A Simplified Data Model for CGs

Peter Becker and Joachim Hereth

September 14, 2001

Abstract

This paper tries to model a simplified data model for CGs that can be used as base for different notations and applications. Based on the draft for an ISO standard the proposed model splits CGs into a core with only a simple structure and gives a number of extensions to introduce additional features in a consistent way.

1 Introduction

This paper proposes a data model for simple Conceptual Graphs (positive Conceptual Graphs with untyped nesting and without generic markers, according to the classification by [CM97]). It is based upon the current[1] draft for an ISO standard for Conceptual Graphs edited by John Sowa [So01] and discussions with various people involved in CGs.

The main purpose of this approach is to give a structure for treating Conceptual Graphs in programs and notations. We distinguish between three aspects for modelling Conceptual Graphs: a data model, a set of operations and the syntactical notations. In this paper we will use the term "syntax" only for specific notations, while the term "semantic" refers to the algebra defined by the data model and the operations upon it. This view is taken since we consider CGs themself as the domain we want to model, as opposed to many other papers which consider CGs as syntax and try to model their semantics.

The data model described in section 2 models only the parts needed for simple CGs, many aspects described in the standard draft are deliberately skipped since they are considered as extensions and this model should only be a base to work with.

In section 3 we discuss possible extensions of the data model given, some of them can be used to get the expressibility of the current Conceptual Graphs, some of them go beyond the usual Conceptual Graphs. Section 5 gives a summary and discussion of the paper.

[1]Date of writing: September 01, 2001

2 Data Model

A data model is the core of every application in an IT system. Often this important aspect is not modelled explicitely and discussions showed that the CG community has no coherent definition of this data model. The data model given in this section is meant to build a referable base model for CGs, at least for purposes of discussion.

The goal of a data model is to be well-defined but understandable. Although the data modelled here is usually seen in the syntactical form of the CG Display Form (DF) it is not meant to model this syntax but the data represented by it. In many cases this will be more complex than the usual representations in DF but if a simpler representation is possible for some data a specific syntax can always allow shortcuts like declaring parts of the alphabet implicitly. This kind of information has still to be modelled for purposes of implementation in IT systems and reasoning.

2.1 Catalog

The *catalog* defines in this data model the basic tokens used in the graphs themself. The term "catalog" is used here not only for the tokens themself but also for the additional structures defined on them, i.e. the type hierarchies.

Comment: The idea of modelling this as CGs themself is surely interesting but we decided to skip it due to some problems in this approach. Since the data model doesn't describe how exactly the information is stored or written an implementation or notation can still use this approach.

2.1.1 Type

The *types* are a set of basic objects on which a partial order is defined by a *subtype* relation which has a greatest element called *universal type* denoted with $\top$.

Comment: We dropped the bottom element since it does not seem important for either implementations or notations.

2.1.2 Relation

The *relations* are a set of basic objects. Each relation has a *signature*. The signature of a relation is defined by a vector of n types, n is called the *valence* of the relation.

For each valence there exists a partial order defined by a *subtype* relation with a greatest element $\top_n$ that has the signature with a list of n occurences of $\top$. If a relation is a subtype of another its signature has to be a subtype of the other in each element.

Comment: This is what describes mathematically the relations themself, not a type of relation, while the vertices in a graph only model one element in the relation. Or to be more exact: the relation is modelled intensionally by this structure and extensionally by all links which belong to it (see Sec. 2.2.2).

To ensure well-defined subtyping we introduced multiple top elements for relations, one for each valence. This way we can model subtypes of relations while ensuring correct signatures.

2.1.3 Instances

The third kind of core data are the *instances*. These don't have any hierarchies, they are just a simple set having unique markers as indices.

Comment: In this model all instances have to be refered to by their markers, there is no way to talk about anything specific without declaring an instance, although this can be done implicitly in a syntax.

2.2 Conceptual Graph

A *conceptual graph* is modelled as bipartite graph from *nodes* and *links* where the links are explicitely modelled as vertices in the data model. The arcs of such graph are modelled through references from a link to a node.

2.2.1 Node

A *node* has a *type*, a *referent* pointing to an instance and it can have a conceptual graph used as *descriptor*.

If a descriptor is given the graph and all nodes and links directly in it are called to be *immediately nested* in the node containing the descriptor. Any node or link c is called *nested* in another node d if it is immediately nested in d or nested in some node immediately nested in d.

Two nodes or links c_1 and c_2 are said to be *co-nested* if they are either the same or immediately nested in the same node d. Any node or link c nested in another node d is said to be *more deeply nested* than any node or link co-nested with d.

A node or link d is said to be *within the scope* of a node or link c if it is co-nested with c or more deeply nested than c.

Comment: This has been highly simplified, excluding a number of features we considered only syntactical aspects and some others that can be modelled as extentions (see section 3).

Some of the main differences are: only the existential quantifier is allowed (the universal is available when negation is added, collections are considered as extension), no literals (might be replaced by representations), only one locator (the other ones don't map into the data model).

The term "node" has been choosen since it matches the typical usage in semantic networks and its interpretation in a CG depends on the quantifier used – used with the existential quantifier it refers more to a specific individual than a concept.

2.2.2 Link

A *link* has a relation as its *type* and has a vector of n *references* to nodes where n is the valence defined for its relation. Each node refered has to be either of the same type or a subtype of the type given in the same position in the signature for its relation and the link has to be in the scope of the node.

Comment: The ability to reference nodes outside the scope of the link and not only those co-nested with it removes the need for coreference sets.

To map this model into coreference sets one could replace the reference to a node with a node of type $\top$ without referent and a coreference link to the node refered.

3 Data Model Extensions

This section describes a number of extensions that can be made to make the basic data model more expressive. Some of them are already covered at least partly in [So01], some are already implemented in tools and others are just ideas that were raised in discussions.

3.1 Representations

For any instance a number of mappings into a set of *representations* can be given. These representations are some kind of encodings (e.g. Unicode strings as names, numbers, binary data) that can be processed by specific applications.

To ensure interoperability between different applications a common set of encodings should be defined and there should be some way to extend this set in a way that any application can decide if it can present a specific encoding or not.

One option for defining a core set of encodings would be reusing the datatypes defined for XML Schema (see [XMLS-DT]). Additional encodings could be defined by reusing the MIME-types ([MIME]).

3.2 Generic Marker

In addition to the individuals a *generic marker* can be allowed which denotes the existence of some individual of the given type without defining the exact one.

3.3 Negation

Negation can be done by introducing a new construct for negating a graph. In the description of CGIF this is describe as a “concept of type Proposition with an attached relation of type Neg” but in our view it is a completely new construct and should be modelled as such.

This means introducing a construct *negated graph* which can occur in any conceptual graph and is a conceptual graph itself. References across the border of this graph are allowed as for the descriptors of nodes, this way cuts can be modelled as presented in [Da00] which is equivalent to the way John Sowa proposes the modelling using co-references, a display form which can be achieved by showing additional nodes with type $\top$.

3.4 Collections

To allow a smaller data model for repetitive graphs the referent of a node could be extended to contain a set of instances instead of a single one which can be read as having a number of graphs where each one establishes the same link for each of the individuals in the set.

3.5 Additional Quantifiers

When negation has been introduced the universal quantifier can be modelled easily using the existential one in combination with negation. In addition to these options one might be interested in modelling collection sizes explicitely, e.g. to state something like "there are at least three different people wearing the same type of shirt". As opposed to the collections described above the exact instances don't have to be known. Minimum and maximum size could be given. This would map into simple graphs in a similar way as the defined collections where a number of implicit distinct instances will be created.

3.6 Lambda Expressions

Lambda expressions can be modelled by adding *formal parameters* into the conceptual graphs. They could be interpreted in the same manner as formal parameters in a declaration in a programming language, defining a signature with a number and the type of concept instances accepted to instantiate this lambda expression.

Additionally there has to be some way to add instances of lambda expressions into a conceptual graph giving the *actual parameters* needed for instantiation. The types and relations have to allow specifying monadic lambda expressions instead of basic objects.

Since lambda expressions are just graphs with formal parameters we don't think they should be modelled as a separate construct since there doesn't seem to be a need for it. Every graph can be read as n-adic lambda expression where n is the number of formal parameters given for it.

3.7 Cross-Document Linking

For many purposes it would be useful to be able to link documents together in different ways. For example, one could use an alphabet defined in a central file to discuss graphs given elsewhere with it.

To accomplish this one has to introduce the notion of a document and to give some way of giving cross-document references in the graphs and for the catalog parts. In some way this can be seen as specialisation of the extisting structures that accomodate with the cross-document referencing issues – nothing new will be introduces but the existing model has to be more specific to be able to handle e.g. concept references in a consistent way across documents.

One of the most central issues in this is having globally unique identifiers. While the current model in a form for one document can use e.g. simple numbers for identifying types, relations and individuals this is not sufficient to ensure that the modelled information can be refered to by an external document.

These problems already have been addressed in the XML field, a technique like XLink ([XLink]) can be used for establishing the links while the concept of namespaces ([XML-NS]) can be used to avoid conflicts.

3.8 Actors

Another extension to introduce new expressivity is adding something for implementing functionality like calculations ([So98]). This is done using actors but there is no common model for this yet. In addition to the actors themselves this needs some way to model variables that can be changed by them.

This extension is rather complex and will not be discussed here.

3.9 Rules

While actors change variables in a graph many applications might want to change the graphs themself. This can be done by using colored or marked graphs like those presented by Baget ([Ba00]) or implemented in the Ossa system. This is another extension not discussed here.

3.10 Reifying types and relations

The model presented here does not allow reasoning on the types or the relation themself. This might be useful sometimes but needs some kind of extension, too. This could be done using a specific type of nodes in a conceptual graph for them.

Another part of CGs that can't be used in CGs with this model are the links themself. If one drops the idea of the bipartite graph and allows links to refer to links this could be achieved, too. The hierarchies could get a common top element to model this more easily (a supertype of $\top$ and all $\top_n$ nodes).

4 Operations

Given this data model operations on it can be defined to give an algebra for using CGs. This algebra will define all syntactical operations that can be done on CGs, the term "syntactical" refering here to CGs as syntax for knowledge representation.

There has some work be done to model these operations in a formal mathematical way ([Pr00], [Mi00]), we expect that this work can be easily reused for defining the operations on the given data model. The data model presented here gives more detail and is closer to implementations and notations but should have a direct mapping to the mathematical structures used in the existing work.

5 Summary

The data model presented in this paper can be used as a base model for developing CG applications and notations. Since this model is deliberately kept simple and left many of the historic burdens of existing formats and applications aside it should be a good base for discussions and further work. One of the most important approaches is to add an algebra to it and to show that thereby one gets a semantic that is useful for CGs. In addition at least some of the extensions should be modelled more explicitely.

The authors hope to give useful input with this into the process of creating a standard for CGs and wish to thank everyone who discussed the data model with us, most noticeably Finnegan Southey and Murray Altheim.

References

[Ba00] J.-F. Baget: Extending the CG Model by Simulations, in: B. Ganter, G.W. Mineau (Eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues*, LNAI 1867, Springer-Verlag, Berlin Heidelberg 2000, pp. 277–291

[CM97] M. Chein, M.-L. Mugnier: Positive Nested Conceptual Graphs, in: D. Lukose et al. (Eds.): *Conceptual Structures: Fulfilling Peirce's Dream*, LNAI 1257, Springer Verlag, Berlin-New York 1997, pp. 95–109.

[Da00] F. Dau: Negations in Simple Concept Graphs, in: B. Ganter, G.W. Mineau (Eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues*, LNAI 1867, Springer-Verlag, Berlin Heidelberg 2000, pp. 263–276

[Mi00] G. Mineau: The Extensional Semantics of the Conceptual Graph Formalism, in: B. Ganter, G.W. Mineau (Eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues*, LNAI 1867, Springer-Verlag, Berlin Heidelberg 2000, pp. 221–234

[MIME] E. Hood: *Multipurpose Internet Mail Extensions (MIME)*, Summary Website, http://www.nacs.uci.edu/indiv/ehood/MIME/MIME.html

[Pr00] S. Prediger: Nested Concept Graphs and Triadic Power Context Families, in: B. Ganter, G.W. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues*, LNAI 1867, Springer-Verlag, Berlin Heidelberg, pp. 249–262.

[So98] J. Sowa: Conceptual Graph Extensions, in: M.-L. Mugnier, M. Chein (Eds.): *Conceptual Structures: Theory, Tools and Applications*, LNAI 1453, Springer-Verlag, Berlin Heidelberg, pp. 3–14

[So01] J. Sowa: *Conceptual Graphs*, Draft for an ISO International Standard, http://www.bestweb.net/ sowa/cg/cgstand.htm

[XLink] W3C Recommendation: *XML Linking Language (XLink), Version 1.0*, http://www.w3.org/TR/2001/REC-xlink-20010627

[XMLS-DT] W3C Recommendation: *XML Schema Part 2: Datatypes, Version 1.0*, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502

[XML-NS] W3C Recommendation: *Namespaces in XML, Version 1.0*, http://www.w3.org/TR/1999/REC-xml-names-19990114