mathematical structure is embodied in the interactions of the conditions, defined by the operators of the domain, the relationship of this interaction complex to a given state, and the relationship of the given state to the goal state.

[1]Dept. of Computer and Information Sciences, University of California, Santa Cruz, CA, 95060 USA
levinson@cse.ucsc.edu [2]

10. The effects of such interaction complexes is entirely domain-independent and is governed by regular laws just as such interactions in matter and energy are governed by the laws (not necessarily all known) of physics.

11. These laws once discovered and exploited by computers will make them intelligent under the definition given above.

12. Computers may be an important tool in the process of discovering these laws.

With these tenets in mind and the ultimate goal of discovering the laws of state-space search before us we are working on these separate but absolutely complementary projects:

- GENERIC-REPRESENTATION: A graph-theoretic representation of the definition of a single-agent or multi-agent state-space search problem that is independent of all domain-specific or arbitrary labels. (*i.e.* a program that can convert the declarative definition of a state space into a more generic, but equivalent, graph-theoretic representation).

- INFORMED-LEARNER: A program that performs well in state-space search given this graph-theoretic definition of the state space, supplied domain-independent heuristics and experience.

- BLIND-LEARNER: A program that performs well on state-space search problems given no definition of the state space or pre-supplied heuristics - just the rewards at the end of the game, experience and legal states (as raw bit vectors) to choose from at each choice point.

- MORPHII: A programming environment that allows experiments to be easily designed and carried out on the above topics.

None of these projects is designed to directly produce the laws of state space search but to gradually give us an understanding that could lead to these laws and to their empirical validation. Meanwhile work on these problems increases the power of the machine. The basis for the work on each of the four subprojects is described below. The summation of this work is fully domain-independent adaptive game-playing software known as MORPHII. MorphII is a successor of the Morph chess system [24, 19] that has achieved approximate novice strength despite using just 1-ply of search and few human-supplied heuristics. This paper gives a fully domain-independent version of Morph as opposed to the original which despite good intentions carries some human biases and chess idiosyncrasies. Further, we believe the new learning mechanism described in the sections on informed-learning and blind-learning directly addresses limitations of the original Morph model: graphs being too specific and insufficient freedom given the system with respect to the class of patterns that can be formed and to the combination of their weights.

4.1 Generic Representation of Games and Search Problems

We claim that a large class of state-space games can be represented by utilizing the notion of directed hypergraphs. A hypergraph is a set S coupled with a set of hyperedges that are subsets of S. A hyperedge is directed if its nodes are ordered. A hypergraph is nested if its nodes themselves may be hypergraphs.

We now define the following game that we call the "generic-hypergraph-game". The specifications have alternatives that may be selected to form other games.

- Each player starts with 0 points.

- Let P1,...,Pn be a finite set of primitive boolean conditions.

- States are n-tuples [P1,...,Pn] representing the truth-values of these conditions.

- The game starts in some specified initial state or the initial state is selected randomly. Precisely, the initial state is chosen randomly from a set of states satisfying a given set of conditions.

- Operators are ordered pairs of a set of primitive boolean conditions as pre-conditions and return another set (not necessarily disjoint from the preconditions) of post-conditions. An operator is legal at a given state if its preconditions are satisfied. Alternatively, operators could have probabilistic effects. At each state one agent is asked to select an operator from among the legal ones.
- Another set of operators are executed automatically after each state is created - these “reward operators” are usually used for standard bookkeeping operations such as assigning the proper number of points to each player.
- Terminal conditions are hyperedges, such that if each of their conditions are achieved the game is over. Games will also terminate if no legal moves are available for the player to move or (possibly in addition) if a position repeats with the same player to move.
- Players alternate turns selecting applicable operators, the player with the most points at the end of the game wins.
- Knowledge limitations: Normally each player has perfect knowledge of the operators and of the current state. Variants include restricting each player’s knowledge to a certain set of conditions in the current state, or to certain definitions of the operators.

4.1.1 Generic Game Taxonomy of Popular Games

Here we outline how popular games can be viewed as variants of the generic hypergraph game of the previous section.

- Single-agent search problems such as the tile puzzles fit naturally in this framework. If the object is to use as few moves as possible, a primitive condition is set that is unaltered by the rules and decrements the player’s score by 1 (through the reward operators) each time it occurs.
- Tic-tac-toe-like games (such as Qubic, Renju and GoMoku) in which objects are placed and never moved, have operators with one condition in the pre-list, one condition in the post list and all terminal hyperedges also produce positive rewards.
- Hex is a game played on a grid of hexagons, that players alternate taking possession of, one player tries to make a path of his own hexagons from top to bottom and the other player from left to right. Hex falls perfectly into the Tic-Tac-Toe format by considering the hexagons as nodes and the various paths as the hyperedges, with rewards assigned correspondingly to which ever player can win in that path. But Hex can be viewed as a “more complex” tic-tac-toe like game in that there are an exponential number of hyperedges per nodes in the graph. Hypergraphs which exhibit this property (including those for checkers and GO) have been shown to be PSPACE-hard (for arbitrarily large boards) and are PSPACE-complete [16] if restricted to polynomial length games.

  > ...the fact that a problem is PSPACE-complete is even stronger indication that it is intractable than if it were NP-Complete; we could have P=NP even if P is not equal to P-Space.

- Chess-like games such as those defined in MetaGame[31] have primitive operators that are more complicated than in Hex. Further, conditions may change non-monotonically (once true may become false). Note that in a natural “bit” representation for these games there may be no explicit pieces or squares but boolean conditions representing piece-square pairs.
- Go-like games (such as Othello), in which there is an element of point accumulation, have richer reward structures than Tic-Tac-Toe and thus although related to Hex have an even greater combinatorial explosion of goal states over primitive nodes since they involve goals that are in essence subsets of hyperedges (nested hyperedges) themselves.

- Bridge-like games have random initial states. Bridge can be represented as 52 x 6 conditions for which player owns which card, whether it has been played, and whether it has been played in the current trick. The communication aspect of bridge can be defined by simply making explicit the reward structure that encourages collaboration and giving each player access to the information on other player's selected operators (corresponding to bids).[3]
- Chance games such as Monopoly and Backgammon, require stochastic conditions to be set on each move that then limit the legal moves available to the next player. The notion of money in Monopoly may be defined directly as the number of points a player has. Note that unlike previously mentioned games, having enough points may be a necessary precondition for certain operators.

Clearly, this representation is not fully adequate in that certain games (such as the stock market?) fall out of this framework, some games are expressed unnaturally and it is combinatorially intractable. Simply enumerating all of the terminal sets of conditions in chess or GO is a daunting task for example. Still, by illuminating the mathematical structure of these games, the verbiage accompanying individual domains that make us deal with them singularly rather than holistically has been removed. The "verbiage" is a very succinct way of abstracting and referring to the underlying structure. Barney Pell's Metagame [31] is an excellent example of generating generic chess-like games at the normal abstract operator level. He also has general heuristics that allow decent MetaGames to be played without domain knowledge other than the rules. These heuristics are however confined to these chess-like games and don't generally deal with the mathematics of state-space search itself. Though it would seem that generalizing them to this larger class of games may be possible.

Hoyle [13] is another domain-independent game-playing and learning system that deals at the abstract operator level. It carries with it a rich set of advisors that embody human supplied heuristics. To the degree that these heuristics are truly domain-independent and operate on a generic as opposed to specific game-representation we can say that Hoyle is an informed learner, but certainly not a blind learner.

In the next section we show how a more natural graph-theoretic generic game structure may be developed by taking advantage of the concept of variables and abstract operators.

4.2 Generic Games with Abstract Operators

To reach a more natural definition of generic games we need to add the notions of domain objects, static relations, dynamic relations, variables and bindings but still keep the "label-free" framework by omitting the arbitrary names assigned to objects, conditions and operators.[4] The framework, incorporating state space search, is inspired by Peirce's existential graphs [35] and the more modern version "conceptual graphs" developed by John Sowa [37] and our own work in experience-based planning [23].

The state space search paradigm in which problems are broken down into initial conditions, terminal conditions (goals) and operators continues to be one of the most pervasive problem-solving models used in AI research[5] The fundamental importance of the paradigm is seen when one recognizes that each of automatic programming (in which the machine translates from one very high-level language into one less "high"), automatic theorem proving (in which one tries to prove formulas from axioms and inference rules), and pattern recognition (in which one parses from one representation into another) are special cases. We now show how all state space search problems as they are normally formalized [21] can be viewed as relation-based transformations over hypergraphs (which are a special case of conceptual graphs).

---

[3]The simplicity of the specification, shouldn't obscure the fact that there are rich and complex research problems in getting computers to bid effectively or to learn to do so.

[4]Once this is done conceptually the names may be retained as mnemonic aids for humans, but recognized as arbitrary by the program.

[5]The objective of a state-space search problem is to find a sequence of operators (transformations) that will convert a state that satisfies the initial conditions into one that satisfies the terminal conditions, under various optimality criteria and resource constraints.

The following is a description of the components of a search problem, with running examples taken from Tic-Tac-Toe[6] and Towers-of-Hanoi on three disks:[7]

- *Each domain will have a finite set of domain objects.* In Tic-Tac-Toe the objects will be squares {S1,S2,S3,S4,S5,S6,S7,S8,S9} and pieces {X,O,B (for blank)}.
  In Towers-of-Hanoi, the objects are Pegs{P1,P2,P3} and Disks {D1,D2,D3}.

- *Unary, binary and higher relations may be defined on these objects.* An n-ary relation is a set of n-tuples of domain objects. In the finite search domains, rather than defining types explicitly we shall simply note that they are implicitly defined as the set of objects that occur in any single field (attribute) of a relation. At the implementation level, relations that are symmetric or transitive may be abbreviated by specifying a kernel set of tuples and then giving the mathematical properties from which the remaining tuples can be inferred. Other abbreviations and computation of relations are possible such as finding adjacent squares on the chessboard through calculation. We break relations into two classes: *static relations* are those whose definition (tuples) remain constant for a given game, and *dynamic relations* are those whose content can change from state to state. A frequent use of static relations is in defining board topology. Most domains have a dynamic relation corresponding to ON to say which pieces are on which squares.

  In Tic-Tac-Toe, we define a "THREE-IN-A-ROW" static relation corresponding to winning sets of squares: {(S1, S2, S3), (S4, S5, S6),...} and the ON dynamic relation initialized as {(B,S1),(B,S2) etc.}. In Towers-of-Hanoi there is a static relation SMALLER-THAN initialized to: {(D1,D2),(D2,D3),(D1,D3)}. Towers-of-Hanoi also has the dynamic relation ON, initialized to: {(D3,P1),(D2,P1),(D1,P1)}.

- *Operators define transformations over states by changing the contents of dynamic relations. They are specified by giving sets of preconditions, additions and deletions.* Note that each of these sets (and their combination) can be viewed as a conceptual graph or hypergraph.

  For Tic-Tac-Toe we have an operator MOVE(piece,square): Pre: ON(B,S1), Add: ON(X,S1), Del: ON(B,S1). (we assume the board is always oriented with X to move.) For Towers-of-Hanoi we have the following MOVE operator:

```
MOVE(d1:disk,p1:peg,p2:peg) =
pre: ON(d1,p1) AND ~(p1=p2)  AND ~((ON(d2,p2) OR ON(d2,p1))
     AND SMALLER_THAN(d2,d1)).
add: ON(d1, p2)
del: ON(d1, p1)
```

  As a further example, the SWAP operator that switches a bottom block for a top block in the blocks world [29] and vice versa in a tower of three blocks would have this representation:

```
SWAP(block:x, block:y, block:z)=
pre: ON(x,y) AND ON(y,z) AND CLEAR(X)
add: ON(z,y)
     ON(y,x)
     CLEAR(z)
del: ON(x,y)
     ON(y,z)
     CLEAR(x)
```

[6] Tic-Tac-Toe is played on a 3x3 board with two players X and O alternating turns selecting cells with X going first. The first player completing a row, column or diagonal wins.

[7] Towers-of-Hanoi is a single agent game involving three disks "small", "medium", "large" placed on top of each other in order on the first peg of three. The object is to move the disks one at a time such that at no time is a disk on top of a disk smaller than it on the same peg, and such that the disks finish all on the third peg.

This representation has an equivalent operator graph involving three variable nodes and six dynamic relation edges. Such a representation of SWAP could apply to thousands of domains that involve this type of swapping, moves in tile-puzzles are an instance, except that one of the variables is instead a domain object (for the blank tile).

- *States are hypergraphs over domain objects.* As the static relations are always true it is only necessary to give the dynamic relations when representing a state.

- *An operator is then applicable in a given state iff there is a 1-1 mapping in variables of the preconditions of the operator to domain objects such that all relations specified in the operator are true (or false, if negated) of those domain objects. The result of applying the operator is to remove from the current state those tuples corresponding to the deletions and add those tuples associated with the additions to the contents of the dynamic relations.*

  Since static relations are constant throughout the problem-solving process those bindings of objects to variables that could ever possibly (constrained by the static relations) satisfy an operator definition, can in principle be computed ahead of time leaving only the dynamic conditions to be checked. This is effected in our implementation presented in section 7.

- *Terminal conditions are defined in exactly the same way as preconditions, addition and deletion conditions of operators. It is automatically assumed that a player having no legal moves is terminal.* For tic-tac-toe, we have THREE-IN-A-ROW(s1,s2,s3) AND ON(X,s1) AND ON(X,s2) AND ON(X,s3) as terminal. In Towers-of-Hanoi, we have ON(D1, P3) AND ON(D2, P3) AND ON(D3, P3) as terminal conditions.

- *Reward conditions are conditions (often the same as terminal) that are coupled with a reward to each player based on the outcome of the game.* For Towers-of-Hanoi and Tic-Tac-Toe, reward conditions are the same as terminals and assign a *win* to the player who has just moved.

- *Finally, FLIP is a static binary relation over domain objects used to define symmetries so that a game can be encoded from one player's perspective only.* For tic-tac-toe FLIP is: {(X,O),(O,X),(B,B),(S1,S1),(S2,S2)... Tower-of-Hanoi, being a single-agent game, does not require a FLIP operator. Similar transformations were introduced in metagame [31]. Figure 4.1 gives an equivalent conceptual graphs description of the above rules for Towers of Hanoi.

- *In summary: An abstract game or search problem is defined as a finite set of domain objects, and finite sets of static relations, dynamic relations, operators, terminal conditions and reward conditions. Finally, for convenience in encoding, we define a symmetry condition (known as FLIP.)*

The conclusion is that a large spectrum of single-agent and multi-agent search problems can be viewed as games of (hyper)graph-to-graph transformation directly analogous to organic chemical synthesis: states are graphs, and operators are graph-to-graph productions, terminal and reward conditions may also be expressed as graphs. This conclusion is not surprising, given that conceptual graphs and other semantic network schemes have been shown to carry the same expressive power as first-order logic. *The conclusion is significant, however, in that it suggests the potential for graph-theoretic analysis of the rules of a domain and ensuing experience for uncovering powerful heuristics and decision-making strategies.* [23] It also suggests that state-space search can be monitored in a uniform manner, we take up this topic in section 7.

4.2.1 Evaluation of representation scheme

The representation scheme does not yet include facilities for inference over static and dynamic relations. Such a facility is available in conceptual graph theory but has not been necessary in the relatively straightforward search domains we have analyzed but would be required for efficient automatic theorem-proving. All such facilities will be available shortly in the Peirce conceptual graphs workbench [15, 11] in which our learning system is implemented.

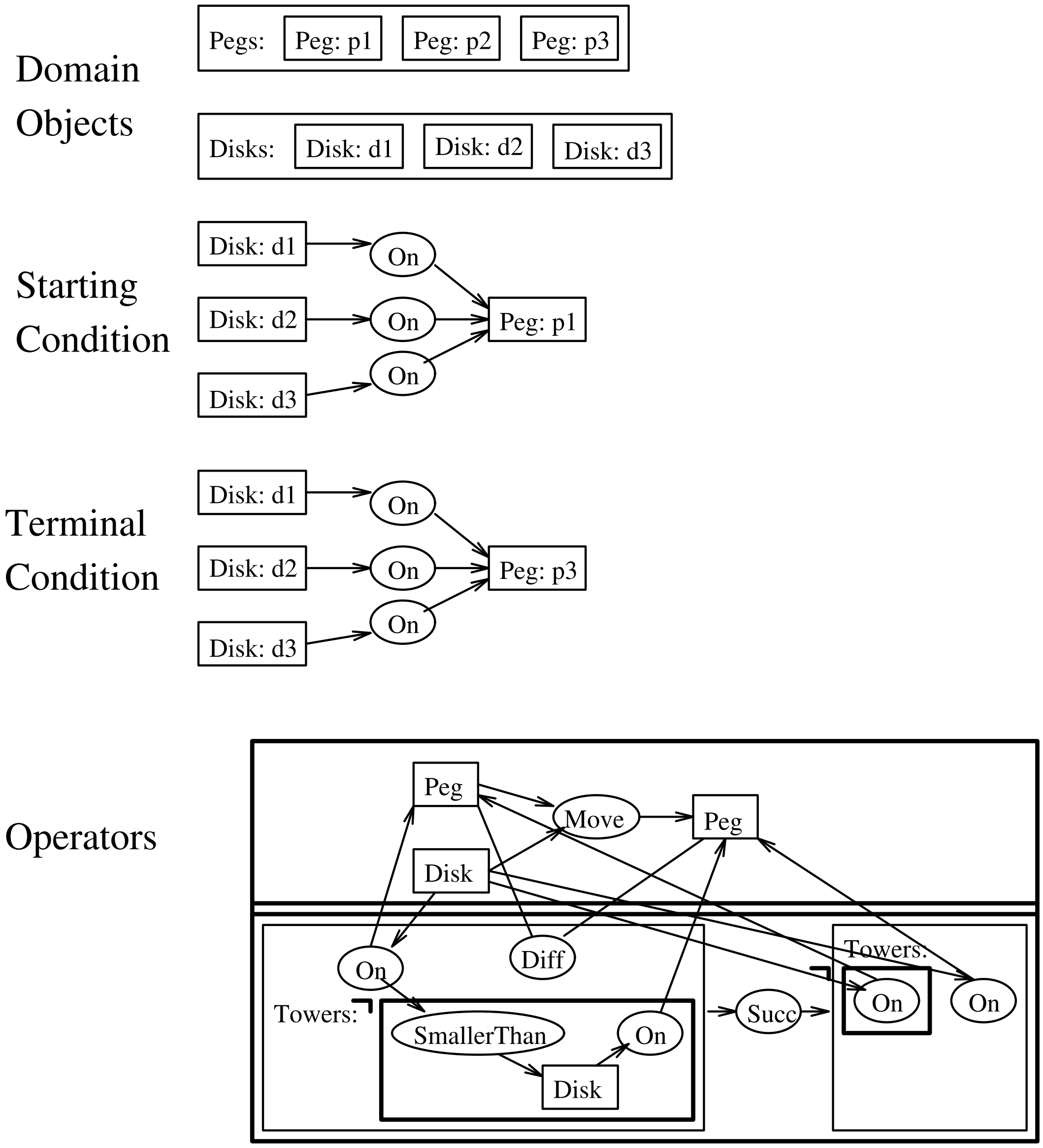

FIGURE 4.1. Conceptual graph specification of the rules of Towers of Hanoi

An important step in showing the generality of the mechanism will be showing that games generated from the MetaGame generator take this structure. The implications of managing and manipulating such a structure are yet unknown, but the potential seems high. For example, any macro-operator that is executable in one search domain will work in any other (single-agent) domain that carries that macro-structure. Using such a domain-independent graph-theoretic representation we have been able to show that the entire TWEAK planning system can be reduced to 5 such abstract operators and a simple control structure, and further that two classical AI problems: Roach's robot problem and Sussman's anomaly can be solved using the same database of domain-abstracted operators as macros [23]. It is hoped that the evaluation of operator-condition interactions can be done across domain structures.

4.3 The hypergraph game: a generalization of Tic-Tac-Toe

To further illustrate the unifying power of studying games based on their mathematical structure let's take a closer look at the tic-tac-toe-like games. Such games reduce to the following "basic"-hypergraph game: Given a hypergraph, players alternate selecting nodes and the first player who owns all nodes in any given hyperedge wins. We shall assume in the discussion below that there are only 2 agents. For example, 3x3 tic-tac-toe has a graph of nine nodes and 8 hyperedges. 3x3x3 has 27 nodes and 48 hyperedges. 4x4x4x4 (Qubic) has 64 nodes and 84 hyperedges. The discussion here builds directly on previous work on forks[13], GoMoku and Qubic[1], hopefully putting that work in proper perspective.

The hypergraph representation (being unlabeled) makes symmetries fully realizable through graph-isomorphism. This mathematical representation also lends itself to reasonable heuristics such as the quality of a node choice is proportional to the number of "live edges" it is involved in and inversely proportional to the number of nodes remaining in each of those edges. (i.e. $\sum_{i \epsilon E} \frac{1}{(n(i)-1)}$ where $n(i)$ is the number of nodes in edge i and E is the set of live edges). Through the use of subgraph analysis more precise heuristics can be developed [13].

4.3.1 Reductions in the basic hypergraph game

It is useful to become familiar with reductions that preserve game-theoretic value (without incorporating search) in basic hypergraph games, since such reductions may very well not have been apparent from the traditional state representation. After each move, a hypergraph may be translated to a smaller but equivalent representation (for calculating the value of a state assuming optimal play). The idea is that we only need to remember for each edge which player owns it and which nodes remain unplayed. Given this information, edges with nodes owned by both players may be removed. A set of duplicate edges (edges that involve exactly the same nodes) can be reduced to a single-edge that preserves ownership if all owners are the same, or carries no ownership if both agents own such an edge. Similarly, if ownership is the same, edges that are subsumed by other edges are removed. So with each move:

1. Remove each edge that the move was to that was owned by another player.
2. Remove the node itself and set the ownership of all other affected edges to the player who has moved (the ownership fields may already be so set).
3. If any edge now has zero nodes, the player who moved wins. If no more edges remain the game is a draw.
4. Remove all but one out of a set of duplicate edges and update ownership as described above. Remove any edge that is a superset of another and carries the same ownership.

Now that we have studied the basic-hypergraph games, it is desirable to analyze to chess-like games. These games have the following additional features:

1. Operators are more complex as they have multiple pre-conditions and post-conditions.
2. Conditions can change non-monotonically: a condition which is true can become false and vice versa.

Chess-like games [31] are simply examples of perfect information, two-agent, search problems.

4.4 Improving on the Morph learning model

Once we understand that search-problems are games of graph-to-graph transformations, we can also understand how knowledge of the existence of a subgraph of a graph representing the current state may carry predictive value of the outcome of the game. Inherent in such a graph are the potentialities of operators that can be applied (or prevented) and a relationship to terminal conditions and goals of the game. Given our current understanding of the physics of state-space search, it is not yet known how to assess the value (as expected outcome) of such a subgraph directly but instead it may be learned statistically from experience as in the Morph chess system. Recall that in that system we limit search control to exactly 1-ply of lookahead to force us to focus on issues associated with heuristic construction and development. In fact, the restriction to 1-ply search is one of degree rather than kind since determining the presence of Morph's patterns requires 1 or 2 ply of search itself. Through the knowledge of static relations in a domain and experience we wish to pre-compile many search results into the heuristic.

Given that the values of subgraphs can be approximated accurately, as we believe is the case in Morph (though further improvements to the weight learning-mechanisms are certainly possible), we must ask ourselves then "why is Morph not a better chess player!?". We cite the following two major reasons for lack of more success in the underlying playing system:

1. *Inappropriate mechanisms for combining the values recommended by each of the individual subgraphs.* The original hypothesis was that the pattern-values could be combined numerically, independent of which patterns have produced these values and the relationship between these patterns. We now believe that this hypothesis is false, if we are to produce a strong heuristic. The Manhattan Distance for tile puzzles suffers similar shortcomings. In fact, experiments we have conducted in which one attempts to combine the values of higher-level patterns (involving several tiles in the 8-puzzle), without knowledge of the underlying patterns but *having their correct values* (as expected distance to the goal for states having the pattern) have also proved unpromising. These experiments were done by hand rather than using statistical methods to form the combination equation as described below, but still demonstrate the difficulty of using even perfect knowledge about a subset of patterns in a complex domain.

2. *A second weakness is in the specificness of the graphs in the chess system.* The system lacked the ability for information learned about one graph to directly influence the values of other similar graphs. This leads to many learning inefficiencies, especially considering that for some specific graphs they may be only seen a few times in the system's career and that this may provide insufficient information to give them accurate values.

Fortunately, these difficulties can be addressed. The original hypothesis underlying the Morph design is that *knowledge of the relationships between objects must be exploited.* This hypothesis has been borne out by the moderate success Morph has had using its graphs. The hypothesis must simply be carried further: the two difficulties above both refer to lack of exploitation of the relationships of the graphs themselves.

To extend our design we developed a generic learning module with the the following properties:

- It contains n subsystems.
- It estimates the value of a state S, by first consulting its subsystems (which are themselves generic learning modules) to get their estimates of the value of S and combines their values numerically (perhaps non-linearly) using an equation that it has learned. The combining rule exploits the past accuracy of the subsystems.
- The module receives feedback (as a "better prediction value") for its predictions from a higher-level system.
- The module learns by modifying its combination equation.

For the purposes of this discussion, we may ignore the learning method used by the generic learning module. Conceptually, any function learning method based on neural nets, regression analysis, nearest neighbor methods, etc. will do. *We will focus on the identification of the subsystems themselves and towards their interaction.*

The individual learning modules are to form a partially-ordered hierarchy based on the "subsystem" relation. It is assumed that all modules of the hierarchy receive feedback for individual states in a state sequence using the same temporal difference learning [?] mechanism as in Morph. At the bottom of the hierarchy are "primitive systems" these are systems with no subsystems that attempt to learn a constant that is the best predictor of their feedback values. Every module is responsible for predicting the outcome of the game, higher-level systems base their predictions by combining the recommendations of lower-level systems. To insure "action" the primitive systems in the hierarchy are initialized using a random, Gaussian distribution around the expected reinforcement value for the game. An informed learner may already produce *through analysis* the values of certain subsystems. Individual subsystems correspond directly with the patterns in the Morph chess system, but are more flexible as they learn how to combine values produced by subsystems lower in the hierarchy rather than attempting to determine a fixed constant weight.

As games move from the simple to complex the complexity of the individual learning module and the hierarchy of learning modules will need to increase as well. For simple games, as in 3x3 Tic-Tac-Toe it is possible to write an evaluation function for optimal play that is simply a linear combination of the values of objects (X,O, or B) on individual squares - corresponding to a 2-level hierarchy, with nine modules on the lower level and one module on the top. However, for a game such as chess, an evaluation function based directly on the values of objects on the 64 squares will not perform well (as experiments we have conducted on neural networks have shown) as it is known that no linear combination of these objects will do (since it is the relationship between objects and not individual objects that is critical). To simply expect a non-linear evaluation function to succeed begs the question, since where do the complex interactions it needs to consider come from! Also, in order to *efficiently* learn the proper coefficients for this complex function, knowledge must be transferred and shared between terms (otherwise all pawn-can-take-queen combinations for example must be learned separately).

We hypothesize that the structure of the rules themselves dictates the structure of the system hierarchy, that by exploiting this structure the learning process can be made much more efficient and that the complexity of the learned evaluation function can be greatly reduced. For example,(consistent with the reduction rules given above) it is sufficient to evaluate a state accurately in tic-tac-toe to know the number of X's and O's in each row, column and diagonal. The counting procedure for each row, column and diagonal can use the same "Three-in-a-row" module and due to symmetry considerations the system need only consider different coefficients for three different types of "three-in-a-row relation". Consider, the generalization of tic-tac-toe, given as the basic hypergraph game above. For an arbitrarily large board, it would be difficult for a single learning module to learn to evaluate states well, without knowing the underlying relationships on the board. However, we hypothesize that any state in such a game can be evaluated accurately (after reductions as above) given the following:

- Primitive subsystems for every node in the domain.
- A subsystem for each hyperedge, that stands above the hierarchy for each individual node.
- A subsystem for each node that combines the values returned by each of its hyperedges.
- A top-level system that combines the values of each of its nodes.

In general, a hierarchical reinforcement learner can be constructed from the rules of a domain as follows:

- The primitive systems correspond to the primitive conditions in the underlying problem domain. These are usually caused by 1-1 relationships between sets as with the dynamic relation ON between pieces and squares in many games. The tile-puzzles have 81 primitive conditions, chess has 13x64 primitive conditions for pieces and squares (and a few more dealing with the potential for en passant capturing, draw by repetition and castling). These primitive conditions form the lowest level in the hierarchy.
- The second level in the hierarchy correspond to state-variables (squares for many domains) which are sets of primitive conditions such that at most one is on at a given time. Naturally, for the state-variables the best predictor will be the value of its primitive condition that is ON.

- The next level(s) in the hierarchy correspond to the static relations and type declarations in the domain. Thus in the tile-puzzles there is a system for every adjacent tile pair, in tic-tac-toe for squares forming a 3-in-a-row relation, in chess for all sorts of relationships between squares that are relevant in the rules (In-same-rank, In-same-file, In-same-diagonal, In-L-Shape, etc.).
- The next level deals for each second-level subsystem (square), with combining the values recommended by all static relationships that it is in.
- The final top-level combines the values for all subsystems immediately below it.

This design deals directly with the first weakness in Morph discussed above since the combination functions learned will be localized and specific to the patterns under consideration (*i.e.* in the new framework two patterns that return exactly the same value may be treated differently by the problem-solving system based on their learned interactions with other patterns).

The second weakness (inability for patterns to share information) has been diminished somewhat by the hierarchy but still remains. Consider chess for a moment: as stated above the system would have to learn new values for every "in-same-diagonal" relation. To handle such redundant effort, each relation type will correspond to exactly one subsystem, while individual instances of the relation will supply input to this subsystem, but take the output as specific input to a higher-level system. Thus only one subsystem need be created for each relation type. Similar redundancy may also be exploitable in higher order subsystems. For example, in chess each square is involved in a similar set of relations with other squares though the amount and types of these relations may vary.

As another example, consider the tile puzzles. The lowest level corresponds to the value of each position-tile pair, level-two to the value of each position, level 3 corresponds to the "adjacency" relation between two squares used to define legal operators. Level 4 combines the value of such conditions at each node. Due to symmetry considerations only 3 types of level 4 modules are necessary. Finally, Level 5 combines the nine values recommended by each position (though the learning may also take advantage of the symmetries exploited on level 4).

4.5 Blind-Learning

Obviously, the problems of blind-learning are much more substantial than in the case of informed-learning, precisely because there is no domain definition to guide the construction of the learning hierarchy. Not only do the generic learning modules have to learn as before, the question of which generic modules are formed and their interrelationship becomes a critical issue. Further, recognizing redundancy so it can be exploited becomes a difficult matter.

Not only can an informed learner start from a much better place, experience should also be exploited more accurately in Informed Learning than in Blind Learning. Still it is important to study blind learning, because in some domains one is not given access to any declarative description of the rules, because the additional difficulties in blind-learning give us a better appreciation for the information supplied in the rules, and because it forces the learning system (and its developer) to make use of all available information. In the blind learning framework the agent is given privy to the following information and no more:

- The current state of the board as a finite-length vector of boolean conditions (bits). It is intended that this bit description be *exactly* the information a legal move generator would need to generate the correct set of legal moves in a given state. Beyond this no interpretation is given to the bits and they may be placed in a "scrambled order". Just as Informed Learning may operate in domains with rule encodings ranging from malicious to benevolent, Blind Learning must operate with state encodings of varying quality.
- That the other agent is playing with the same set of rules it is, i.e the legal moves and rewards from any set of conditions is fixed and the same for each player (under some inversion or symmetry of the board).
- The states resulting from each of its legal moves at any given point in the game.

- The reinforcement for the game in terms of some reward that the agent is trying to maximize.

Consider how Tic-Tac-Toe would appear to the blind learning agent: The board can be represented as 18 bits - 2 bits for each square: 00-empty, 01-X, 10-O, 11-is an unused code. To make things more obscure one can consider the board as representing a base 3 number (even with the squares placed in a strange order) and thus states could be represented as a 14 bit binary number. Thus, even a straightforward game like Tic-Tac-Toe could quickly disguise any resemblance to the standard board and the form of the rules of play. This is intended. We feel that methods must be developed to produce a general learner - one capable of adapting, given proper amounts and variety of experience to any game-playing environment. *It is our hope that the graphs denoting the subsystems will help to exploit any isomorphic interactions of values wherever they might occur. It is exactly this exploitation of "analogous relationships" through graph-isomorphism that separates MorphII from virtually all other learning systems.*

A blind-learner would start as a two-level system with primitive subsystems for individual bits and a top level system that learns to combine the values for each bit. Intermediate subsystems may evolve by creating a subsystem for every set of bits that disappear or appear with a single operator application. These add-sets and delete sets correspond to complex relations, intersections of these may correspond to simpler relations. Further, state-variables (corresponding to level 2 in the hierarchy discussed above) can be recognized by starting with all bits in a single variable and gradually splitting into smaller variables by finding counter-examples [26]. Such variables correspond very well to unary relations or types in the hidden game definition. For example, these could be pieces or squares in a non-malicious encoding of chess.

4.6 Exploiting analogous relationships

In the blind-learning framework subsystems will be formed based on relation instances rather than relation types as in informed-learning. Thus, it is critical to try to identify similar relations (by comparing coefficients in their learned combination equations). Such identification requires a type of fuzzy-graph matching.

To further understand the motivation for fuzzy-graph matching consider the blind learning framework and the combinatorial explosion associated with chess. Consider, for instance, the difficulty of learning that placing one's queen where a pawn can capture it is usually bad regardless of where it occurs on the board – including in those places it has not yet occurred! Learning such a straightforward concept is actually a difficult one for all learning algorithms - especially those operating on uninterpreted structures. Here we are assuming no knowledge of piece, square, pawn or queen etc. has been supplied.

We have used the term "variable" to denote a set of bits such that at most one is on at a given time and have described how such variables can be learned. We shall say that the value of a variable at a given time is equal to the value (if any) of its positive bit. Thus the generic learning module can be applied to learning the value of a conjunction of two variables as a combination of their underlying values.

We shall assume that the blind learning system has observed several instances of the pawn-can-take-queen relationship (i.e relationships between two uninterpreted piece-square conditions due to this relationship) and encounters a new instance. If it has seen this instance as a capture before it hopefully will be fine, since it would have created a specific relation for the deleted and added conditions. But if not seen before there may still be a learned interaction between the pawn-square condition and a variable formed by the conditions referring to the square the queen is on.

That is, the desired concept can be learned as a result of learning the square "variable" where the queen is on and having formed a conjunction of the pawn-square condition (or a variable involving it) with this variable. The variable gets instantiated with the value of the queen-on-square condition. Presumably there have been other captures by the pawn to the square the queen is on. Thus the learned interaction between these squares will hopefully look similar (to some degree) to learned interactions between other pairs that denote "attack". Thus, sharing closeness with these other interactions (where in fact queens may have been captured) a possibly accurate transfer of knowledge may be made.

The assumption is that games of interest have regular underlying structure - to the degree that this assumption of regularity is false the more difficult is the learning task and the more false inferences that will need to be made before the proper structure can be learned – if at all.

If no connection between these squares has been noticed before, being ignorant of board topology and the rules, it would be virtually impossible to recognize the interaction – though possibly it would be inferable very indirectly from other patterns.

4.7 MorphII: Domain-Independent Games Environment in C++

The blind learning and informed learning testbeds described above are available as part of the public domain software known as the Peirce Conceptual Graphs Workbench[11]. The learning system in Peirce, known as MorphII, accepts the rules of a single agent and multi-agent state-space search domain, translates them into conceptual graphs, and then monitors games and learning via a "super-referee". As the code is fully objectified in C++, it is possible for learning modules to be coded and tested as well as for modifications to be made to our initial implementations of blind learning and informed learning algorithms as described above. Declarative rule sets for a number of games are available, including backgammon, chess, tic-tac-toe, and the 8-puzzle. It is also possible to test the power of different search algorithms and human supplied heuristics in these domains. Our implementation methodology is UDS as described in conference proceedings.

4.7.1 Performance Results

With this scheme, we have been able to monitor a variety of domains including Tic-Tac-Toe, Tower-of-Hanoi, 8-puzzle, and Hexpawn[8] at a level of efficiency that is faster (in some cases up to 10 times) than previous programs of ours that had been written specifically for these domains but were not incremental.

We give timings for UDS playing at various search depths and monitoring a variety of search domains that have been compiled into its network from rule definitions as in Section 6. The results reported are using a C++ implementation on a SUN SparcII with 64 megabytes of main memory. As UDS is domain-independent and not hard-coded for any of these domains it clearly should not be as fast as any domain-specific system. Still, through incrementality, its results, as shown, are reasonable and would likely beat (for domains with high branching factors) a hard-coded program that does not exploit incrementality.

Table 4.1 presents results from our earlier implementation in which relation tables are stored as a sparse matrix. In particular, a relation is stored as a set of tuples where each tuple is identified by giving its arguments. In the table, "Random vs Random" means that a random-agent competes against another random agent and "n Ply" means one agent is searching this deep while the other agent is playing randomly. The 500 moves Tower-of-Hanoi results are with game restarts and the 500 move 8-puzzle results are for 1 game.

Table 4.2 presents results from our most recent implementation in which the multi-dimensional bit array method is used for storing relation tables. Each tuple is an individual bit cell in the matrix. A 1 means the tuple is present, a 0 means the tuple is absent. The subscripts of the cell correspond to the arguments of the tuple. The main advantages of the bit matrix storage scheme is that processing is uniform for all tuples and that row operations on the matrices can exploit the bit-parallel concurency provided by the logic operations of the workstation. In comparison with Table 4.1 we see speedups of a factor of 5 to 81, typically around 25. Towers-of-Hanoi shows the greatest speedup, and Hexpawn the least. With the exception of Hexpawn, speedups are significantly greater at higher ply than lower ply.

[8] Hexpawn is a simple version of chess played on a 3x3 board. Each player has 3 pawns. In the starting configuration each player's pawns are along their respective edge of the board, with the middle rank unoccupied. White moves first and players alternate moves. Pawns move one square forward and capture diagonally as in normal chess. The first player to get get his/her pieces to the opposite side of the board or to capture all the opponent's pieces wins. The game is a draw given optimal play. Hexpawn 6 is Hexpawn but on a 6x6 board where each player has 6 pawns.

	100 Moves		500 Moves	
	total	average	total	average
Single–Agent Games				
Towers-of-Hanoi:				
Random vs Random	< 1.0	–	2.0	0.004
1 Ply	3.0	0.03	15.0	0.03
2 Ply	19.0	0.19	102.0	0.20
3 Ply	110.0	1.10	618.0	1.24
8 Puzzle:				
Random vs Random	< 1.0	–	2.0	0.004
1 Ply	4.0	0.04	22.0	0.04
2 Ply	42.0	0.42	205.0	0.41
3 Ply	298.0	2.98	1605.0	3.21
Double–Agent Games				
Hex Pawn:				
Random vs Random	2.0	0.02	10.0	0.02
1 Ply	6.0	0.06	30.0	0.06
2 Ply	22.0	0.22	110.0	0.22
3 Ply	80.0	0.80	405.0	0.81
Tic-Tac-Toe:				
Random vs Random	3.0	0.03	15.0	0.03
1 Ply	9.0	0.09	45.0	0.09
2 Ply	55.0	0.55	290.0	0.58
3 Ply	415.0	4.15	2095.0	4.19

TABLE 4.1. Tuple–List Execution Speeds, in seconds

Tables 4.3 and 4.4 summarize the quality of blind learner's playing. Tower-of-Hanoi starts from randomly selected legal initial states; for both Tic-Tac-Toe and Hexpawn, draws are counted as wins for the second player. The statistics presented are for a very basic blind learner (compared to that previously outlined) that simply learns the values of individual bits through gradient-descent weight updating in the uninterpreted state representation, or randomly selected sets of bits in the underlying representation. For Towers-of-Hanoi, 9 bits are used for the blind learner to represent which disk is on which peg. Both Hexpawn and Tic-Tac-Toe 18 bits are used to represent the state (00 for blank, 01 for O or white pawn, 10 for X or black pawn for each of the 9 positions; 11 is not used). No effort was made to identify and exploit generalized relationships such as three-in-a-row for Tic-Tac-Toe, and only those patterns necessary for monitoring the games were stored.

Tables 4.5 and 4.6 present results for the Generalized Learing Module. Table 4.5 shows how performance grows with training. The percent wins are cumulative over all games played, and a draw is counted as a win. Table 4.6 gives performance over 250 games *after* learning; the wins and losses from prior games building the knowledge base are excluded.

4.8 Conclusion

Eric Baum recently pointed out [4] the inherent potential in exploiting the information (mathematical structure) inherent in a declaration of the rules of a given domain:

> The computer science approach has since Shannon basically regarded a game as defined by its game tree. But what makes a game interesting is that it has a low complexity, algorithmically efficient definition apart from the game tree.... Any procedure which only accesses the underlying simplicity of a game in the form of an evaluation function is inherently doing the wrong thing.. .. the main open question is how to go beyond the evaluation function picture of games.

	100 Moves		500 Moves	
	total	average	total	average
Single–Agent Games				
Towers-of-Hanoi:				
Random vs Random	0.02	0.0002	0.17	0.0003
1 Ply	0.11	0.0011	0.55	0.0011
2 Ply	0.46	0.0046	2.15	0.0043
3 Ply	1.55	0.0155	7.67	0.0153
8 Puzzle:				
Random vs Random	0.09	0.0009	0.40	0.0008
1 Ply	0.35	0.0035	1.46	0.0029
2 Ply	1.34	0.0134	6.83	0.0137
3 Ply	5.41	0.0541	27.58	0.0552
Double–Agent Games				
Hex Pawn:				
Random vs Random	0.21	0.0021	1.25	0.0025
1 Ply	0.53	0.0053	3.35	0.0067
2 Ply	2.03	0.203	10.59	0.0212
3 Ply	7.22	0.0722	36.76	0.0735
Tic-Tac-Toe:				
Random vs Random	0.10	0.001	0.55	0.0011
1 Ply	0.51	0.0051	2.65	0.0053
2 Ply	2.36	0.0236	12.80	0.0256
3 Ply	13.91	0.1391	68.37	0.1367

TABLE 4.2. Bit–Matrix Execution Speeds, in seconds

	Lowest	Highest	Average
Towers-of-Hanoi:			
Number of moves	7	21	11
Games won in $\leq$ 10 moves			77%

TABLE 4.3. Blind Learner, Single–Agent Games (100 Games)

We agree strongly with this insight and are pursuing it as deeply as we can. In this paper we show how individual search and game domains may be viewed as instances of a general graph-theoretic game of abstract mathematical relations. We argue that the advantages of taking this perspective are many:

1. An efficient domain-independent monitor can be developed that exploits the matrix and logic operations of the underlying hardware.

2. A study can be made of the relationship of the structure of the rules (problem definition) of a given domain and the structure of a hierarchical reinforcement learner or neural net for experientially learning the value of states in that domain. In particular, we outline an algorithm for converting from the relation-based representation of a domain to a reinforcement learning network for that domain.

3. The problem definition, monitor and learner for a given domain can be seen as arising together from the inherent structure of that domain, rather than being viewed as three separate processes.

4. As a result of this coherence, domains that are similar in structure give rise to similar learners and monitors. In fact UDS illustrates how the same network data structure that monitors the domain can be the basis of a reasonable learner for that domain.

5. The relationships of parts-and-wholes in the domain definition may be extracted to identify the placement of "Generic Learning Modules" in the reinforcement hierarchy. Thus learning research

Agent		Wins	
1	2	Agent 1	Agent 2
Hex Pawn			
Random	Random	59%	41%
Random	Blind Learner	21%	79%
Blind Learner	Random	67%	33%
Tic-Tac-Toe			
Random	Random	74%	26%
Random	Blind Learner	44%	56%
Blind Learner	Random	76%	24%

TABLE 4.4. Blind Learner, Double–Agent Games (500 Games)

Agent		Agent 1 Wins		
1	2	10 Games	100 Games	500 Games
Hex Pawn				
Random	Random	40%	82%	92%
1-Ply APS	Random	100%	88%	90%
2-Ply APS	Random	100%	88%	96%
Hex Pawn 6				
Random	Random	60%	51%	56%
1-Ply APS	Random	80%	72%	86%
2-Ply APS	Random	70%	86%	91%
Tic-Tac-Toe				
Random	Random	80%	72%	70%
1-Ply APS	Random	90%	92%	92%
2-Ply APS	Random	70%	96%	99%

TABLE 4.5. GLM, Double–Agent Games

can be pursued from the higher perspective of the interaction of learning modules rather than as a choice between methods.

The performance of our matrix-based monitor verifies the practicality of the general graph–theoretic view of search domains. The results of the learning system presented in this paper, are preliminary but highly encouraging, as they demonstrate that reasonably strong reinforcement learners of differing structure can be constructed from a problem description. Further research will study how the reinforcement learner can enhance itself by adding new relational tables to the network. This self-organization coupled with parameterless learning rules will further support our theme of giving the computer maximal responsibility for its learning structure.

We will be comparing MorphII to more traditional function learning methods such as linear regression, perceptrons, nearest neighbor, and neural nets. We believe that MorphII should be able to outperform these methods in that there should be no effective limit to its application of representational transformations on the data and its network structure. The nature and implementation of these representational transformations requires further study.

Acknowledgments

Barney Pell provided useful criticism and encouragement during the writing of the paper. John Amenta developed much of the MorphII software and obtained performance results. Yuxia Zhang assisted in developing the declarative definitions of games and implemented the Generic Learning Module. James D. Roberts helped with the figures and the final draft of this paper.

Agent		**Wins**	
1	2	Agent 1	Agent 2
Hex Pawn			
1-Ply APS	Random	88%	12%
2-Ply APS	Random	100%	0%
Hex Pawn 6			
1-Ply APS	Random	94%	6%
2-Ply APS	Random	90%	9%
Tic-Tac-Toe			
1-Ply APS	Random	80%	20%
2-Ply APS	Random	100%	0%

TABLE 4.6. GLM, Double–Agent Games, After Learning

5

LINEAR — Linear Notation Interface

John Esch[1] Maurice Pagnucco[2] Michel Wermelinger[3] and Heather Pfeiffer[4]

5.1 Introduction

The purpose of the LINEAR subgroup is to coordinate the development of a standard linear form grammar for Conceptual Graphs and to develop related tools. The main aim of this paper is to present a linear form grammar for conceptual graphs which we hope will form the basis for future standardization efforts. We also suggest some possible extensions to this grammar and contrast it with the grammar used by PEIRCE.

In the following section we present a linear form grammar based on the work of Esch [14] while section 5.3 discusses some suggested extensions to the linear form grammar. Section 5.4 presents the linear form grammar adopted by the current version of PEIRCE (version 0.1) and discusses some of the more significant differences between this grammar and that of section 5.2. In appendix 5.7 we include Sowa's comments on possible future directions with a discussion on integrating the CG linear form grammar with that of KIF[18], the Knowledge Interchange Format developed by the Interlingua Committee of the DARPA-sponsored Knowledge Sharing Effort.

5.2 Linear Form Grammar — Current Status

This section contains extended BNF productions defining a grammar for Conceptual Graph Linear Forms and some Extended Linear Forms.[5] It is based on [14]. The underlying character set is ASCII[6] and the case of letters is only significant within strings (production 260). However, actual implementations may provide for user-definable options, like typelabels in uppercase and variables in lowercase. The user should also be able to choose whether the Extended or the Contracted Form (see section 5.3) is to be used on output. Furthermore, in production 356 parsers may let the arc before an *r-node* be optional, as well as the last comma before the period terminating the graph.

[1]Paramax Systems Inc. (A Unisys Company), P.O. Box 64525, MS U1N28, St. Paul, MN 55164, USA. email: `esch@email.sp.paramax.com`

[2]Knowledge Systems Group, Basser Department of Computer Science, Madsen Building, F09, University of Sydney, Sydney, NSW 2006, AUSTRALIA. email: `morri@cs.su.oz.au`

[3]Departamento de Informática, Universidade Nova de Lisboa, 2825 Monte da Caparica, PORTUGAL. email: `mw@fct.unl.pt`

[4]New Mexico State University Department of Computer Science Box 30001/CS Las Cruces, NM 88005, USA. email: `hdp@cs.nmsu.edu`

[5]A preliminary YACC version of this grammar is available through anonymous ftp from `ftp.cs.su.oz.au` as file `morri/linear94.tar.Z` (includes this report) or from `ftp.fct.unl.pt` as file `pub/di/papers/Wermelinger-M/1994/cg_linear.y` (just the YACC grammar).

[6]However, for a true mapping between Natural Language and CGs the international ISO character set should be used.

5.2.1 Meta-Language productions

<table>
<tr><th>Number</th><th>Name</th><th>Production</th></tr>
<tr><td>100</td><td>Terminal</td><td>Terminal symbols are written between quotes, e.g. “0”.</td></tr>
<tr><td>110</td><td>Non-Terminal</td><td>Non-Terminal symbols are denoted by italics, e.g. concept.</td></tr>
<tr><td>120</td><td>Alternatives</td><td>Alternatives are separated by a vertical bar |, e.g.
“0” | “1” means a 0 or a 1.</td></tr>
<tr><td>130</td><td>Groupings</td><td>Groupings are contained in curly braces { and }, e.g.
{A B C} means A followed by a B followed by a C, and
{A | B | C} means either an A, or a B or a C.</td></tr>
<tr><td>150</td><td>Options</td><td>Options are contained in square brackets [and], e.g.
[rlink] means an optional rlink and
[“–>” | “<–”] means –> or <– or neither.</td></tr>
<tr><td>160</td><td>Repetition</td><td>Repetition is denoted by ... following a symbol, group
or option, e.g. cgraph... means 1 or more
cgraphs, { }... means 1 or more of the group or
alternatives within a group, and [ ]... means 0 or more of
the option or alternatives within the option.</td></tr>
<tr><td>170</td><td>Spaces</td><td>Spaces, returns, linefeeds, tabs, etc. within a production
are treated as a single space except within terminals or as
otherwise explicitly noted by, e.g. a newline non-terminal.
Where such a space occurs, it implies zero or more spaces,
returns, linefeeds, tabs, etc. in the language being defined
which are also interpreted as a single space for parsing;
no space between successive terminals or non-terminals is
interpreted as no space in the target language.</td></tr>
<tr><td>180</td><td>Comments</td><td>Comments in the meta and target languages are denoted by a
double semicolon where it and any other text remaining on
the line are interpreted as an end of line.</td></tr>
</table>

5.2.2 Low Level Productions

<table>
<tr><th>Number</th><th>Non-Terminal</th><th>Production</th></tr>
<tr><td>200</td><td>digit</td><td>“0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”</td></tr>
<tr><td>202</td><td>number</td><td>digit...</td></tr>
<tr><td>204</td><td>letter</td><td>“A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” |
“J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” |
“S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z” |
“a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” |
“j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” |
“s” | “t” | “u” | “v” | “w” | “x” | “y” | “z”</td></tr>
<tr><td>206</td><td>punctuation</td><td>“[” | “]” | “(” | “)” | “{” | “}” | “-” | “<” |
“>” | “@” | “#” | “*” | “,” | “.” | “:” |
“ ” | “~” | tab | lambda | forall</td></tr>
<tr><td>208</td><td>eol</td><td>[comment] {eof | return | newline}</td></tr>
<tr><td>210</td><td>other-char</td><td>character-other-than-digit-letter-punctuation-or-eol</td></tr>
<tr><td>220</td><td>name-char</td><td>digit | letter | other-char</td></tr>
<tr><td>230</td><td>text-char</td><td>name-char | punctuation</td></tr>
<tr><td>240</td><td>text</td><td>text-char...</td></tr>
<tr><td>250</td><td>comment</td><td>“;;” text</td></tr>
<tr><td>260</td><td>string</td><td>“"” text “"”</td></tr>
<tr><td>270</td><td>identifier</td><td>letter[name-char]...</td></tr>
<tr><td>280</td><td>name</td><td>{letter | digit}[name-char]...</td></tr>
<tr><td>290</td><td>long-name</td><td>name{“ ”...name}... ;;a sequence of names</td></tr>
</table>

5.2.3 Conceptual Graph Productions

Number	Non-Terminal	Production
300	*cgraph*	*c-node* | *relation conlink* | *actor conlink*
310	*c-node*	*concept* [*rlink*]
312	*r-node*	*relation* [*conlink*]
314	*a-node*	*actor* [*conlink*]
320	*concept*	*ind-concept* | *context* | *str-concept*
322	*ind-concept*	“[” [*typefield*] [*reffield*] [*annotation*] “]”
324	*typefield*	[*not*]*typelabel* | *lambda-exp*
326	*exist-ref*	“[” [[*typelabel*] “:”] “*” *generic-ref* “]”
328	*str-concept*	“[” *string* “]” ;;short for [WORD: *string*]
330	*relation*	“(” [*not*]*typelabel* [*annotation*] “)”
340	*actor*	“<” *typelabel* [*annotation*] “>”
350	*conlink*	*arc c-node* | “-” *conlist*
352	*rlink*	*arc r-node* | *arc a-node* | “-” *rlist*
354	*conlist*	{*newline arc c-node* }... “,”
356	*rlist*	{*newline* {*arc r-node* | *arc a-node*} }... “,”
360	*typelabel*	*long-name* [“@” *canon-name*]
362	*canon-name*	*name*
364	*annotation*	“;” *long-name*
370	*context*	[*not*] “[” *cgraph*... “]”
374	*not*	“~”
376	*lambda-exp*	*lambda* “(” *generic-ref* “)” *cgraph*
380	*arc*	[*number*] { “<–” | “–>” }
390	*generic-ref*	*identifier*

5.2.4 Referent Field

Number	Non-Terminal	Production
400	*reffield*	“:” [*names*] [*vars*] [*mark*] [*not*] [*referent*]
402	*names*	*name*[“=” *name*]...
404	*vars*	*variable*[“=” *variable*]...
406	*variable*	“*” [*identifier*]
408	*mark*	“!” | “?”
410	*referent*	*individual* [*measure*] | *generic-referent*
412	*individual*	“#” [*name*]
414	*measure*	“@” *long-name*
420	*generic-referent*	*set* [*quantity*] | *cgraph* | *literal* | *string* | *measure*
430	*set*	[“COLL” | “DISJ” | “DIST” | “RESP”] “{” *set-ref-list* “}”
432	*set-ref-list*	“*” | *set-ref*[“,” *set-ref*]...
434	*set-ref*	*name* | *variable* | *individual*
440	*quantity*	“#” | *forall* | “@” *long-name*
450	*literal*	“'” *long-name* | “'” *list*
452	*list*	“(” [*long-name* | *list*]... “)”

5.2.5 Extended Linear Forms

Number	Non-Terminal	Production
500	*lf-file*	$\{$*statement* $\mid$ *comment*$\}$...
502	*statement*	$\{$*cgraph* $\mid$ *other-stm*$\}$ "."
504	*other-stm*	*canon-stm* $\mid$ *type-stm* $\mid$ *relation-stm* $\mid$ *actor-stm* $\mid$ *schema-stm* $\mid$ *proto-stm* $\mid$ *selecting-stm* $\mid$ *individual-stm* $\mid$ *canonical-stm*
510	*canon-stm*	"CANON" *canon-name* ["IS" *cgraph*]
512	*type-stm*	"TYPE" [*not*]*typelabel arg* "IS" [*exist-ref*] *cgraph*
514	*relation-stm*	"RELATION" [*not*]*typelabel args* "IS" [*exist-refs*] *cgraph*
516	*actor-stm*	"ACTOR" *typelabel args* "IS" *cgraph*
520	*schema-stm*	"SCHEMA" ["FOR"] *typelabel args* "IS" *cgraph*
522	*proto-stm*	"PROTOTYPE" *typelabel args* "IS" *cgraph*
524	*selecting-stm*	"SELECTING" "GRAPH" ["FOR"] *typelabel* "IS" *cgraph*
526	*individual-stm*	"INDIVIDUAL" *typelabel ind* "IS" *cgraph*
528	*canonical-stm*	"CANONICAL" "GRAPH" "IS" *cgraph*
540	*arg*	"(" *identifier* ")"
550	*args*	"(" *identifier* $\{$","*identifier*$\}$... ")"
560	*exist-refs*	*exist-ref*...
570	*ind*	"(" ["#"]*identifier* ")"

5.3 Extensions to Linear Form

The grammar presented in the previous section already contains some extensions to a minimal linear form grammar. In this section we suggest some further possible extensions to this grammar.

Firstly, we consider some extensions mentioned by Sowa [38]. These extended linear forms will in fact be parsed by the grammar in section 5.2 — this table presents their equivalent linear forms.

Name	Contracted Form	Extended Form
name contraction	`[C:Name]` `[C:Name#1234]`	`[C]->(NAME)->[WORD:"Name"]` `[C:#1234]->(NAME)->[WORD:"Name"]`
measure contraction	`[C:@10m]` `[C:#1234@10m]`	`[C]->(MEAS)->[MEASURE:10m]` `[C:#1234]->(MEAS)->[MEASURE:10m]`
quantity contraction	`[C:{*}@2]`	`[C:{*}]->(QTY)->[NUMBER:2]`
negation	`(R)->~[C]`	`(R)->[C]<-(NEG)`
proposition contraction	`[PROPOSITION:G]`	`[PROPOSITION]->(STMT)->[GRAPH:G]`
situation contraction	`[SITUATION:G]`	`[SITUATION]->(DSCR)->[PROPOSITION]->(STMT)->[GRAPH:G]`

The next table presents further extensions, which for the time being aren't processed by the previous grammar.

Name	Contracted Form	Extended Form
empty set	`[C:{}]`	`[C:{*}]->(QTY)->[NUMBER:0]`
cardinality restrictions	`[C:{*}@>2]` `[C:{*}@2-4]`	`[C:{*}]->(QTY)->[NUMBER]->(GT)->[NUMBER:2]` `[C:{*}]->(QTY)->[NUMBER:*x]-` `->(GE)->[NUMBER:2]` `->(LE)->[NUMBER:4]`
subrange types	`[NUMBER>0:1]` `[SPEED<=55mph]`	`[@lambda(x) [NUMBER:*x]->(>)->[NUMBER:0] : 1]` `[@lambda(x) [SPEED:*x]->(MEAS)->[MEASURE]-` `->(<=)->[MEASURE:55mph]]`

Further extensions and changes to the grammar may be considered. Most of them need further discussion among the CG community. We therefore list several topics that we would like to see discussed at the workshop.

Special Symbols The current grammar doesn't define the *lambda* and *forall* non-terminals. The symbols "@lambda" and "@every" have been suggested by Sowa. However, ambiguity may arise in productions 360 and 440. Another suggestion might be to use the quote character, as for lists. Or one can simply introduce the notion of reserved identifier (as in Pascal) which can't be used otherwise. In this case, no special symbol like @ would be necessary.

Names and Identifiers If typelabels like `PHYS-OBJ` and `>` (for the greater-than relation) and variables like "*123" and "*x_1" are to be possible[7], the definitions of *name* and *identifier* (productions 270 and 280) must be changed or even merged together. Moreover, it would be nice to allow accents for an easier mapping between Natural Language and CGs. For example, the Portuguese word *acção* (action) could be mapped into $\boxed{\text{A,CC\~{}AO}}$. Notice that this isn't possible for the moment because both "," and "˜" are part of *punctuation* and as such they can't be considered as *other-char*. Also, it isn't possible to define the non-member function as `ACTOR ~member(element, list) IS`... and `SCHEMA FOR ~boat(x) IS`... isn't possible either because *typelabel* (production 360) doesn't include *not* (it is defined case by case in the other productions where *typelabel* occurs).

Other A graph like $\neg\boxed{\text{TAUTOLOGY: } \boxed{\text{*p}}\ \neg\boxed{\text{*p}}}$ is not possible because production 370 doesn't allow typelabels and coreference markers for *negated* contexts. It might be handy to allow (nested) multiple-line comments, especially for batch-oriented CG "compilers". Furthermore, the terminal "**" has been proposed for sets, instead of simply "*".

5.4 Comparison with PEIRCE Linear Form

The grammar in section 5.2 represents many years of work and most of the mainstream ideas on conceptual graphs. PEIRCE on the other hand, being only in its first release, uses a much smaller grammar.

In this section we first present the linear form productions of the grammar parsed by the current PEIRCE release and then contrast it with the grammar of section 5.2 pointing out some of the more significant differences between the two.

[7]They aren't for the moment because both "-" and ">" aren't part of *other-char* and as such can't appear in names, while identifiers can't begin with digits and can't include *other-chars*.

5.4.1 Grammar for PEIRCE 0.1

The following table presents the linear form grammar accepted by the current version of PEIRCE [8] [9] (see pp. 11-13).

<table>
<tr><th>Non-Terminal</th><th>Production</th></tr>
<tr><td>sentence</td><td>term “.” | term “?”</td></tr>
<tr><td>term</td><td>unconnected_graph | declaration</td></tr>
<tr><td>unconnected_graph</td><td>connected_graph | connected_graph unconnected_graph</td></tr>
<tr><td>connected_graph</td><td>node link</td></tr>
<tr><td>graph</td><td>node link</td></tr>
<tr><td>node(Type)</td><td>left_delimiter(Type) node_label(Type) right_delimiter(Type)</td></tr>
<tr><td>node_label(Type)</td><td>type | type “:” referent | variable | graphs</td></tr>
<tr><td>referent</td><td>variable | individual_marker | graphs</td></tr>
<tr><td>variable</td><td>“*” variable_name | variable_name</td></tr>
<tr><td>individual_marker</td><td>string | individual | ’#’ individual</td></tr>
<tr><td>individual</td><td>individual_first_char individual_suffix</td></tr>
<tr><td>individual_first_char</td><td>upper_case | digit</td></tr>
<tr><td>individual_suffix</td><td>empty | individual_suffix_char individual_suffix</td></tr>
<tr><td>individual_suffix_char</td><td>alphanumeric | “_”</td></tr>
<tr><td>graphs</td><td>graph | graph graphs</td></tr>
<tr><td>left_delimiter(CONCEPT)</td><td>“[”</td></tr>
<tr><td>left_delimiter(RELATION)</td><td>“(”</td></tr>
<tr><td>right_delimiter(CONCEPT)</td><td>“]”</td></tr>
<tr><td>right_delimiter(RELATION)</td><td>“)”</td></tr>
<tr><td>link</td><td>empty | arc node link | “-” links “,”</td></tr>
<tr><td>links</td><td>link_one_of_many | link_one_of_many links</td></tr>
<tr><td>link_one_of_many</td><td>“\n” next_link link</td></tr>
<tr><td>next_link</td><td>arc node | node arc node</td></tr>
<tr><td>arc</td><td>“–” | “–” type_label “–” | “–>” | “–” type_label “–>” |
“<–” | “<–” type_label “–”</td></tr>
<tr><td>variable_name</td><td>lower_case variable_name_suffix</td></tr>
<tr><td>variable_name_suffix</td><td>variable_name_suffix_char variable_name_suffix</td></tr>
<tr><td>variable_name_suffix_char</td><td>alphanumeric | “_”</td></tr>
<tr><td>type</td><td>type_label | type_label “(” feature_list “)”</td></tr>
<tr><td>type_label</td><td>first_type_label_char type_label_suffix. | string</td></tr>
<tr><td>first_type_label_char</td><td>upper_case_char | digit | “–”</td></tr>
<tr><td>type_label_suffix</td><td>empty | type_label_suffix_char type_label_suffix</td></tr>
<tr><td>type_label_suffix_char</td><td>alphanumeric | “_”</td></tr>
<tr><td>feature_list</td><td>feature_pair | feature_pair “,” feature_list</td></tr>
<tr><td>feature_pair</td><td>type_label “=>” feature_term |
type_label “=>” feature_term “:” variable |
feature_term | feature_term “:” variable</td></tr>
<tr><td>string</td><td>““” non_double_quoted_chars “””</td></tr>
<tr><td>declaration</td><td>type_label “<” type_labels | type_label “::” individual</td></tr>
<tr><td>typelabels</td><td>type_label | type_label “,” type_labels</td></tr>
</table>

5.4.2 Comparison with Standard Linear Form Grammar

Firstly, there are a number of things considered in the PEIRCE grammar that are not addressed by the grammar presented in section 5.2. These include:

- Declaration of subtype relationships[9] (e.g., `Type` $<$ `Type`$_1$`, ..., Type`$_n$)

[8] PEIRCE can in fact parse more than just conceptual graphs in linear form (see [9]) but we shall restrict ourselves to discussing the linear form portion of its grammar.

[9] Production 512 handles this partially.

- Declaration of conformity relationships[10] (e.g., `Type :: #1234`)
- Type labels on arcs denoting relations `[C1]-R->[C2]` (e.g., see [9] p.28)
- Feature terms (e.g., [9] p.28 `[WHITE KING(RANK=>0): *x2]`)

Among the things covered by our grammar but not as yet by PEIRCE are:

- Negated contexts
- Actors
- Reasonably complex referents — PEIRCE currently allows only generic and individual referents

There are also some other differences between the two grammars. In particular:

Feature	Grammar in Section 5.2	PEIRCE v0.1
Relation/Concept	Differentiated in grammar	Not differentiated in grammar but code checks that relations are connected to concepts and vice versa
Variables	Must begin with "*" followed by an identifier beginning with a letter	Must begin with lowercase letter may be preceded by "*"
Individuals	Must begin with "#" followed by an name beginning with a letter or digit	Must begin with uppercase letter or digit may be preceded by "#"
Type Label	Series of names beginning with a letter or digit. May also appear negated	Must begin with uppercase letter or be a string

5.5 Conclusions

In this report we have investigated a number of issues concerning the linear form of a conceptual graph. We have presented a fairly comprehensive grammar that represents many of the mainstream developments in conceptual graphs over the years. This grammar was compared with that of PEIRCE 0.1 to give an idea of the fundamental differences between the two. Regarding the integration with KIF, the minimal change that should be incorporated in CGs is the addition of "?"*identifier* as a form of variable as well as "*"*identifier* as now allowed. The semantics would be that "?"*identifier* refers to a previously defined variable, if any, while "*"*identifier* defines a new scope for that variable.

Our agenda for future work includes:

- Clean up the grammar making a distinction between symbols and separators.
- Maybe add a general purpose macro mechanism for contracted and expanded forms.
- Augment and adapt the descriptions and the examples contained in [14] to the new grammar.
- Write some graph examples both in standard and PEIRCE notation in order to show the differences between both grammars more clearly.

[10] Production 526 and the occurrence of individuals in concepts handle this.

5.6 Acknowledgements

The authors would like to thank Gerard Ellis for providing a copy of the PEIRCE linear form grammar included in section 5.4 and John Sowa for letting us use his notes on interfacing CGs with KIF.

5.7 Appendix: Future Directions — Interfacing with KIF

Last week[11], there was a meeting with Mike Genesereth, Richard Fikes, Jim Fulton, Bob Neches, and me about the relationship between KIF, CGs, and SUMM (Fulton's Semantic Unification Meta-Model). Our conclusion was that we would propose both KIF and CGs as "normative languages" for the IRDS Conceptual Schema and we would adopt the SUMM ontology for metalanguage definitions.

That means that one critical issue is to get KIF and CG syntax lined up in a one-to-one correspondence wherever possible. There will be many cases where we have to map one feature in one of the languages into an expression in the other, but we would like to minimize those occurrences. I'm all in favor of adopting ideas from LIFE and other sources, but any syntactic constructions that might create additional complexity in the KIF $\leftrightarrow$ CG mapping should be avoided. I think that we should encourage the LIFErs to map their syntax to the Peirce software, but we should avoid cluttering up the CG syntax with too many additional conventions.

The basic syntactic unit in KIF is the *word* as defined by the Common Lisp reader (i.e. the blank and the parentheses are the main delimiters). Words are classified in the following categories:

Variables Words whose first character is "?" or "@". There are two kinds:

1. Individual variables begin with "?": `?x ?y ?abc_123`
2. Sequence variables begin with "@": `@x @y @abc_123`

Operators Words that are defined as operators, but not distinguished by any special character. There are 4 kinds:

1. Term operators: `listof setof quote if cond the setofall kappa lambda`
2. Sentence operators: `= /= not and or => <= <=> forall exists`
3. Rule operators: `=>> <<= consis`
4. Definition operators: `defobject defunction defrelation := => :&`

Constants Any word that is neither a variable nor an operator. There are four kinds, but they are not distinguished by any special syntax: object constants, function constants, relation constants, and logical constants.

Following is some discussion about the possible ways of representing these KIF constructs in CGs:

Variables Since both KIF and CGs use a prefix symbol for marking variables, I would be reluctant to do anything that would make CGs diverge farther from KIF. Prolog and LIFE are much less important for the standards issues than KIF. Following are some possibilities for extensions to CG notation:

1. Keep "*" as the marker for CG variables and map it to "?" for KIF individual variables. The sequence variables in KIF occur only at the end of a term, and I think that we can represent them with a slight extension to the current CG notation for sequences: `(+ ?a ?b ?c)` becomes `[NUMBER: <*a, *b, *c>]->(+)->[NUMBER]` and `(+ ?a ?b @c)` becomes `[NUMBER: <*a, *b : *c>]->(+)->[NUMBER]`. The vertical bar could be used, as in Prolog, to show that the last variable unifies with the tail of a list. That would eliminate the need for a special marker to distinguish individual variables from sequence variables.

[11]This appendix is the (slightly edited) message sent by John Sowa to the CG mailing list on February 10, 1993.

2. Adopt the KIF "?" for an individual variable and "@" for sequence variables. The "*" could be used as the prefix for quantifiers; e.g. `*` by itself would be "exists", `*every` would be equivalent to the KIF `forall`, `*1` would mean exactly one, etc.
3. Adopt the KIF "?" for variables, the notation <?a,?b:?c> for sequences, use "*" as the quantifier prefix, and free up "@" for some other purpose.

One point that came up last week was the problem of *capturing* a CG variable when a graph that happened to contain the same variable name was copied into a context. In predicate calculus or KIF, a new occurrence of an existential quantifier introduces a new variable, even if the name happens to be the same; e.g. $(\exists x)(cat(x) \wedge (\exists x)mat(x))$. The two x's in that formula would be distinct. This is bad coding practice, but it might arise in computer-generated expressions. If we had separate symbols for the quantifier and the variable, we could write `*?x` in the context where the variable was defined, and just `?x` at subsequent references. This convention would simplify the mapping from KIF to CGs, and it might be useful in CGs by themselves.

Operators 1. Term operators: Some of these would map to CG expressions in the referent field; `listof` becomes `<,,,>`; `setof` becomes $\{,,,\}$; `setofall` becomes `*@all`; `lambda` becomes `@lambda` or `*lambda`; `the` becomes `#`; etc. Others like `if` and `cond` would become relations.

2. Sentence operators: The quantifiers would become `*every` and `*`; `= /= or <=>` would become dyadic relations; `not` would become `NEG` or some single character to be determined; `=>` would be the current "IF-THEN" construction; and we could add a backwards "IF" for `<=`.

   Note: When an upside-down "A" is available in the character set, it may be used for the universal quantifier. If it is not available, I have been using `@every` instead of `@forall` because the usual occurrences of `@every` map more directly to the English word "every". I would prefer to use `@all` in combination with $\{*\}$ to represent the KIF `setofall`. Whether we use "@" or "*" for the quantifier prefix depends on the escape character for variables.

3. Rule operators: These would become relations that apply to contexts.
4. Definition operators: These would also become relations that link contexts, but most of them could be defined in terms of lambda.

Constants Since the CG referent field already contains some special symbols, I think that we would have to be more restrictive. I would suggest the following categories:

1. Simple names: Any string of alphanumeric characters plus the underscore that starts with a letter or "_": `John elephant abc_12 _123`
2. Complex names: Any string of characters delimited with single quotes: `'John' 'elephant' 'Grand Central Terminal' '+' '**'`

   We could treat `'John'` and `John` as equivalent. There is a question about whether the upper/lower case distinction should be significant in names and variables. I think that KIF and CGs should adopt the same conventions on this point.

3. Numbers: Whatever the ANSI standards people want to define as a recognizable number for CG and KIF: `23.4 -1.06e+2 0x90AB`
4. Strings: Any string of characters delimited with double quotes: `"John" """" "23.4" "This is a sentence."`

   We should consider what to use as an escape character; i.e. something like `\n` for newline, `\t` for tab, etc. Again, this should be the same for KIF and CGs.

6

Object-Oriented Design of Conceptual Graph Processor

C. Munday and D. Lukose[1]

ABSTRACT
In this paper, the authors describe the result of a comparative study on the features of existing Conceptual Graph Processors, by reviewing the soundness, completeness, canonicity and extendibility of these systems, identify the desirable and the undesirable features, and the limitations of these systems. This comparative study enabled the authors to identify the desirable features of these systems. These features are than incorporated into a new Conceptual Graph Processor that they are developing. The design paradigm adopted is Object-Oriented, and the system is currently in development stage.

6.1 Introduction

Since the publication of John Sowa's seminal work on the Conceptual Graph(CG) Theory [37], there have been a large number of researchers carrying out extensive research and development on various theoretical and practical issues in utilising this knowledge representation scheme for building knowledge based systems. Several research groups have developed conceptual graph processing tools. Among the most notable ones are the Deakin Group [17], the Loughborough Group (private communications)[2] and the PEIRCE Group [12] [25]. Refer to Appendix A for more information on the toolsets developed by these research groups. These research groups have developed highly sophisticated toolsets that they have successfully used on various research projects. Each of these tools demonstrate unique attributes which highlights the levels of it's sophistication, usability, and knowledge base consistency.

In this paper, the authors describe an object oriented architecture for a Conceptual Graph Processor. This tool, currently in development at the University of New England is made up of two major components: Conceptual Graph Editor and Conceptual Graph Processor. In a nutshell, the objective of this project is to build a knowledge base editor and a processor for Conceptual Graphs, by taking in account the following desirable features:

- Soundness - implement the CG formalism as close as to the theory as described in [37].
- Completeness - represent and interconnect all component of the CG knowledge base and implement the Canonical Formation Rules and other advanced graph operations.
- Canonicity - implement appropriate canonicity checking procedures in both the editor and the processors.
- Extendible - adopt a design and programming methodology that is easy to modify and extend in future.

In Section 2 we carry out a comparative evaluation of the completeness of three existing Conceptual Graph ToolSets. This is followed by a description of the Object-Oriented architecture of the system we are developing. In Section 4, we review the detailed design of the Objects used in our architecture, and we conclude this paper with a brief description of the future directions of this project.

---

[1]Department of Mathematics, Statistics, and Computing Science, University of New England, Armidale, 2351, N.S.W. AUSTRALIA. {cmunday | lukose}@neumann.une.edu.au

[2]Experience on the Loughborough ToolSet was gained from the interactions of one of the author with the Conceptual Graph research group at the Loughborough University of Technology, during his Leverhulme Fellowship (1992-1993) at the university.

6.2 Comparison of Existing Conceptual Graph Processors

It is not a difficult task to compare the existing systems. We review the three existing systems by comparing the following features:

- Representation Completeness
- Knowledge Base Components
- Canonical Formation Rules
- Advance Graph Operators
- Auxilary Features

Each of the above features were then sub-divided in a more detailed list as shown in Appendix B. In addition to the comparisons of these features, the authors also attempted to identify a set of desirable and undesirable features of these systems. We shall attempt to categories them below:

(a) **Deakin ToolSet**

Desirable

- Highly extendible - easy to incorporate new abstractions (e.g., incorporation of Actor Graphs and the Problem Maps)
- Demonstrates clearly in a very simple manner the inter-relationships between the components of the conceptual graph knowledge base.
- Has a Knowledge Processing Language (Control Script)

  Structured English Commands + Variable Bindings + Control Structure = Control Script

Undesirable

- Only binary relations.
- No Generalisation Hierarchy
  - difficult to identify the following:
    - common generalisation
    - common specialisation
  - searching for a graph or any form of abstraction is time consuming, and this slows down all operations.

(b) **Loughborough ToolSet**

Desirable

- Concept and Relation Catalogue
  - very useful for canonical checking.
- Performs canonical checking when modifying/adding graphs into the knowledge base.

Undesirable

- Too much canonical checking is done when building the knowledge base. The system tends to get into a continuous type and/or relation definition process, thus very slow in building the knowledge base.
- No Generalisation Hierarchy
  - same problem as the Deakin ToolSet.

(c) **Peirce**

Desirable

- Generic POSET data structure
  - has Generalisation Hierarchy (used as the Database).

- Bit Encoding of Nodes in the Generalisation Hierarchy.
- Generic Graph Matching Algorithms
- Very fast and efficient graph searching and matching algorithms.

Undesirable

- The current version of PEIRCE has three data structures: TypeLattice, Generalisation Hierarchy, and ConformityRelation. A further Poset subclass have to be created for the relation hierarchy to be consistent with the original proposal of Sowa [37], where there should be at least the following three partial ordered sets:
  * Generalisation Hierarchy for storing Conceptual Graphs;
  * Type Hierarchy for representing the inter-relationships between the types; and
  * Relation Hierarchy for representing the relations.
- Difficult to realise the canonical checking procedures when it is difficult to establish the inter-relationships between the knowledge components in the knowledge base.

6.3 Design of the System

6.3.1 Research and Development Methodology

The research and development methodology that has been adopted for the development of this systems is a six phase process, with the last three phases being highly iterative, as shown in Figure 1.

6.3.2 Adopted Design

The proposed system has three major components: processor, editor and the knowledge base, as shown in Figure 2.
The knowledge base will contain all the components that make up the conceptual graph knowledge base. They are:

- Type Hierarchy
- Relation Hierarchy
- Generalisation Hierarchy
- Conformity Relation
- Concept type and relation type definitions
- Concept and Relation Catalogue
- Schema
- Prototype
- Composite Individuals

There are only seven major data structures which holds the knowledge base together. The main data structure is the POSET (Paritally Ordered SET). There are three instances of POSET: Type Hierarchy, Relation Hierarchy and Generalisation Hierarchy. We have developed three other data structures, one each for the nodes in each of the hierarchies, and we also have developed suitable data structures for concepts, relations and conceptual graphs. All of these data structures will be described in detail in Section 4. The processor will posses all the graph operations that are described in [37]. These operations are sub-divided into two categories: Primitive Operations and Advanced Operations. The primitive operations include:

- Copy
- Restrict
- Join

(a) Study the existing editors and processors to obtain the following:

- (i) identify the architecture of the Conceptual Graph Knowledge Base
- (ii) identify the design paradigm
  (e.g., relational vs object-oriented)
- (iii) identify the inter-relationships between all the knowledge components in a Conceptual Graph Knowledge Base
- (iv) identify canonical checking procedures
- (v) identify the algorithms for the Canonical Formation Rules
- (vi) identify algorithms for Maximal Join, Projection, and {Concept | Relation} {Contraction | Expansion}

(b) Adopt the best features among the systems studied

(c) Design a Prototype System

(d) Build the Prototype System

(f) Redesign Prototype System

(e) Evaluate Performance

FIGURE 6.1. Research and Development Methodology

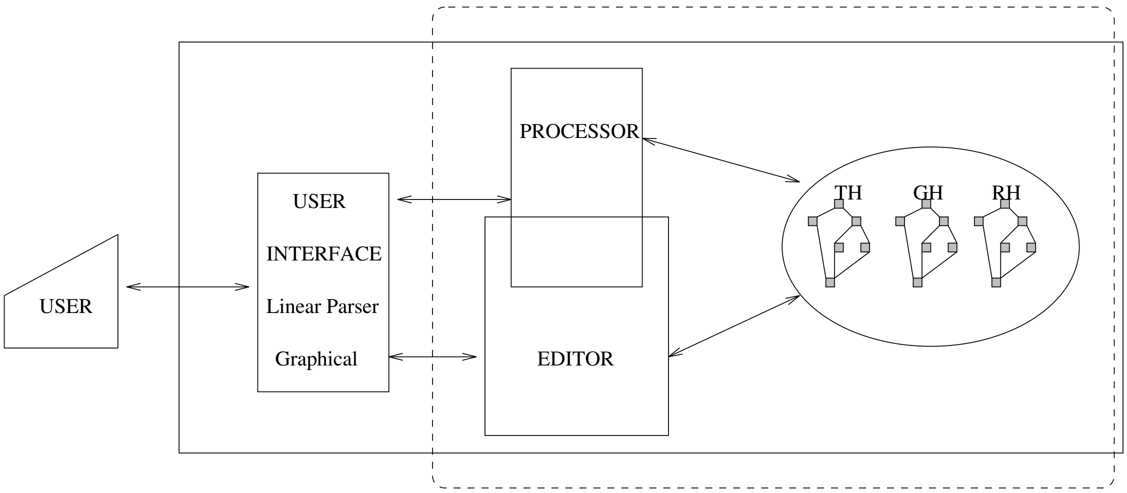

FIGURE 6.2. Architecture of the system

- Simplify.

These operations form the Canonical Formation rules. The advanced operations are:

- Type Expansion and Contraction (Concept and Relation)
- Maximal Join
- Projection

The editor will comprise of function/procedures for constructing all of the components of the conceptual graph knowledge base (as described earlier), and these function will also contain relevant routines to maintain the integrity of the knowledge base.

6.4 Object Descriptions

6.4.1 The Poset Structure

The POSET data structure is considered the key to the overall design. The speed of insertion and retrieval of objects is crucial to the usability of the system. Ellis [10] has made the most notable contribution to increasing the speed of these operations. In [10], a method of bit encoding the nodes in the poset was used to dramatically reduce the number of comparisons needed to find or insert objects.
Currently our design does not implement this method of traversal, we prefer a more simpler method named "Enhanced Search" as described in (Wermelinger, 1993) which we intend to modify at a later date to take advantage of the work done in [10]. The "Enhanced Search" as Wermelinger described is a topological traversal of the poset. It has been shown empirically that topological traversal works best (leads to less comparisons) with real knowledge bases.
The operations to be performed on the poset structure include:

- exists
- insert
- remove
- immediate predecessors
- immediate successors

The design of the poset class structure is presented in Figure 3.
The exists operation is merely a topological traversal of the poset. The algorithm is taken from Wermelinger [45]. Visiting each level in turn is achieved by the use of a priority queue. The elements in the queue are ordered according to their level in the poset. The priority queue is implemented as an array of n stacks, where n is the depth of the poset. An element on the n^{th} level of the poset in enqueued on the n^{th} stack. Once a stack i is empty, elements are popped off the next non-empty stack j, where $j > i$. The enqueing and dequeuing of elements is a constant time operation if the level is known before hand. The level for a newly inserted object is computed by the maximum level of it's parents +1.
Insertion and removal of an element is trivial, once its parents (immediate predecessors) and children (immediate successor) are known. Therefore, the operations are divided into two phases, the search for parents and the search for children. The algorithm again uses a priority queue to walk to the immediate predecessors.
Once the parents are known we can use this information to find the children, as the children will be found in the intersection of the subhierarchies under each parent. Once an element is found to be the child of an object, its subhierarchy can be removed from the search as these elements cannot be immediate successors.

6.4.2 POSETs In The Knowledge Base

The knowledge base will consist of three instances of the poset class, one for each of the hierarchies: Type Hierarchy, Relation Hierarchy and the Generalisation Hierarchy. The nodes in each hierarchy are described in Section 4.3. The three hierarchies are interconnected in a complex manner as described in Section 5.

```
template <class T> class poset
{
   protected:
      posetNode<T> *top;
      posetNode<T> *bottom;
      int currentToken;

      void newQuery()
      int see(posetNode<T> *u)
      boolean seen(posetNode<T> *u)
      void unsee(posetNode<T> *u)
      int match(posetNode<T> *u)
      boolean matched(posetNode<T> *u)
      void unmatch(posetNode<T> *u)

      void addChildrenToSearch(posetNode<T> *u, \
                               priorityQueue<posetNode<T> *> &Q);
      boolean parentsMatched(posetNode<T> *);
      BitSet ORing(Set<posetNode<T> *> &);
      void visitSubHierarchy(posetNode<T> *);
      void removeLinks(Set<posetNode<T> *> &, Set<posetNode<T> *> &);
      int nextLevel(Set<posetNode<T> *> &u);
      void propagateLevel(posetNode<T> *u);


   public:
      poset(T topElement, T bottomElement);
      ~poset() {}

      boolean isMember(T u);
      void insert(T u);
      void insert(T u, Set<posetNode<T> *> &parentSet, \
                       Set<posetNode<T> *> &childSet);
      void insert(T superType, T u);
      void remove(T u);
      Set<posetNode<T> *> parents(T u);
      Set<posetNode<T> *> children(T u);
      Set<posetNode<T> *> children(T u, Set<posetNode<T> *> parents);
      void printOn(ostream &out);

};
```

FIGURE 6.3. Definition of the class poset.

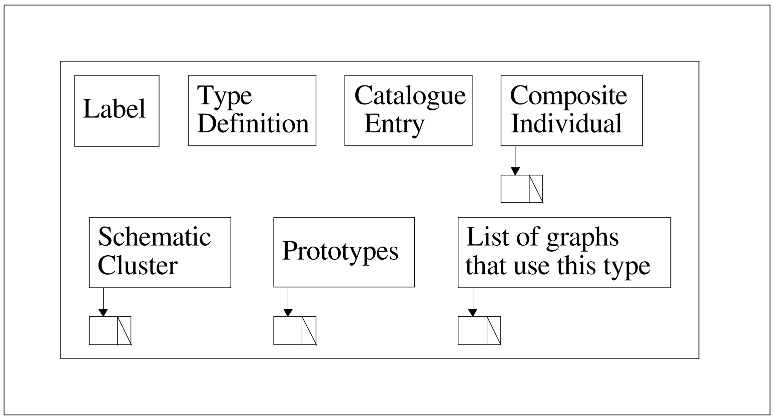

FIGURE 6.4. Diagram of a conceptual type.

6.4.3 POSET Nodes

A node in the type hierarchy contains a large amount of information relevant to a conceptual type. This information includes:

(a) Type Label

(b) Type Definition

(c) Conceptual Catalogue

(d) Composite Individuals

(e) Schematic Cluster

(f) Prototypes

(g) List of graphs that use the type

Components (b) and (c), are merely pointers to conceptual graph objects, while components (d) to (g) are lists of pointers to conceptual graph objects. All conceptual graphs will appear in the generalisation hierarchy. A class definition for a conceptual graph is presented in Section 4.6. A diagram and class definition of a conceptual type is presented in Figures 4 and 5, respectively.
The nodes in the relational hierarchy will contain information relevant to a relational type. In this case each type will contain the following information:

(a) Type Label

(b) Relational Definition

(c) Relational Catalog

(d) List of graphs that use the type.

As in a conceptual type, the components (b) and (c) are pointers to conceptual graph objects, and component (d) is a list of pointers to conceptual graph objects. This information can be seen visually in Figure 6, and is defined using C++ in Figure 7.

6.4.4 Data Structure For Concepts

The concept in a graph will contain a type, a referent, and an unique identifier for the purpose of resolving ambiguities when referring to a concept. The conceptual type is a pointer to a node in the type hierarchy (the definition of the node was given in Section 4.3). Having a pointer into the type hierarchy to define the concept's type allows immediate access to everything that is known about the type. This reduces the search time to zero and thus greatly increases the speed of some operations.
The referent of a concept as described in [37] can take many forms, this is handled by defining a base class 'referent', and from this, derived classes for each type of referent. Figure 8 depicts the architecture and Figure 9 outlines the class definition.

```
class conType
{
   private:
      label         *conceptLabel;
      cg            *typeDefinition;
      cg            *conceptCatalogue;
      list<ind *>    compositeIndividuals;
      list<cg *>    schematicCluster;
      list<cg *>    prototypes;
      list<cg *>    listOfGraphs;

   public:
      conType() { conceptLabel = NULL; }

      label &getLabel() { return *conceptLabel; }
      void setLabel(label *theLabel) { conceptLabel = theLabel; }

      void setDefinition(cg *);
      void setCatalogue(cg *);
      void addIndividual(cg *);
      void addSchema(cg *);
      void addPrototype(cg *);
      void addToListOfGraphs(cg *);

      cg *getDefinition();
      cg *getCatalogue();
      cg *getIndividual(unsigned long);
      cg *getIndividual(label);
      list<cg *> getSchemaCluster();
      list<cg *> getPrototypes();
      list<cg *> getListOfGraphs();
};
```

FIGURE 6.5. Class definition for a conceptual type.

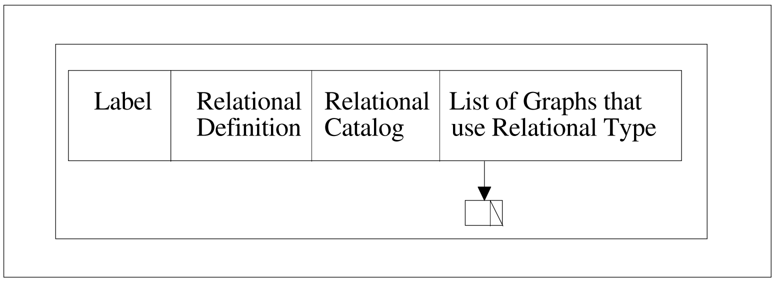

FIGURE 6.6. Diagram of a relational type.

```
class relType
{
   private:
      label       *relationType;
      cg          *relationDefinition;
      cg          *relationCatalog;
      list<cg *>  listOfGraphs;

   public:
      relType();

      label &getLabel();
      void setLabel(label *);

      void setDefinition(cg *);
      void setCatalog(cg *);
      void addToListOfGraphs(cg *);

      cg *getDefinition();
      cg *getCatalogue();
      list<cg *> getListOfGraphs();
};
```

FIGURE 6.7. Class definition for a relational type.

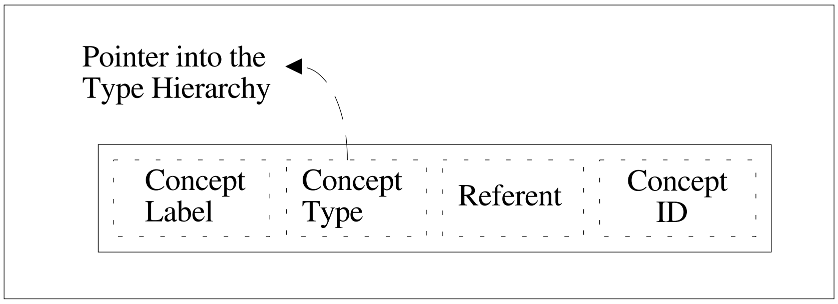

FIGURE 6.8. Diagram of the concept class

```
class concept
{
   private:
      label       *conceptLabel;
      conType     *conceptType;
      referent    *myReferent;
      unsigned long conceptID;

   public:
      concept()
      ~concept()

      unsigned long getID();
      void setLabel(label *L);
      void setType(conType *C);
      void setReferent(referent *Ref);
      label *getLabel();
      conType *getType();
      referent *getReferent();
      void clearLabel();

      void clearType();
      void clearReferent();
};
```

FIGURE 6.9. Definition of the Concept class

6.4.5 Data Structure For Relations

A relation will contain a relational type, an unique identifier and a list of pointers to concept objects. This list of pointers represents the incoming and outgoing arcs of the relation. The relational type is a pointer to a relational node in the relational hierarchy as described in Section 4.3. The unique identifier again provides a means to resolve the ambiguities encountered when referring to a relation. The list of pointers to concept objects provides a means to represent n-adic relations even though the current status of the processor only handles binary relations. We can identify the concepts that point into and out of the relation as, it is always the n^{th} arc that points away.
The pointer into the relational hierarchy serves the same purpose as in the case for concepts. That is, to have access to all information that is known about the relational type. The diagram and definition of a relation is given in Figures 10 and 11, respectively.

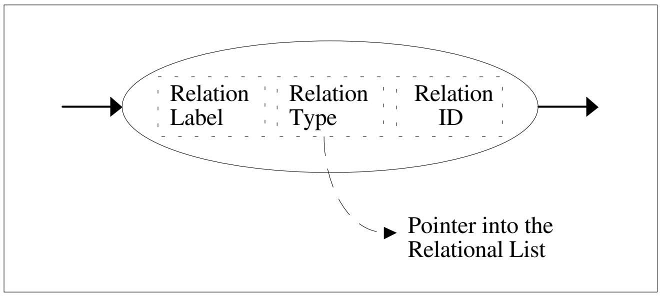

FIGURE 6.10. Diagram of the Relation class

```
class relation
{
   private:
      unsigned long relationID;
      label   *relationLabel;
      relType   *relationType;
      list<concept *> arcList;

   public:
      relation()
      ~relation()
      unsigned long getID();
      void setLabel(label *L);
      void setType(relType *R);

      label *getLabel();
      relType *getType();

      void clearType();
      void clearLabel();

      void addArc(concept *from, concept *to);
      list<concept *> getArcList();
};
```

FIGURE 6.11. Definition of the Relation class

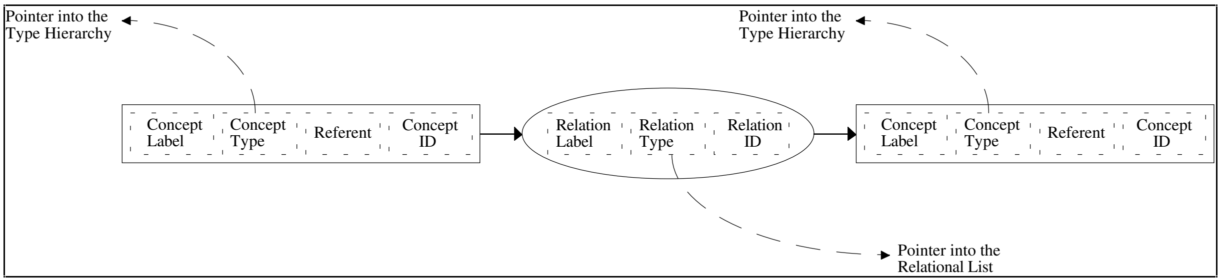

FIGURE 6.12. Diagram of the Conceptual Graph class

6.4.6 Data Structure For Conceptual Graph

The conceptual graph class consists of a label representing the name of the graph, a list of concepts and a list of relations, as shown in Figure 12 and its definition in Figure 13. A conceptual graph contains a list of concepts and a list of relations, with each maintaining a pointer into their respective hierarchies. Effectively, maintaining pointers into the hierarchies places all the information that is known about each type into the graph itself.

6.5 Architecture Of The Knowledge Base

The architecture of the knowledge base appears simple in structure, containing only three similar objects at the top level, which are the type hierarchy, relational hierarchy and generialisation hierarchy. The complexity comes in the manner that these three objects are inter–linked. Figure 14 depicts the use of the poset structure in the storage of the knowledge base components.
The links between the relational hierarchy and the generalisation hierarchy is shown in Figure 15. Each relation contained in a conceptual graph points to a type in the relational hierarchy. For example, the

```
class cg
{
   private:
      label *name;
      list<concept *> conceptList;
      list<relation *> relationList;

   public:
      cg();
      ~cg(){}

      void insertConcept(concept *);
      void insertRelation(relation *);
      list<concept *> getConceptList();
      list<relation *> getRelationList();

      boolean isEmpty();

      boolean isSubtypeOf(cg);

      label &getName() { return *name; }
      void setName(label *L) { name = L; }
};
```

FIGURE 6.13. Definition of the Conceptual Graph class

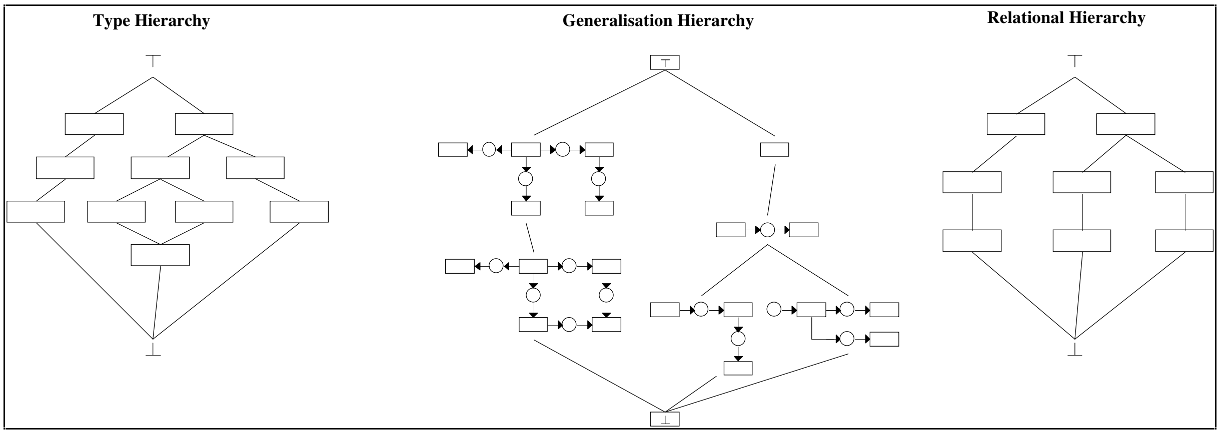

FIGURE 6.14. Conceptual Graph Knowledge Base

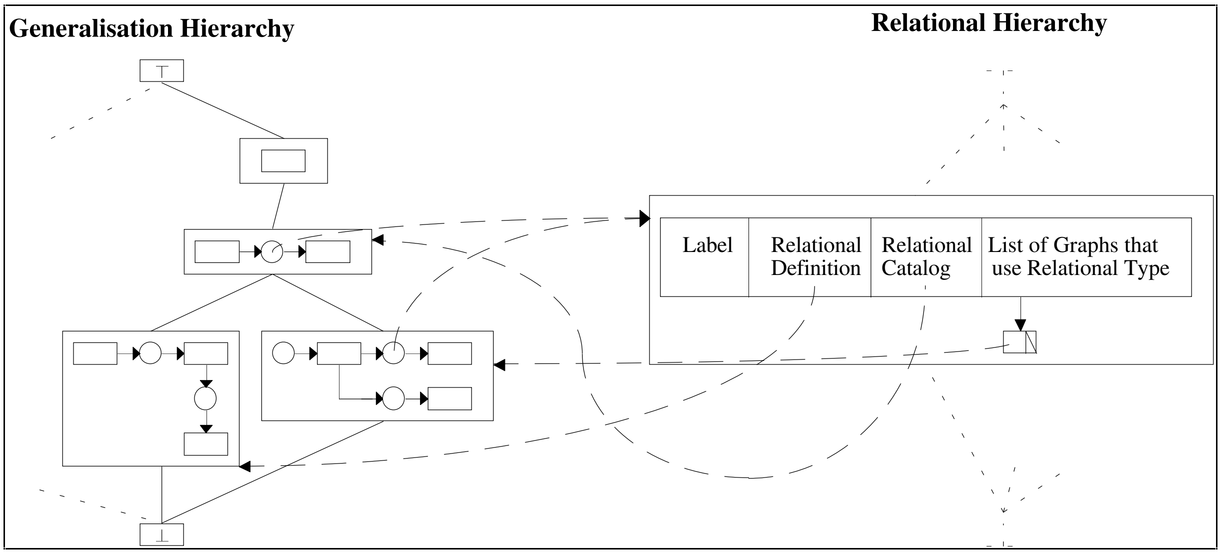

FIGURE 6.15. Links between Relational Hierarchy and Generalisation Hierarchy

relational definition points to the differentia graph that defines the relation, the relational catalog pointer points to the graph representing the relational constraints. The list of pointers to the graphs are used to maintain referential integrity. A type shouldn't be deleted if it is being used.
A similar case exists for the relationship between the type hierarchy and the generalisation hierarchy, as shown in Figure 16. The type definition pointer in the type node of the type hierarchy points to the differentia graph for the particular type.

6.6 Conclusion

With the completion of the objectives outlined in this paper, we will have at our disposal (and it will be available to the general Conceptual Graph community) a research tool that is implemented as close to the original theory as proposed by Sowa [37]. Once the prototype is available, the authors intend to carry out extensive evaluation on the performance of this system. During this process, we intend to identify the design flaws, limitations and possible improvements for future research and developments. Further to this, we intend to add the following components to enable a flexible user interface:

- Linear Parser
- Graphical User Interface

Future developments will involve the following:

- Development of improved data structures for the nodes in the POSET (where required).
- Incorporate bit-encoding formalism to improve the speed for matching and retrieval of objects.
- Design an improved data structure for the class "Conceptual Graph".
- Extend the current architecture to incorporate the following:
  - n-ary relations.
  - Lambda expression to replace the "concept type" in a concept.
  - Improve on the concept and relation catalogue.

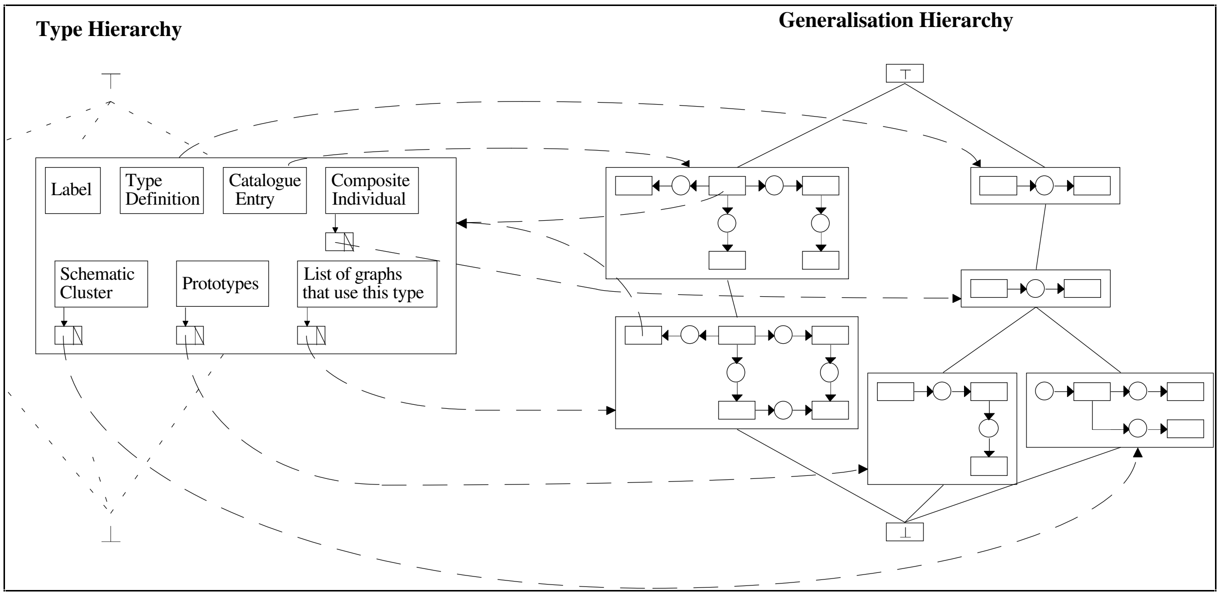

FIGURE 6.16. Links between Type Hierarchy and Generalisation Hierarchy

6.7 Acknowledgement

The authors wish to express their gratitude to the members of the graduate class in Conceptual Structures at UNE in 1994 for their valuable discussions during the initial stages of the design of this system. The authors also appreciate the valuable advice and contributions made by G. Ellis and B. J. Garner. We thank Allan Williams and Margaret McDonald for their expertise in LaTex.

6.8 APPENDIX A - Previous Work

Deakin ToolSet

Development Year:	1985 - 1992
Principle Developer:	Prof. B. J. Garner, Dr. E. Tsui, Mr. D. Lui, and Dr. D. Lukose
Modules:	Knowledge Based Editor Canonical Graph Processor Interactive Concept Classifier Rule Acquisition System General Purpose Inference Engine Self-Organising Conceptual Dictionary Semantic Interpreter Agent System and Executable Conceptual Structures
Interface:	Structured English
Platform:	Sun OS, UNIX, MUprolog
Implementation:	Relational

Loughborough ToolSet

Development Years:	1986 - 1992
Principle Developers:	Mr. P. Kocura, Kai Kwong Ho, D. Moorhouse, G. A. Sharpe, and J. Heaton
Comments:	Several CG Systems developed over the years. Development still in progress.
Systems include:	Conceptual Graph Editor Canonical Graph Processor Theorem Prover
Interface:	Linear, Graphical
Platform:	MS-DOS, C++ Sun OS, UNIX, C++, Prolog
Implementation:	Object-Oriented, Relational

Peirce

Development Years:	1992 - now
Principle Developers:	Mr. G. Ellis
Systems include:	Conceptual Graph Editor Canonical Graph Processor
Interface:	Linear, Graphical
Platform:	Sun OS, UNIX, G++
Implementation:	Object-Oriented

6.9 APPENDIX B - Comparative Review

FEATURES	Deakin	Loughborough	Peirce
Representation Completeness			
Relation Representation	binary	binary	n-ary
Nested Graphs	yes	no	yes
Lambda Expression in Concept	no	no	no
Knowledge Base Components			
Generalisation Hierarchy	no	no	yes
Type Hierarchy	yes	yes	yes
Relation Hierarchy	no	no	yes
Conformity Relation	yes	yes	yes
Relation Catalogue	no	yes	no
Concept Catalogue	no	yes	no
Type Definition	yes	yes	no
Relation Definition	yes	yes	no
Composite Individual	yes	yes	no
Schemata	yes	yes	no
Prototype	yes	yes	no
Simple Graphs	yes	yes	yes
Canonical Formation Rules			
Copy	yes	yes	yes
Restrict	yes	yes	no
Simplify	yes	yes	no
Join	yes	yes	no
Advanced Graph Operators			
Maximal Join	yes	yes	no
Projection	yes	yes	yes
Type Expansion	yes	yes	no
Relation Expansion	yes	yes	no
Type Contraction	yes	yes	no
Relation Contraction	yes	yes	no
Auxilary Features			
Graph Operation on Nested Graphs	no	no	no
Interface	Structured English	Linear & Graphical	Linear
Scalibility	no	no	yes

7

Morph Interactive User Guide

August 14, 1994

7.1 Platform Operating Conditions

The Morph system has been tested under both DOS and UNIX, and should be operable under any other platform which contains a standard C++ compiler. The executable is named "morph" and is approximately 409K in size on UNIX and 161K on DOS. Memory use is dynamic. Therefore, under DOS, Morph can operate on all smaller domains given that the running parameters do not create usage over the 640K DOS limit. Under UNIX the system will dynamically use memory available on the machine. On DOS the system can learn about 50 games per execution, but can load and save any size database (unless it is gargantuan). On UNIX, given a machine with a large memory, Morph can run and learn

7.2 Rule Files

Morph requires standard ASCII files containing the rules of the game for which the system will play and learn for. Morph will prompt for the name of the game you wish the system to operate in, and will search the current directory for the named rule file (a directory path can also be given if the game file is not located in the executable path).

Current game rule files available are:

```
File     Game         Descript
++++     ++++         ++++++++
tic      TIC-TAC-TOE  An old favorite.

hexp     HEX PAWN     A 3x3 board, with pawns on each side.  Player
                      wins if he is first to get his pawns to the other
                      side of the board before his opponent.  Pawns
                      move one square forward and capture diagonally.

hex6     HEX PAWN 6   Same as Hex Pawn, except the game is played on
                      a 6x6 board, with 6 pawns to each side!

pennies  PENNIES      You are given a stack of 15 pennies.  You can
                      only remove 1,2, or 3 pennies at a time.  Players
                      alternate removing pennies.  If you are the last
                      person to remove the last penny, you win.

nim      NIM          You play with 3 stacks of sticks, the number of
                      sticks in each stack are 3,5 and 7 respectively.
                      You can remove any number of sticks from any
                      one stack when it is your turn to play.  The
                      player who removes the last stick from all the
                      stacks wins.

nim2     NIM 2        Same as above, except an additional Dynamic
                      Relation was added to the original rules
                      to allow and test for much stronger play.

hanoi    TOWERS OF    Towers of Hanoi.  A classic.  Given a stack
          HANOI       of three different size blocks and 3 pegs.
```

```
                         Restack the blocks from peg 1 to peg 3 without
                         ever putting a bigger block onto a smaller block.

8puzzle  EIGHT PUZZLE    Given 8 tiles numbered 1-8, you must reconfigure
                         the tiles in numerical order, but can only do so
                         by sliding them around given one empty space on
                         a 3x3 square grid.  The final goal is the
                         following configuration:

                                        1 2 3
                                        4   5
                                        6 7 8
```

7.3 Rule File Syntax

Morph can only read in game rule files that have been written in a structured manner, that describe the rules of the game in a way which the system can understand. We are currently in an ongoing process of enhancing this description process in order to be able to add more complex game domains such as Othello and Chess. We know the Morph system is capable of operating and learning in these more complex domains, but the rule translation and description syntax is still being worked on.

For the TPL version of Morph, only a very basic rules set has been defined. (The Vector version has an upgraded rules set, which will be utilized for the Othello and Chess domains.)

Without detailing the theory and nomenclature of the different items in a rules file (which we have described elsewhere) I will summarize the syntax and listing of a rule's file for a given game domain, using our current basic TPL rules set.

7.4 TPL Rules Set

Upon examination of the currently available rule's files, one will find all rules for previous mentioned domains basically define a game domain in this order:

```
1.  OBJECTS              -Primitive objects in domain
                          (such as X's, O's, Squares).
2.  STATIC RELATIONS     -Relations in game domain that never change.
3.  DYNAMIC RELATIONS    -Relations in game domain that do change.
4.  DYNAMIC DEFINED      -Relations in game domain that do change,
                          but change based on other Dynamic Relations.
5.  OPERATORS            -Operators used to change Dynamic Relations.
6.  TERMINALS            -Terminal Conditions of game domain.
7.  END                  -Used to end the Rules file.
```

All rule files begin at the very top with two integers, defining the minimum and maximum number of Agents that can operate in the game domain. Following this, one will find "comments, which Morph will ignore while reading in the rules. (Note that a comment currently must precede each Table description, due to the fairly primitive parsing mechanism we are currently implementing.)

Each rules file then begins with OBJECT descriptions. Using the game rules file "nim2" as an example, we have below:

```
%
% Object
%
0
STICKS
{ S0, S1, S2, S3, S4, S5, S6, S7}
{ S0, S1, S2, S3, S4, S5, S6, S7}

%
% Object
```

```
%
0
STACK
{ STACK1, STACK2, STACK3}
{ STACK1, STACK2, STACK3}
```

NIM 2 uses two sets of Objects, STICKS and STACK. Each Object description is first proceeded with the comment Object, then the integer 0 (which notifies the parser that an Object is being read in). The name of the OBJECT type is then declared, and can be any set of contiguous characters. This is done on the first line.
On the next line, the list of objects of the given type are declared, and again can be any set of unique contiguous characters. Brackets open and close the listing, and objects must be separated by commas. The final bracket must also be adjacent to the last Object listed.
On line two must be listed the flip value of listed Objects. In Tic-Tac-Toe for example, the flip object of X is O.
STATIC RELATIONS are listed next. For Nim2, two static relations were defined: GT and END_STICK. Below is a description for GT:

```
%
% Static Relation
%
2
GT<STICKS, STICKS>
{
  <S7, S6>
  <S7, S5>
  <S7, S4>
  <S7, S3>
  <S7, S2>
   .....      (dots added for brevity sake)
  <S1, S0>
}
```

Again we proceed the description with a 3 line comment (required), then the digit 2, which tells the parser a Static Relation is now being read. The name of the Static Relation is then listed GT (short for Greater Than), followed by triangle brackets that contain the argument types of the Static Relation. In this case, both arguments are STICKS, and the listing that follows, is a set of two argument tuples defining the Greater Than relation of all sticks compared to each other in the game domain.
The exact syntax must be followed carefully in order for the current system parser to work properly. Thus, a single opening bracket must first be placed on the line following the name-argument of the Static Relation, and then on each subsequent line a single Tuple set can be listed, opened and closed with triangle brackets, with legal Object names separated by commas in the middle. The entire set of Tuples then must be closed with a closing bracket on the last line.
DYNAMIC RELATIONS are defined in the exact same manner as STATIC RELATIONS with the one exception that the integer that defines the type of table being read in is ”1” instead of ”2”.
DYNAMIC DEFINED RELATIONS are special case Dynamic Relations which have the same syntax as an OPERATOR relation, but contain only pre conditions and no add or delete conditions. In nim2, there is one Dynamic Defined below:

```
%
% Dynamic Defined
%
3
EACH_STACK(STICKS: s1, STICKS: s2, STICKS: s3) =
pre: ON<s1, STACK1> && ON<s2, STACK2> && ON<s3, STACK3> }
add: }
del: }
```

Three line comment begins the table, followed by the identifying integer value 3. Next, is the name of the relation EACH_STACK, followed in parenthesis the types and names of variables used to define the relation. The exact syntax must be followed as seen above. On the second line is the set of pre-conditions listed, that will determine the type of Tuples (of ¡s1, s2, s3¿ variety) that will be added to this table. Each of the preconditions list either a Static or Dynamic Relation, followed by a list of variables or Objects that must be of the same type as the pre-condition Table types, which will then be matched and joined by the Morph system to create this table's tuples. There are no add or delete conditions, which are reserved only for OPERATOR table types. But the end bracket must be used as seen above on all three pre, add, and del conditions in order for correct parsing.

The OPERATOR class table then is the same as above, except add and delete conditions are also listed. (It is also important to note in both Dynamic Defined and Operator Tables, variables are only allowed to bind to one unique object for this TPL rules version.) In nim2 one Operator table is listed as follows:

```
%
% Oper
%
5
REMOVE1(STICKS: s2, STICKS: s3) =
pre: ON<s2, STACK1> && GG<s2, s3> }
add: ON<s3, STACK1> }
del: ON<s2, STACK1> }
```

Integer 5 declares to the parser this is an Operator class Table. Name of the table and arguments are then listed on the next line, and on following lines the pre, add, and delete conditions of the given Operator. Note that when multiple pre or add conditions are listed, they are separated by the logical ”&&”. The TPL version of the rules, only logical & is available. In the newer Vector version, logical operators have been expanded to ” ” NOT, and ”——” OR.

Finally, the TERMINAL conditions are listed for a game domain, after the OPERATOR tables. Terminal conditions are listed and have the same syntax as Dynamic Defined Tables, except the identifying parsing integer is 6. For nim2, there is one Terminal condition table:

```
%
% Terminal
%
6
TERMINAL(STICKS: s1) =
pre: END_STICK<s1> && ON<s1, STACK1> && ON<s1, STACK2> && ON<s1, STACK3>}
add: }
del: }

Note that ending bracket for the precondition is adjacent to the final
pre-condition Table listing, and there are no add and delete conditions.

Finally all rule files end with:

%
% End
%
8
```

To let the Morph system parser know the rules have been completed and actual play and learning can begin. Note that the above parsing and reading in of the rules by the system takes 2-3 seconds at most.

7.5 Command Line Arguments

The Morph executable contains optional command line arguments that can be used prior to execution. The command ”morph -u” will print a brief synopsis of command line options as follows:

```
Usage:
          [-i iterations]       Number of game iterations.
          [-l logfile]          Name logfile other than default.
          [-n]                  Turn off learning.
          [-q]                  Quiet screen report.
          [-s #full ply depth]  Turn on search.
```

```
[-u]                  Usage message.
[-z]                  Turn on Search Variation Verbosity.

default: -i 1 -s 1 -l NULL -q FALSE -z FALSE -n FALSE
```

```
The 7 command line options can be entered in any order.  If an
option has a required argument following it (such as iterations)
a space must be between the option and the argument.  For example,
"morph -i 3" the i option calls for game iterations, and the number
of iterations is 3.


The options in more detail are as follows:

[-i iterations]  "morph -i 1000" would tell Morph to play and/or learn
                 1000 games before ending the execution phase.  Dash i,
                 space, followed by an integral number equal to the
                 number of games.  Note that Morph loads and saves a
                 database once only, but does learn in between each game
                 iteration, modifying the loaded DB which is kept in
                 dynamic memory.

                 Default: 1 iteration.


[-l logfile]     Dash l followed by any name will tell Morph to open
                 a logfile which will record the game(s) it plays when
                 executing in an ASCII file.  Thus a written record
                 can be kept of any game.

                 Default: no logfile is opened.


[-n]             A command line switch to turn off the learning phase
                 of the system.  Morph will operate  normally except
                 the system will not learn during given execution, i.e.
                 the learn database will not be changed.

                 Default: FALSE, learning system on.


[-q]             This turns off all screen reports while Morph is
                 active.  Use this option when playing hundreds or
                 thousands of games in one learning session, otherwise
                 I/O actually slows down execution tremendously.

                 Default: FALSE, verbosity is on.


[-s #full ply]   Dash s determines the Search range of a Morph APS
                 Agent if selected during an interactive prompt.  The
                 current Search module is a full-depth Search system,
                 that will Search at the given ply specified in the
                 argument, e.g. "-s 4"  will Search 4 ply.  The current
                 TPL version is only working at 1 and 2 ply due to a
                 coding bug still being worked on.  The Search module
                 will work in all game domains that have been defined
                 in a rules set.
```

```
                Default: s 1, 1 ply Full Depth Search.


[-u]            Usage message which lists these command line options
                with a one line description.

                Default: no usage message.


[-z]            Search Verbosity.  This option will turn on a screen
                report of the actual ply search taking place by a
                Morph APS Agent.  It will show node evaluations and
                selection, along with Operator variations at each
                Search level and final Node-Operator selection for
                the game domain currently being explored.

                Note that during variation print out, the system will
                stop periodically awaiting a user input to continue.
                Type any letter and press return to continue.

                Default: FALSE, no search variation print out.
```

7.6 Execution and Interactive Prompts

To execute Morph, type "morph" followed by desired command line arguments. The system will then print the following:

```
> Welcome to Morph.
> Enter name of rules file:
```

Here, one simply enters the name of the rules file which contains the rules to the game you wish the system to play. The syntax of these rules are described in sections 3 and 4 of this paper. One can also enter a path if the rules file is not located in the same directory as the morph executable.

Next follows a list of Agent types that can be selected for the given domain and the maximum or minimum number of Agents that can or should be selected for the domain. Agents represent the players in the game, either of computer origin or human interactive. For example, after entering "tic" above, which puts Morph in the Tic-Tac-Toe domain, morph will print as follows:

```
> Exactly 2 agents must be selected for this Domain.

> 1) APS
> 2) APS Observer
> 3) Human.
> 4) Greedy.
> 5) Random.

> Agent #1:
> Agent #2:
```

To select agent types, enter the number corresponding to the Agent listed. Any combination of Agents can be selected in any order for the domain selected. Agent 1: APS, the Adaptive-Predictive-Search Agent has been described in several papers elsewhere by us and is the main learning/search Agent for the Morph system (but not necessarily the only one that could potentially be defined). Agent 3: Human, allows human interaction by interactive prompts during the course of the game, so by selecting 1 and 3, an APS agent will play against the Human user, the Human user being the second agent, second to move. Agents 4, and 5 are computer Agents but are non-learning. Agent 5: Random, simply selects its moves randomly during the course of the game. One can thus play a game of two Agents both Random, by selecting 5 and 5 respectively.

Agent 4: Greedy, also selects its moves randomly, with the exception that if the current game can be won in a single move, Greedy will select that move and win immediately. This Agent is slightly more powerful than its counterpart Random, and will usually win against it if a 4,5 or 5,4 combination is selected.

Finally, selection 2, APS Observer, allows an APS Agent to learn but not actually participate in the game. Therefore the Morph system can learn by observation and need not actually play in the domain it is learning in. If two Human Agents are selected for the domain, an APS Observer (any number) can be selected to observe the two Humans in their contest, and will learn from the results of the game. (Note however, the APS Observer must be selected first prior to selecting other Agents, to allow a repeat prompt of Agents.)

7.7 APS Database Selection

If an APS agent is selected during the Agent prompt for a domain, then a corresponding Database must be loaded for that given Agent for the given domain. The Morph system will prompt as follows:

```
> Enter name of Database for #1  APS :
```

Enter the name of an already existing DB for the domain that was selected. (Note that an APS Agent requires a database for pattern- weight storage representing the accumulated learned knowledge.)

If the Morph system is unable to find the named DB, a new DB will be created for the APS Agent selected with the given DB file name. This database will then represent an APS Agent who has just been born in the given domain, and is just beginning to learn how to play.

Databases for domains in which rules have already been defined also exist for Morph and end with the suffix .db, with the rules file preceding the suffix. The pennies has the DB file pennies.db, and nim, nim.db. One can load these already aged DB's for the game domain defined if desired, or can start new learning DB's from scratch.

7.8 SAMPLE LEARN RESULTS:

```
                HEX PAWN 3x3:  APS 1PLY vs. RANDOM
                ==================================

 # GAMES  Results     Win    Loss   Draw   Total        Tot W  Tot L  Tot D
 -------  -------     ----   ----   -----  ------       -----  -----  -----
  250     112/29/109  44.8%  11.6%  43.6%  112/29/109   44.8%  11.6%  43.6%
  500     133/16/101  53.2%   6.4%  40.4%  245/35/210   49.0%  14.0%  42.0%
  750     124/17/109  49.6%   6.8%  43.6%  369/52/319   49.2%   6.9%  42.5%
 1000     145/24/81   58.0%   9.6%  32.4%  514/76/400   51.4%   7.6%  40.0%
 1250     130/19/101  52.0%   7.6%  40.4%  644/95/501   51.5%   7.6%  40.1%
 1500     117/28/105  46.8%  11.2%  42.0%  761/123/626  50.7%   8.2%  41.7%
 1750     115/21/114  46.0%   8.4%  45.6%  876/144/740  50.1%   8.2%  42.3%
 2000     122/19/109  48.8%   7.6%  43.6%  998/163/849  49.9%   8.2%  42.3%


                HEX PAWN 6x6:  APS 1PLY vs. RANDOM
                ==================================

 # GAMES  Results    Win    Loss   Draw   Total        Tot W  Tot L  Tot D
 -------  -------    ----   ----   -----  ------       -----  -----  -----
  250     199/43/8   79.6%  17.2%  3.2%    199/43/8    79.6%  17.2%   3.2%
  500     216/32/2   86.4%  12.8%  0.8%    415/75/10   83.0%  15.0%   2.0%
  750     218/26/6   87.2%  10.4%  2.4%    633/101/16  84.4%  13.5%   2.1%
  1000    218/26/6   87.2%  10.4%  2.4%    851/127/22  85.1%  12.7%   2.2%
  1250    215/21/14  86.0%   8.4%  5.6%   1066/148/36  85.3%  11.8%   2.9%
  1500    210/34/6   84.0%  13.6%  2.4%   1276/182/42  85.1%  12.1%   2.8%
  1750    215/26/9   86.0%  10.4%  3.6%   1491/208/51  85.2%  11.9%   2.9%
  2000    218/25/7   87.2%  10.0%  2.8%   1709/233/58  85.5%  11.7%   3.5%
```

PENNIES: APS 1PLY vs. RANDOM

# GAMES	Results	Win	Loss	Total	Total W	Total L
250	233/17	93.2%	6.8%	233/17	93.2%	6.8%
500	231/19	92.4%	7.6%	464/36	92.8%	7.2%
750	235/15	94.0%	6.0%	699/51	93.2%	6.8%
1000	249/1	99.6%	0.4%	948/52	94.8%	5.2%
1250	249/1	99.6%	0.4%	1197/53	95.8%	4.2%
1500	250/0	100.0%	0.0%	1447/53	96.5%	3.5%
1750	250/0	100.0%	0.0%	1697/53	97.0%	3.0%
2000	250/0	100.0%	0.0%	1947/53	97.4%	2.7%

NIM: APS 1PLY vs. RANDOM

# GAMES	Results	Win	Loss	Total	Total W	Total L
250	179/71	71.6%	28.4%	179/71	71.6%	28.4%
500	208/42	83.2%	16.8%	387/113	77.4%	22.6%
750	177/73	70.8%	29.2%	564/186	75.2%	24.8%
1000	181/69	72.4%	27.6%	745/255	74.5%	25.5%
1250	173/77	69.2%	30.8%	918/332	73.4%	26.6%
1500	190/60	76.0%	24.0%	1108/392	73.8%	26.2%
1750	188/62	75.2%	24.8%	1296/454	74.0%	26.0%
2000	194/56	77.6%	22.4%	1490/510	74.5%	25.5%

NIM 2: APS 1PLY vs. RANDOM (+1 Dynamic Relation)

# GAMES	Results	Win	Loss	Total	Total W	Total L
250	210/40	84.0%	16.0%	210/40	84.0%	16.0%
500	213/37	85.2%	14.8%	423/77	84.6%	15.4%
750	236/14	94.4%	5.6%	659/91	87.9%	12.1%
1000	233/17	93.2%	6.8%	892/108	89.2%	10.8%
1250	238/12	95.2%	4.8%	1130/120	90.4%	9.6%
1500	232/18	92.8%	3.6%	1362/138	90.8%	9.2%
1750	237/13	94.8%	5.2%	1599/151	91.4%	8.6%
2000	239/11	95.6%	4.4%	1838/162	91.9%	8.1%

8

References

The following is a complete bibliography based on the citations in the previous chapters.

8.1 References

[1] V. Allis. Qubic solved again. In J.V.D. Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3*, pages 192–204. Ellis Horwood, 1992.

[2] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6), 1984.

[3] Lenzerini M. Batini, C. and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *Computing Surveys*, 18(4):323–364, 1986.

[4] E.B. Baum. How a bayesian approaches games like chess. In *Games: Planning and Learning, Proceedings of the AAAI Fall Symposium*, Menlo Park, CA, October 1993. AAAI Press.

[5] P.P. Chen. The entity relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[6] Peter Creasy. Schema integration and model translation. In G. Ellis and R. Levinson, editors, *Proceedings of the Second International Conference on Peirce: A Conceptual Graphs Workbench*, 1993.

[7] Peter Creasy. Conceptual graphs as a canonical data model. 1994. Proceedings of the 2nd International Conference on Conceptual Structures, Baltimore, August 16-20.

[8] Peter Creasy and Gerard Ellis. A conceptual graphs approach to conceptual schema integration. In Guy W. Mineau, Bernard Moulin, and John F. Sowa, editors, *Conceptual Graphs for Knowledge Representation*, number 699 in Lecture Notes in Artificial Intelligence, Berlin, 1993. Springer-Verlag. Proceedings of the 1st International Conference on Conceptual Structures, Quebec City, Canada, August 4-7.

[9] Gerard Ellis. *PEIRCE User Manual*, 1993.

[10] Gerard Ellis. *Managing Complex Objects*. PhD thesis, The University of Queensland, 1994. in preparation.

[11] Gerard Ellis and Robert Levinson, editors. *Proceedings of the First International Workshop on PEIRCE: A Conceptual Graphs Workbench*. Department of Computer Science, The University of Queensland, 1992.

[12] Gerard Ellis and Robert A. Levinson, editors. *Proceedings of the First International Workshop on Peirce: A Conceptual Graphs Workbench*, Las Cruces, New Mexico, July 10 1992. available by anonymous ftp `ftp.cs.uq.oz.au`, `/pub/peirce/peirce92.ps.Z`.

[13] S. Epstein. Deep forks in strategic maps – playing to win. In *Heuristic Programming in Artificial Intelligence 2*, pages 189–203. Ellis Horwood, 1991.

[14] John W. Esch. Linear forms for conceptual structures. In Timothy E. Nagle, Janice A. Nagle, Laurie L. Gerholz, and Peter W. Eklund, editors, *Conceptual Structures: Current Research and Practice*, Ellis Horwood Series in Workshops, pages 595–604. Ellis Horwood, New York, 1992.

[15] B. R. Gaines. Representation, discourse, logic and truth: Situated knowledge technology. In *Conceptual Graphs for Knowledge Representation*, number 699 in Lecture Notes in Computer Science, pages 36–63. Springer-Verlag, 1993.

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[17] Brian J. Garner, Eric Tsui, Dickson Lukose, and Jane Koh. Progress on an extendible graph processor for knowledge acquisition, planning and reasoning. In Tim Nagle, Jan Nagle, Laurie Gerholz, and Peter Eklund, editors, *Conceptual Structures: Current Research and Practice*. Ellis Horwood, 1992.

[18] Michael R. Genesereth and Richard E. Fikes. Knowledge Interchange Format version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, June 1992. "Living document" of the Interlingua Working Group of the DARPA Knowledge Sharing Effort.

[19] J. Gould and R. Levinson. Experience-based adaptive search. In R. Michalski and G. Tecuci, editors, *Machine Learning:A Multi-Strategy Approach*, volume 4, pages 579–604. Morgan Kauffman, 1994.

[20] Tomisha Kamada and Satoru Kawai. A general framework for visualizing abstract objects and relations. *ACM Transactions on Graphics*, 10(1):1–39, Jan 1991.

[21] R. E. Korf. Planning as search. *Artificial Intelligence*, 1987.

[22] Navathe S. Larson, J. and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1984.

[23] R. Levinson and K. Karplus. Graph-isomorphism and experience-based planning. In D. Subramaniam, editor, *Proceedings of Workshop on Knowledge Compilation and Speed-Up Learning*, Amherst, MA., June 1993.

[24] R. Levinson and R. Snyder. Adaptive pattern oriented chess. In *Proceedings of AAAI-91*, pages 601–605. Morgan-Kaufman, 1991.

[25] Robert Levinson and Gerard Ellis, editors. *Proceedings of the 2nd International Workshop on Peirce: A Conceptual Graphs Workbench*, Quebec City, Canada, August 7 1993. available by anonymous ftp `ftp.cs.uq.oz.au`, `/pub/peirce/peirce93.ps.Z`.

[26] Robert A. Levinson. Towards domain-independent machine intelligence. In Guy W. Mineau, Bernard Moulin, and John F. Sowa, editors, *Conceptual Graphs for Knowledge Representation*, number 699 in Lecture Notes in Artificial Intelligence, Berlin, 1993. Springer-Verlag. Proceedings of the 1st International Conference on Conceptual Structures, Quebec City, Canada, August 4-7.

[27] Jiarong Li. Using algebraic constraints in interactive text and graphics editing. In David A. Duce and Pierre Jancene, editors, *Proceedings of the European Computer Graphics Conference and Exhibition*, pages 197–205, Nice, France, September 1988. North-Holland.

[28] John Lyons. *Introduction to Theoretical Linguistics*. Cambridge University Press, Cambridge, 1968.

[29] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1980.

[30] Frances Newbery Paulisch. *The Design of an Extendible Graph Editor*. Number 704 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York. ISBN 0-387-57090-X.

[31] Barney Pell. METAGAME: A new challenge for games and learning. In H. J. van den Herik and L. V. Allis, editors, *Programming in Artificial Intellegence: The Third Computer Olympiad*. Ellis Horwood, 1992.

[32] Sandra Perez and Anthony Sarris (eds). IRDS conceptual schema. Technical Report X3H4/93-196, American National Standards Institute, New York, 1993.

[33] Luis A. Pineda. A compositional semantics for graphics. In David A. Duce and Pierre Jancene, editors, *Proceedings of the European Computer Graphics Conference and Exhibition*, pages 155–169, Nice, France, September 1988. North-Holland.

[34] George G. Roberston, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3d interactive animation. *Communications of the ACM*, 36(4):57–71, Apr 1993.

[35] D.D. Roberts. The existential graphs. In *Semantic Networks in Artificial Intelligence*, pages 639–664. Roberts, 1992.

[36] Castellanos M. Saltor, F. and M. Garc!a-Solaco. Suitability of data models as canonical models for federated databases. *ACM SIGMOD Record*, 20(4), 1991.

[37] John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, MA, 1984.

[38] John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. The System Programming Series. Addison-Wesley Publishing Company, 1984.

[39] C. Spaccapietra, S. Parent and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, 1992.

[40] S. Spaccapietra and C. Parent. View integration: a step forward in solving structural conflicts. *IEEE Transactions on Data and Knowledge Engineering*, 1993.

[41] Shin Takahashi, Satoshi Matsuoka, Akinori Yonezawa, and Tomisha Kamada. A general framework for bi-directional translation between abstract and pictorial data. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 165–174, Hilton Head, South Carolina, USA, Nov 1991.

[42] G. Verheijn and J. van Bekkum. NIAM: An information analysis method. In Olle *et al.*, editor, *Information Systems Design Methodologies: A Comparative Review*. North Holland, Amsterdam, 1982.

[43] John M. Vlissides. Generalized graphical object editing. Technical Report CSL-TR-90-427, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA, June 1990.

[44] John M. Vlissides and Mark A. Linton. Unidraw: A framework for building domain-specific graphical editors. *ACM Transactions on Information Systems*, 8(3):237–268, Jul 1990.

[45] Michel Wermelinger. Searching efficiently in posets. Technical report, Centro de Informática da Universidade Nova de Lisboa, March 1993.