UNDERSTANDING THE SEMANTICS OF CONCEPTUAL GRAPHS

MARKO NIINIMÄKI

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1999-4

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1999-4, MARCH 1999

UNDERSTANDING THE SEMANTICS OF CONCEPTUAL GRAPHS

MARKO NIINIMÄKI

University of Tampere
Department of Computer Science
P.O.Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4539-2
ISSN 0783-6910

Understanding the semantics of conceptual graphs

Marko Niinimäki
man@sjoki.uta.fi
University of Tampere
Department of Computer Science

March 3, 1999

Contents

Abstract

In this paper, we propose a simple and well established semantics for conceptual graphs. This model theoretic semantics is an alternative to previous approaches, where a game theoretical semantics has been suggested. In addition to that, we thoroughly compare conceptual graphs with First Order Predicate Logic (FOPL). The comparison is based on semantics. Based on this comparison, we present some clarifying remarks on the semantics of conceptual graphs.

The comparison of the expressive power of FOPL and conceptual graphs is carried out by a consideration of translation algorithms. John Sowa's algorithm translates an arbitrary conceptual graph into a FOPL formula. The algorithm that we shall outline in this paper translates an arbitrary closed FOPL formula into a conceptual graph. On the basis of the algorithms, it can be claimed that the expressive power of conceptual graphs (with limited syntax) equals to that of a slightly limited FOPL. Without the syntactic limitations, the expressive power of conceptual graphs would probably exceed the expressive power of FOPL.

We also provide a set of knowledge representation examples using both FOPL and conceptual graphs. The examples indicate that in some cases conceptual graph formalism is harder to read than FOPL.

1 Introduction

Conceptual graphs have been proposed as an alternative to First Order Predicate Logic (FOPL) in knowledge representation ([6], see also [1]). FOPL and its derivatives (subsets), like Horn -clauses and Prolog, are perhaps the best known and established formalisms for knowledge representation. To gain acceptance, the alternative formalisms should offer advantages over FOPL. The advantages can be

- simplicity: formulas expressed by an alternate formalism may be easier to read and to write (or closer to natural language) than formulas of FOPL and its derivatives,
- deductive power: FOPL is known to be semi-decidable. More limited formalisms can have a proof theory that is decidable, and even tractable,
- expressional power: with traditional FOPL, uncertainty or default reasoning, for example, cannot be handled.

In order to see if conceptual graphs are more simple, have more deductive power or have more expressional power than FOPL, we shall carry out an extensive comparison between FOPL and conceptual graphs in sections 3 and 4.

To make the comparison more easy, we use simplified forms of both FOPL and conceptual graphs. This is explained in detail in section 2.1 and 3.1. Section 2.1 contains a standard presentation of a slightly limited FOPL and section 2.2 its semantics. We shall use the term "FOPL" to refer to this limited version. The same kind of presentation of conceptual graphs (limited syntax) and their semantics will follow sections 3.1 and 3.2.

The basis for the comparison is the following simple notion: If any expression of a language L_1 can be translated into an expression of a language L_2 and any expression of a language L_2 can be translated into an expression of a language L_1, then the languages have equal expressive power. The translation is justified by the semantics of the languages L_1 and L_2.

Sowa ([6]) has presented an algorithm that translates a conceptual graph into a FOPL formula. Sowa states, however, that the expressional power of conceptual graphs may exceed the expressional power of FOPL: "When the referents of concepts are limited to single individuals, conceptual graphs cannot go beyond first-order logic." ([6], p. 116). Later Sowa extends the notation so that a referent of a concept can be a set of individuals. Sowa's algorithm obviously applies only to conceptual graphs where the referents of concepts are single individuals. This is the limited syntax of conceptual graphs defined in section 3.1.

Assuming that Sowa's algorithm translates an arbitrary conceptual graph into a formula of FOPL, it follows that FOPL is at least as expressive than conceptual graphs. In section 4 we outline an algorithm for translating closed formulas of FOPL into conceptual graphs. On the basis of these two algorithms it can be claimed that conceptual graphs (with limited syntax) and a slightly limited FOPL equal in expressive power.

2 First Order Predicate Logic

2.1 The language L_{FOPL}

A language is a set of valid expressions (in the case of logic, well-formed formulas) composed from a set of terminal symbols by rules of a grammar. The set

T_{FOPL} contains the terminal symbols of the language L_{FOPL}. Elements of T_{FOPL} are constants $(c_0, c_1, c_2, ..)$, variables $(x_0, x_1, x_2, ..)$, connectives $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$, quantifiers $(\exists, \forall)$, parenthesis, comma and unary and binary predicate symbols $(P_0, P_1, P_2, .., R_0, R_1, R_2, ..)$.[1]

In a more formal approach, predicates can have any arity and the information of a predicate's arity must be given. In this paper we use only unary and binary predicates. *P*-predicates are unary and *R*-predicates binary.

In metalanguage, i.e. describing expressions of L_{FOPL}, we use symbols x, y, z to denote variables, a, b, c to denote constants, A, B, C to denote formulas, and H when a predicate (no matter if its arity is 1 or 2) is discussed. In an informal approach (like in section 3.1.2), we also use lowercase natural language words instead of predicate symbols.

In this paper, T_{FOPL} does not include the identity symbol $=$ and there are no functions in the language.

Variables and constants are jointly called TERMS. Terms are written after the predicate symbol, between the parenthesis. The number of these terms is determined by the arity of the predicate. These terms are called the ARGUMENTS. A predicate symbol, followed by its arguments (in parenthesis) is called an ATOMIC FORMULA. An atomic formula is called a GROUND FORMULA, if all the arguments are constants.

A quantifier BINDS the variable x in formulas $\exists x(A)$ and $\forall x(A)$. A variable in formula A is bound if it is bound by a quantifier. Otherwise it is free. If there are no free variables in a formula, then the formula is called a CLOSED FORMULA. A ground formula is an example of a closed formula.

A REPLACEMENT of a variable x with a variable y or a constant c is denoted with *A(x/y)* and *A(x/c)*, respectively.

If replacements are performed in a formula and the result of these replacements is a formula where there are no two quantifiers that both bind a variable x, then the replacements are called UNAMBIGUOUS NAMING of variables.

Gr_{FOPL} is a grammar that generates all formulas of L_{FOPL}. The terminal symbols of the grammar are elements of T_{FOPL} and the symbol "|" in the production rules means "or". The production rules of the grammar are:

[1] The inverse binary predicate symbols (see section 2.3) can be considered as a special case of binary predicate symbols.

1. $X \mapsto x_0|x_1|x_2|..$
2. $C \mapsto c_0|c_1|c_2|..$
3. $U \mapsto X|C$ (constants and variables are terms)
4. $P \mapsto P_0|P_1|P_2|..$ (unary predicates)
5. $R \mapsto R_0|R_1|R_2|..$ (binary predicates)
6. $F \mapsto P(U)|R(U,U)$ (atomic formulas are formulas)
7. $F \mapsto \neg(F)$ (a negation of a formula)
8. $F \mapsto (F \wedge F)$ (two formulas connected with $\wedge$)
9. $F \mapsto (F \vee F)$ (two formulas connected with $\vee$)
10. $F \mapsto (F \rightarrow F)$ (two formulas connected with $\rightarrow$)
11. $F \mapsto (F \leftrightarrow F)$ (two formulas connected with $\leftrightarrow$)
12. $F \mapsto \exists X(F)$
13. $F \mapsto \forall X(F)$ (a quantifier and a variable and a formula together form a formula.)

It is worth noticing that the grammar Gr_{FOPL} allows formulas with bound variables (like $\exists x_0(P_0(x_0))$ and $\exists x_0(\exists x_1(R_0(x_0,x_1))))$, and formulas with free variables: $\exists x_2(\exists x_1(\exists x_0(P_0(x_3))))$.

It can be easily observed that L_{FOPL} is context free, since all the rules of the grammar are context free.[2]

2.2 Semantics of L_{FOPL} (model theory)

In this section we shall give a short account on semantics of FOPL. We shall discuss about model theory, satisfiability (truth in some L-model) and validity (truth in all L-models). The formal definition of L-model is presented after these notions.

[2]For details on formal languages and grammars see e.g. [4].

A formula is said to be SATISFIABLE, if it can be true in some situation, i.e. in some L-model. For instance, the formula $\exists x_0(P_0(x_0))$ is satisfiable. If the predicate P_0 is interpreted "to be an even natural number", the formula is satisfied when x is 2,4,6.. In this case, the set $\{2,4,6\}$ is called the extension of the predicate P_0, too.

Similarly, a set of formulas is satisfiable (or consistent), if there is a L-model, where all the formulas of the set are (simultaneously) true.

It is evident that the formula $\exists x_0((P_0(x_0) \wedge \neg(P_0(x_0))))$ cannot be true in any L-model. This kind of formula is UNSATISFIABLE. A set of formulas is unsatisfiable (or inconsistent), if there is no L-model, where all the formulas of the set can be (simultaneously) true.

A formula is VALID if it is true in all L-models. $\neg(\exists x_0((P_0(x_0) \wedge \neg(P_0(x_0)))))$ is an example of a valid formula.

The formal definitions of an L-model and truth in L-model are as follows:

An L-model is a pair $< UoD, v >$, where UoD is a non-empty set and v is an interpretation function. An interpretation function is a function that interprets ("maps") terminal symbols of the language into elements of UoD.

In this paper we employ the idea of an extended language (see [2]). This means that all individuals of the UoD are named with some constant. In other words, instead of L_{FOPL} the language $L_{FOPL'}$ is used. $L_{FOPL'}$ is extended from L_{FOPL} by adding a constant c_u for every individual $u \in UoD$.

Let v be an interpretation function of the language L_{FOPL} and v' an interpretation function of the language $L_{FOPL'}$. v' is equal to v except that $v'(c_u) = c_u$ for all $u \in$ UoD.

Let M be a L-model. The following set of conditions, TC_{FOPL} defines the truth value of a formula in L-model M:

1. Let t be a term. If the formula is of form $P(t)$, it is true in M iff t is interpreted to be an element of UoD and t is a member of the extension of the unary predicate P.

2. Let t and u be terms. If the formula is of form $R(t,u)$, it is true in M iff t and u are interpreted to be elements of UoD and (t,u) is a member of the extension of the binary predicate R.

3. If the formula is of form $\neg(B)$, it is true in M iff B is not true in M.

4. If the formula is of form $(B \wedge C)$, it is true in M iff B and C are true in M.

5. If the formula is of form $(B \wedge C)$, it is true in M iff both B and C are true in M.

6. If the formula is of form $(B \vee C)$, it is true in M iff either B or C or both are true in M.

7. If the formula is of form $(B \rightarrow C)$, it is true in M iff it is NOT the case that B is true in M, and C is false in M.

8. If the formula is of form $(B \leftrightarrow C)$, it is true in M iff either

   - both it B and C are true in M.
     or
   - both it B and C are false in M.

9. If the formula is of form $\exists x(B)$, it is true in M iff there is an interpretation, in which *B(x/a)* is true for some $a \in UoD$.

10. If the formula is of form $\forall x(B)$, it is true in M iff there is an interpretation, in which *B(x/a)* is true for all $a \in UoD$.

If a formula A is (inevitably) true in an L-model when formulas of a set S are true in that L-model too, then A is a LOGICAL CONSEQUENCE of the formulas of S. This is denoted by $S \models A$.

In the above case, if S in an empty set, then A is a valid formula. This is denoted by $\models A$.

If formulas A and B are true in the same L-models, then they are LOGICALLY EQUIVALENT. This is denoted by $A \Leftrightarrow B$.

The following "shortcut rules" S1 - S16 are examples of logical equivalences.

S1. $(A \vee B) \Leftrightarrow \neg((\neg(A) \wedge \neg(B)))$,

S2. $\neg(\neg(A)) \Leftrightarrow A$,

S3. $((A \wedge B) \vee C) \Leftrightarrow ((A \vee C) \wedge (B \vee C))$, S4. $\neg((A \wedge B)) \Leftrightarrow (\neg(A) \vee \neg(B))$,

S5. $\neg((A \vee B)) \Leftrightarrow (\neg(A) \wedge \neg(B))$,

S6. $(A \rightarrow B) \Leftrightarrow \neg((A \wedge \neg(B)))$,

S7. $(A \rightarrow B) \Leftrightarrow (\neg(A) \vee B)$,

S8. $(A \leftrightarrow B) \Leftrightarrow (\neg((A \wedge \neg(B))) \wedge \neg((B \wedge \neg(A))))$,

S9. $(A \leftrightarrow B) \Leftrightarrow ((A \rightarrow B) \wedge (A \rightarrow B))$,

S10. $\forall x(H(x..)) \Leftrightarrow \neg(\exists x(\neg(H(x..))))$,
where H is a predicate symbol and $x..$ stands for its arguments.

S11. $\neg(\forall x(H(x..))) \Leftrightarrow \exists x(\neg(H(x..)))$,

S12. $\neg(\exists x(H(x..))) \Leftrightarrow \forall x(\neg(H(x..)))$,

S13. $((A \wedge B) \wedge C) \Leftrightarrow ((A \wedge (B \wedge C)))$,

S14. $((A \vee B) \vee C) \Leftrightarrow ((A \vee (B \vee C)))$,

S15. $(A \wedge B) \Leftrightarrow (B \wedge A)$,

S16. $(A \vee B) \Leftrightarrow (B \vee A)$.

These shortcut rules will be utilised when FOPL formulas are transformed into a form that is easy to process and understand. This form will be used when FOPL formulas are compared with conceptual graphs.

In this paper, we do not present proof theory for FOPL. Instead, we state the following short remarks on semantics and proof theory.

A proof theory for any language consists of a set of axioms and a set of deduction rules. Let A be a formula and S a set of formulas. If a A can be deduced from the formulas S, then A is called a theorem. This is denoted by $S \vdash A$. Suppose there is a proof theory PT_{FOPL}. Language L_{FOPL} with semantics TC_{FOPL} and the proof theory PT_{FOPL} are jointly called an interpreted system of logic. In this system, if every theorem is a valid formula, then the system is SOUND. If the system is sound and every valid formula is also a theorem then the system is COMPLETE. It is possible to construct a proof theory so that the system is complete (see e.g. [5]). Thus, all the semantic shortcut rules are syntactic, too. For instance, since it is known that the shortcut $(A \vee B)$ is logically equivalent to $(B \vee A)$, we can also deduce $(B \vee A)$ from the set $\{(A \vee B)\}$.

2.3 Quantifiers and inverse relations

To support the discussion in sections 2.4 and 3.1.6, we present a short notion on the relationship of quantifiers and relations.

The inverse relation R^{-1} of R consists of the pairs (y,x) for each pair $(x,y) \in R$. Let R be a binary predicate. In this paper, the name "inverse predicate" will be used for R^{-1}.

The following notion relates the inverse relations with quantifier sequences:

$$Qy(Qx(R(x,y))) = Qy(Qx(R^{-1}(y,x))),$$

where Q's are (possibly different) quantifiers.

To compare FOPL formulas with conceptual graphs, it is profitable to arrange the variables so that they appear in the same order in the relations as in the quantifier sequence. For binary predicates, this is easy, based on the previous notion; for example, $\exists y(\forall x(R(x,y))) = \exists y(\forall x(R^{-1}(y,x)))$.

2.4 The translation form

Any FOPL formula can be transformed into a TRANSLATION FORM by using the shortcut rules S1 - S16. The translation form makes FOPL formulas more uniform and thus easier to process. A closed formula in a translation form can, in principle, be translated into a conceptual graph (see section 4 and the appendix).

An algorithm that transforms an FOPL formula A into a translation form formula $C(A)$ is as follows:

REPEAT

1. Use shortcut rule S9 to replace equivalences with implications. Use shortcut rule S7 to replace implications $(B \to C)$ with $(\neg(B) \vee C)$.
2. If there are binary predicates in the formula, use commutativity- and associativity shortcuts (S13 - S16) so that the binary predicates appear as early (left hand side) as possible in the formula.

3. Limit the scope of the $\neg$ -connective by applying the following shortcuts until none of them can be applied.
   - shortcut S2: replace $\neg(\neg(B))$ with B,
   - shortcut S5: replace $\neg((B \vee C))$ with $(\neg(B) \wedge \neg(C))$,
   - shortcut S4: replace $\neg((B \wedge C))$ with $(\neg(B) \vee \neg(C))$.
4. Perform an unambiguous naming of the variables.
5. Move all the quantifier - variable pairs to the left hand side of the formula.
6. Using the notion of section 2.3, transform the binary predicates into their inverse predicates if necessary. That is, maintain the same order of variables in binary predicates as in the quantifier sequence.

UNTIL none of the rules above can be applied.

3 Conceptual graphs

Conceptual graphs are a general, large formalism for knowledge representation, developed by John F. Sowa ([6]). The term "general" means that the formalism is not designed to e.g. physical modeling, but can serve as a same kind of paradigm as FOPL. Sowa mentions that so-called existential graphs by Peirce (see [3]) have served as an archetype for conceptual graphs.

Sowa's system of conceptual graphs includes mechanisms to implement deductions using a concept hierarchy. In this paper, we omit features of the hierarchy.

In addition to this, our discussion of conceptual graphs differs form Sowa's presentation as follows:

1. In Sowa's account, concept nodes can refer to a set of referents (constants or variables, see [6], p. 116). Definitions of concept node and referent appear on page 11 in this paper.
2. Sowa employs relations, whose arity is greater then 2. In this paper, the arity is limited to 2.

3. Sowa presents various "shortcuts" in the language for proper names, measures and quantities.

4. Sowa uses concept nodes that have no referent. However, this is a shortcut for the representation of general concept (see below).

5. Equally, Sowa uses the universal quantifier without an explicit variable. In this paper, the variables are explicit.

6. Sowa uses a formalism "#" for a specified referent. This is replaced with a constant in this paper.

7. The connective $\wedge$ does not appear in Sowa's (or Peirce's, [3]) accounts, but graphs are written next to each other to represent conjunction. In this paper, the $\wedge$ -connective is explicit.

8. Sowa represents most of the connectives and the universal quantifier as shortcuts. Here they are included in the terminal symbols of the language.

9. Sowa suggests that the set of relation nodes consists solely of symbols corresponding to "cases" of case grammars. In this paper the set of relation symbols is a matter of choice.

10. Sowa (and Peirce) use a line of identity to denote the same referent in two or more concepts. In this paper, referents with the same variable or constant "automatically" refer to same individuals. The same convention is used in logic programming.

It is evident that the differences 3 - 10 are only notational and have no effect to the expressional power of the language. The two first ones, however, limit the expressional power to a great extend. Therefore, in this paper the version of the conceptual graphs that we use will be called conceptual graphs with limited syntax. This version is a subset of Sowa's original conceptual graph language.

3.1 The language of conceptual graphs (limited syntax), L_{CG}

3.1.1 Introduction

Like in FOPL, normal connectives are employed into the set of terminal symbols of conceptual graphs (limited syntax): $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

The existential quantifier is not needed in the language, since all quantifiers all assumed to be existentially quantified, unless they are universally quantified. Thus, only the universal quantifier ($\forall$) is in the set of terminal symbols.

Infinite sets of variables $x_0, x_1, x_2, ..$ and constants $c_0, c_1, c_2, ..$ will be included, too. In metalanguage, variables are refered by *x, y* and *z* and constants by *a, b* and *c*.

Instead of unary and binary predicates, in conceptual graphs (limited syntax) there are CONCEPT NODES and RELATION NODES. A concept node is constructed of a left bracket ([), a concept type symbol ($P_0, P_1, P_2, ..$), a colon (:) a referent (a constant or a variable) and a right bracket (]). In metalanguage, P and Q stand for concept type symbols and informally uppercase natural language words are used instead of predicate symbols. A concept node $[P : x]$ is interpreted the same way as the expression $\exists x(P(x))$ in FOPL. A concept node may include a universal quantifier, too. A special predicate symbol $\top_c$ is included in terminal symbols. This is discussed in section 3.1.2.

Relation nodes connect concept nodes to each other in a conceptual graph. A relation node consists of an arrow ($\leftarrowtail$ or $\rightarrowtail$) that is connected with a concept node, a relation symbol ($R_0, R_1, R_2..$) in parenthesis and another arrow ($\leftarrowtail$ or $\rightarrowtail$) that is connected with another concept node.[3] Informally, uppercase natural language words stand for relation symbols, too.

A concept node alone is a conceptual graph, but a relation node must be connected to two[4] concept nodes. A simple example of a conceptual graph with a relation node is:

$$[P : x] \rightarrowtail (R) \rightarrowtail [Q : y].$$

Intuitively, a conceptual graph (or a graph for short) is

- a single concept node or
- two concept nodes, connected with a relation node or
- a construction, where two or more conceptual graphs are joined by connectives.

The scopes of quantifiers in conceptual graphs are discussed in section 3.1.4.

[3] The question whether to use left or right directed arrows is related with the semantics of the relation. This will be discussed in section 3.1.6.

[4] This is since in this paper no other arities than 2 are allowed for relations.

3.1.2 Simple examples

A concept node, whose referent is a variable is called a generic concept. A concept node, whose referent is a constant is called an instantiated concept.

(1) $$[PAINTING : x] \quad [PAINTING : a]$$

A generic concept and an instantiated concept

or more formally

(2) $$[P_0 : x_0] \quad [P_0 : c_0]$$

The relationship "owns" (i.e. "x owns y") is presented by:

$$\rightarrowtail (OWN) \rightarrowtail,$$

where the left $\rightarrowtail$ connects the relation node to a concept node, whose referent is x and the right $\rightarrowtail$ connects the relation node to a concept node, whose referent is y.

The inverse relation "is owned by" can be presented simply by reversing the direction of the arrows, that is:

$$\leftarrowtail (OWN) \leftarrowtail .$$

In FOPL, $\exists x(\exists y(R(x,y)))$ is a well-formed closed formula. In conceptual graphs, it seems that the types of x and y should be represented by a concept node. For this purpose, the symbol $\top_c$ was introduced in the set of terminal symbols. $\top_c$ is needed to represent an unknown type. The conceptual graph that conveys the information "there is some x that owns some y" is

(3) $$[\top_c : x] \rightarrowtail (OWN) \rightarrowtail [\top_c : y].$$

Using connectives, one can construct conceptual graphs that have CONTEXTS like in graphs 4 and 5.

$$[[ARTIST : a] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : b] \\ \wedge \\ [COLLECTOR : c] \rightarrowtail (BUY) \rightarrowtail [PAINTING : b]].$$

(4)

Graph 4, that conveys the information "A certain artist paints a certain painting that is bought by a certain collector".

$$[[ARTIST : a] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : b] \\ \wedge \\ \neg[[COLLECTOR : c] \rightarrowtail (BUY) \rightarrowtail [PAINTING : b]]].$$

(5)

Graph 5 that conveys the information "A certain artist paints a certain painting and a certain collector does not buy it".

In graph 5 the brackets following the connective ¬ enclose a SUB GRAPH *[COLLECTOR: c]* $\rightarrowtail$ *(BUY)* $\rightarrowtail$ *[PAINTING: b].*

The following conventions are employed for negations:

- ¬ in front of a context simply means that the graph in the context is negated, like in graph 5.
- ¬ in the concept node, like $[\neg P : x]$ means that there exists a variable x whose type is not P.
- The $\sim$ -symbol is reserved to represent negation in relations. For example, the graph

  $$[P : x] \rightarrowtail (\sim R) \rightarrowtail [Q : y]$$

  means that there exists a variable x of type P and a variable y of type Q and they are not related by R.

In the appendix of this paper, we present some more complicated examples of knowledge representation with conceptual graphs. The same information will be represented in FOPL, too, to enable comparison of "readability" of FOPL formulas and conceptual graphs. Moreover, in the appendix, the translation from FOPL formula into a conceptual graph is represented using the algorithm of section 4.

3.1.3 The grammar of conceptual graphs

The set of terminal symbols, T_{cg}, for the language of conceptual graphs (limited syntax) is

$T_{cg} = \{\wedge, \neg, \vee, \rightarrow, \leftrightarrow, \forall, (,), [,], x_0, x_1, .., c_0, c_1, .., P_0, P_1, .., R_0, R_1, .., \top_c, \sim, \leftarrowtail, \rightarrowtail\}$.

The production rules of the grammar are.

1. $U \mapsto x_0|x_1|x_2|..$ (Variables are referents.)
2. $U \mapsto c_0|c_1|c_2|..$ (Constants are referents.)
3. $P \mapsto \top_c|P_0|P_1|P_2|..$ (Concept type symbols.)
4. $R \mapsto R_0|R_1|R_2|..$ (Relation type symbols.)
5. $A \mapsto [P:U]|[\forall P:U]|[\neg P:U]$ (Concept nodes.)
6. $F \mapsto A$ (Concept nodes are conceptual graphs.)
7. $F \mapsto A \rightarrowtail (R) \rightarrowtail A|A \leftarrowtail (R) \leftarrowtail A$ (Concept nodes, joined by a relation, are conceptual graphs.)
8. $F \mapsto A \rightarrowtail (\sim R) \rightarrowtail A|A \leftarrowtail (\sim R) \leftarrowtail A$ (Negated relation.)
9. $F \mapsto \neg[F]$ (A negated conceptual graph is a conceptual graph.)
10. F $\mapsto$ [F $\wedge$ F]
11. F $\mapsto$ [F $\vee$ F]
12. F $\mapsto$ [F $\rightarrow$ F]
13. F $\mapsto$ [F $\leftrightarrow$ F] (Connectives.)

A graph is called SIMPLE if it has no connectives.

3.1.4 Quantifiers – The scopes

Previously, we stated that a there are no free variables in a conceptual graph. In this section, the scope of quantifiers in graphs is discussed.

For example, the graph 6

$$[[ARTIST : x] \wedge [RICH : x]] \tag{6}$$

could be interpreted in two ways

a) in the same way as the FOPL formula $(\exists x(artist(x)) \wedge \exists x(rich(x)))$, i.e. “there exists someone who is an artist and someone who is rich” or

b) in the same way as the FOPL formula $\exists x((artist(x) \wedge rich(x))$, i.e. “there exists someone who is an artist and rich”.

For consistency it is reasonable to maintain the LATTER interpretation. In practice this means that different variable symbols are used to represent different individuals of UoD.

Equally, the variable x in the graph 7 refers to the same individual.

$$[[RICH : x] \wedge \neg[[ARTIST : x] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : z]]] \tag{7}$$

A similar convention will be used with the universal quantifier, too. Let us consider graphs 8 (“all the collectors buy all the paintings”) and 9

$$[COLLECTOR : \forall x] \rightarrowtail (BUY) \rightarrowtail [PAINTING : \forall y] \tag{8}$$

$$[COLLECTOR : \forall x] \rightarrowtail (BUY) \rightarrowtail [PAINTING : \forall x]. \tag{9}$$

According to the previous agreement of the scopes of the quantifiers, the graph 9 is evidently not meaningful though syntactically correct.

Equally, in the graph 10 it is evident that the same implicit existential quantifier binds the *x* variables of both the *ARTIST* nodes.

$$\neg[[[ARTIST : x] \wedge \neg[[ARTIST : x] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : y]]]] \tag{10}$$

3.1.5 Quantifiers – The universal quantifier

The following graphs, 11 and 12, reveal a semantic problem that is related with the universal quantifier:

$$[PAINTING : \forall x]. \tag{11}$$

$$[PAINTING : \forall x] \rightarrowtail (PAINT) \rightarrowtail [ARTIST : y]. \tag{12}$$

The graph 11 expresses that "every object in the UoD is a painting". However, the graph 12 does not claim that, but it expresses that all those objects in the UoD that are paintings have an artist. Thus, the graph 12 could be presented in the form 12':

$$[[PAINTING : \forall x] \rightarrow [PAINTING : x] \rightarrowtail (PAINT) \rightarrowtail [ARTIST : y]].$$

3.1.6 Quantifiers – Quantifier sequences

As explained in section 2.3, for the sake of clarity it is sometimes profitable to use inverse relations. In conceptual graphs the inverse relations can be presented in an easy and intuitive way: by reversing the direction of $\rightarrowtail$ or $\leftarrowtail$ symbols connecting the relation symbol with the concept nodes.

Concerning FOPL and conceptual graphs, the following analogies seem obvious (for translation details, see section 4):

The the FOPL formula $\forall x(\exists y(R(x,y)))$, the CG counterpart is $[\top_c : \forall x] \rightarrowtail (R) \rightarrowtail [\top_c : y]$.

For the formula $\forall y(\exists x(R(x,y)))$ that equals to $\forall y(\exists x(R^{-1}(y,x)))$ the counterpart is $[\top_c : \forall y] \leftarrowtail (R) \leftarrowtail [\top_c : x]$.

Using the inverse relation, the order of variables in quantifier sequences can be preserved, which is an advantage in the translation process.

3.1.7 Quantifiers – Summary and algorithm

The scopes of the quantifiers are essential in terms of semantics. Since the scopes of quantifier in graphs are more implicit than in FOPL, we present a simple algorithm *make_quantifiers_explicit*.

The algorithm transforms it input (a graph) into a form where quantifiers and variables are presented in front of the context where they first appear. This form, the explicit quantifier form, is utilised when discussing the semantics of graphs.

The algorithm is recursive and it uses a list of variables, V.

Let G be a graph. Those subgraphs of G that have universal quantifiers, are assumed to be in the same form as in the graph 12'.

make_quantifiers_explicit(G) =

> If there is a relation node in G:
>
> Traverse in the direction of the arrows of the relation. For both the concept nodes:

if the concept node is of form $[P : x]$ and if the variable, x, does not appear in the list V, insert $\exists x$ in the front of the current context and add x into the list V.

if the concept node is of form $[P : \forall x]$ and if the variable, x, does not appear in the list V, insert $\exists x$ in the front of the current context and add x into the list V.

If there is a concept node in G

if the concept node is of form $[P : x]$ and if the variable, x, does not appear in the list V, insert $\exists x$ in the front of the current context and add x into the list V.

if the concept node is of form $[P : \forall x]$ and if the variable, x, does not appear in the list V, insert $\exists x$ in the front of the current context and add x into the list V.

For each subgraph of G, G_i, $make\_quantifiers\_explicit(G_i)$

When the *make_quantifiers_explicit* -function is applied to graphs 6 - 10 the explicit quantifier forms of the graphs are:

6
$$\exists x[[ARTIST : x] \wedge [RICH : x]]$$

7
$$\exists x[[RICH : x] \wedge \neg[\exists z[ARTIST : x] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : z]]]$$

8
$$\forall x \forall y[[[COLLECTOR : \forall x] \wedge [PAINTING : \forall y]] \rightarrow \\ [COLLECTOR : \forall x] \rightarrowtail (BUY) \rightarrowtail [PAINTING : \forall y]]$$

9
$$\forall x[[[COLLECTOR : \forall x] \wedge [PAINTING : \forall x]] \rightarrow \\ [COLLECTOR : \forall x] \rightarrowtail (BUY) \rightarrowtail [PAINTING : \forall x]]$$

10
$$\neg[\exists x[[ARTIST : x] \wedge \neg[\exists y[ARTIST : x] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : y]]]]$$

3.2 Semantics of L_{CG}

The semantics for the language *Lcg* is based on the same principles as the semantics for the language L_{FOPL}. Let Lcg' be an extended language, M a model and EA the explicit quantifier form of a graph A.

The following rules define the truth conditions TC_{cg} for a graph A in the model M.

1. Interpretation of concept nodes:

   Let t be a term. If the graph is of form $[P:t]$, then it is true in M if and only if the counterpart of t in UoD is a member of the extension of P.

   If the graph is of form $[\neg P:t]$, then it is true in M if and only if the counterpart of t in UoD is not a member of the extension of P.

2. Let B and C be concept nodes and t_B and t_C terms such that t_B is the referent of B and t_C is the referent of C.

   If the graph is of form $B \rightarrowtail (R) \rightarrowtail C$, the it is true in M if and only if

   - B is true in M and
   - C is true in M and
   - the counterparts of t_B and t_C in UoD form a pair $< t_B, t_C >$ that is a member of the extension of R.

3. If the graph is of form $B \rightarrowtail (\sim R) \rightarrowtail C$, then it is true in M, if and only if

   - B is true in M and
   - C is true in M and
   - the counterparts of t_B and t_C in UoD form a pair $< t_B, t_C >$ that is not a member of the extension of R.

4. If the graph is of form $\neg[B]$, then it is true in M, if and only if B is not true in M.

5. If the graph is of form $[B \wedge C]$, then it is true in M, if and only if B and C are true in M.

6. If the graph is of form $[B \vee C]$, then it is true in M, if and only if either B or C or both are true in M.

7. If the graph is of form $(B \rightarrow C)$, the it is true in M, if and only if it is NOT the case that B is true in M, and C is not true in M.

8. If the graph is of form $(B \leftrightarrow C)$, then it is true in M, if and only if

   - both B and C are true in M
     or
   - both B and C are not true in M.

9. If *EA* includes the expression $\exists x$, and there is a node $[B : x]$ in the graph, then the graph is true in M if and only if the counterpart of x in UoD is a member of the extension of B.

10. If *EA* includes the expression $\forall x$, and there is a node $[B : x]$ in the graph, then the graph is true in M if and only if *B(x/a)* is true for all the elements a in UoD.

On the basis of these conditions, one can easily observe that the truth conditions for connectives in conceptual graphs are equal to the truth conditions for connectives in FOPL. Therefore, the shortcut rules (S1 - S9, S13 - S16)) that are related with connectives, can be applied to conceptual graphs, too.

4 Translations, discussion and summary

On the basis of the truth conditions TC_{fol} and TC_{cg} it is easy to justify the following analogies between conceptual graphs and FOPL:

- The FOPL formula $P(t)$ is semantically identical with the conceptual graph $[P : t]$, since $P(t)$ and $[P : t]$ are true in M under the same conditions.
- Analogously, the FOPL formula $R(t, u)$ is semantically identical with the conceptual graph $[\top_c : t] \rightarrowtail (R) \rightarrowtail [\top_c : u]$.

Based on these analogies, it is possible to design an algorithm that translates a given FOPL closed formula into a conceptual graph that (according to the truth conditions) has the meaning as the formula. The algorithm as a whole will be presented in an extended version of this paper, but in principle, it works as follows:

Let A be a closed FOPL formula where all the predicates are either unary or binary. Using the notion in 2.3, it is possible to generate a form of A where the variables in predicates are in the same order as in the quantifier sequence. Moreover, A can be processed in to the translation form represented in section 2.4. This form of A can be translated into explicit quantifier form of a conceptual graph. The resulting conceptual graph can be constructed simply by erasing the quantifier sequences.

Now, since FOPL formulas with only unary and binary predicates on one hand and conceptual graphs (with limiyed syntax) on the other hand can be translated into each other, it can be claimed that these languages have equal expressive power. This fact has many interesting consequences. It is possible to design a proof theory for conceptual graphs and study the language, its semantics and its proof theory. Together, they form an interpreted system of conceptual graphs, in the same manner as there are interpreted systems of FOPL. This kind of system of conceptual graphs was suggested by Sowa (see [6]) and he maintains that the system is sound and complete.

Moreover, the expressive power of conceptual graph with limited syntax is greater than that of logic programming, a subset of FOPL.

Since conceptual graphs and FOPL have equal expressive power, the use of conceptual graph could be justified by their "elegancy" or ease of use. Yet, as some of the examples in the appendix demonstrate, the representation using conceptual graphs is sometimes more complex than the one using FOPL.

In an "unlimited" language of conceptual graphs, the referent of a concept node can be a set instead of a single term. This kind of language has an expressive power that exceeds FOPL, since a first order predicate language with arbitrarily changing arities sounds computationally very demanding. Therefore it is difficult to estimate the properties of an interpreted system that is based on it.

Acknowledgements

The author wishes to thank the following persons for comments and advice: Matti Heikkurinen, Marko Junkkari, Hannu Kangassalo, Erkki Mäkinen, Timo Niemi, Tommi Vehkavaara and especially Ari Virtanen.

References

[1] M. Jackman & C. Pavelin. Conceptual graphs. In G.A. Ringland & D.A. Duce, editors, *Approaches to Knowledge Representation*. Wiley and Sons, 1989.

[2] V. Rantala & A. Virtanen. *Logiikan peruskurssi*. Matemaattisten tieteiden laitos, Tampereen yliopisto, 1990.

[3] D. D. Roberts. *The Existential Graphs of Charles S. Peirce*. Mouton, 1973.

[4] G. Rozenberg & A. Salomaa. *Cornerstones of Undecidability*. Prentice-Hall, 1994.

[5] H. Salminen & J. Väänänen. *Johdatus logiikkaan*. Gaudeamus, 1993.

[6] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison Wesley, 1984.

Appendix

G1. a is a painting:

$$painting(a).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$[PAINTING : a].$$

G2. There exists (at least one) painting:

$$\exists x(painting(x)).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$[PAINTING : x].$$

G3. There exists something that is not a painting:

$$\exists x(\neg(painting(x))).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$[\neg PAINTING : x].$$

G4. There is no (such thing as) kitsch.

$$\neg(\exists x(kitsch(x))).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$\neg[[KITSCH : x]].$$

G5. There are artists and collectors:

$$(\exists x(artist(x)) \wedge \exists y(collector(y))).$$

In the translation form the formula is $(\exists x(\exists y((artist(x)) \wedge (collector(y))))$. The graph counterpart for this formula is

$$[[ARTIST : x] \wedge [COLLECTOR : y]].$$

G6. Some one is both an artist and a collector.

$$\exists x((artist(x) \wedge collector(x))).$$

The formula is in the translation form. The graph counterpart for this formula is

$$[[ARTIST : x] \wedge [COLLECTOR : x]].$$

G7. There is at least some one that is an artist and not a collector.

$$\exists x((artist(x) \wedge \neg(collector(x)))).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$[[ARTIST : x] \wedge [\neg COLLECTOR : x]].$$

G8. Everything is art:

$$\forall x(art(x)).$$

The formula is in the translation form. The counterpart for this formula is the conceptual graph

$$[ART : \forall x].$$

G9. For everything, there is something that is more beautiful than it:

$$\forall x(\exists y(morebeautiful(y,x))).$$

Using the inverse relation we get:

$$\forall x((\exists y)(morebeautiful^{-1}(x,y))),$$

that is the translation form of this formula. The graph counterpart is thus:

$$[\top_c : \forall x] \leftarrow (MOREBEAUTIFUL) \leftarrow [\top_c : y].$$

G10. There is something that is more beautiful than anything else:

$$\exists x(\forall y(morebeautiful(x,y))).$$

The formula is in the translation form. Its graph counterpart is

$$[\top_c : x] \rightarrowtail (MOREBEAUTIFUL) \rightarrowtail [\top_c : \forall y].$$

G11. The collector b buys the painting a:

$$((collector(b) \wedge painting(a)) \wedge buy(b,a)).$$

The translation form of the formula is

$$((collector(b) \wedge (painting(a)) \wedge buy(b,a)).$$

The counterpart of this formula is the conceptual graph

$$\begin{array}{c} [[[COLLECTOR : b] \rightarrowtail (BUY) \rightarrowtail [PAINTING : a] \\ \wedge \\ [[PAINTING : a] \wedge [COLLECTOR : b]]]. \end{array}$$

According to the truth conditions TC_{cg}, this means the same as

$$[COLLECTOR : b] \rightarrowtail (BUY) \rightarrowtail [PAINTING : a].$$

G12. Some collector buys the painting a:

$$[COLLECTOR : x] \rightarrowtail (BUY) \rightarrowtail [PAINTING : a].$$

$$\exists x(((collector(x) \wedge painting(a)) \wedge buy(x,a)))).$$

The translation process is similar to that in G11.

G13. There exists a collector who does not buy the painting a:

$$\exists x((collector(x) \wedge (\neg(buy(x,a)) \wedge painting(a)))).$$

The translation form of the formula is

$$\exists x((\neg(buy(x,a)) \wedge (collector(x) \wedge painting(a)))).$$

The counterpart for this formula is the conceptual graph

$$\begin{gathered}[[[COLLECTOR : b] \rightarrowtail (\sim BUY) \rightarrowtail [PAINTING : a] \\ \wedge \\ [[PAINTING : a] \wedge [COLLECTOR : b]]].\end{gathered}$$

According to the truth conditions TC_{cg}, this means the same as

$$[COLLECTOR : b] \rightarrowtail (\sim BUY) \rightarrowtail [PAINTING : a].$$

G14. No collector buys the painting a:

$$\neg(\exists x((collector(x) \wedge painting(a) \wedge buy(x,a))).$$

The translation form of the formula is

$$\neg(\exists x(buy(x,a) \wedge (collector(x) \wedge painting(a)))).$$

The counterpart for this formula is the conceptual graph

$$\begin{gathered}\neg[[COLLECTOR : x] \leftarrowtail (BUY) \leftarrowtail [PAINTING : a] \\ \wedge \\ [COLLECTOR : x] \wedge [PAINTING : a]].\end{gathered}$$

G15. No collector buys any painting:

$$\neg(\exists(x)(\exists(y)((collector(x) \wedge (painting(y) \wedge buy(x,y)))))).$$

The translation process, similar to the one in the previous example leads to:

$$\begin{gathered}\neg[[[COLLECTOR : x] \leftarrowtail (BUY) \leftarrowtail [PAINTING : y] \\ \wedge \\ [[COLLECTOR : x] \wedge [PAINTING : y]]]].\end{gathered}$$

G16. Each painting has been painted by some artist:

$$\forall x(painting(x) \rightarrow \exists y((artist(y) \wedge paint(x,y)))).$$

The process of transforming the formula into the translation form is as follows:

Eliminate $\rightarrow$:

$$\forall x(\neg(painting(x) \wedge \neg(\exists y((artist(y) \wedge paint(x,y)))))).$$

Move quantifiers:

$$\forall x(existsy((\neg(painting(x) \wedge \neg((artist(y) \wedge paint(x,y)))).$$

The graph counterpart of this formula is:

$$\neg[[[PAINTING : \forall x] \wedge \neg[[ARTIST : y] \rightarrowtail (PAINT) \rightarrowtail [PAINTING : y]]]].$$

According to the semantics of the quantifier $\forall$ and the shortcut rule S6 this graph is equal to

$$[PAINTING : \forall x] \rightarrowtail (PAINT) \rightarrowtail [ARTIST : y].$$

G17. There exists a painting that all the collectors want:

$$\exists x((painting(x) \wedge \forall y((collector(y) \rightarrow want(x,y))))).$$

The translation prosess is similar to that in the previous example. Its result is the graph:

$$[[PAINTING : x] \wedge$$
$$[COLLECTOR : \forall y] \rightarrowtail (WANT) \rightarrowtail [PAINTING : x]].$$