Conceptual graphs for semantics and knowledge processing

by Jean Fargues
Marie-Claude Landau
Anne Dugourd
Laurent Catach

This paper discusses the representational and algorithmic power of the conceptual graph model for natural language semantics and knowledge processing. Also described is a Prolog-like resolution method for conceptual graphs, which allows one to perform deduction on very large semantic domains. The interpreter that we have developed is similar to a Prolog interpreter in which the terms are any conceptual graphs and in which the unification algorithm is replaced by a specialized algorithm for conceptual graphs.

Introduction

The conceptual graph model seems to be a very promising unified model because it generalizes many ideas contained in preceding work on natural language semantics, such as that of Fillmore [1], Schank [2], Montague [3], Wilks [4], and Kamp [5], for example. This model is a general framework for representing knowledge, and it can be used as the core model of future integrated knowledge systems.

The conceptual graph model was introduced by John Sowa in his book [6] in a rigorous and formal way.

The first purpose of this paper is to emphasize the main properties of this model by comparing them with the properties which are usually required in any powerful model for natural language semantics. The second purpose is to describe the principle of a Prolog-like deductive system based on conceptual graphs that we have implemented.

We present in the first section the conceptual graph model, and we discuss its representational and algorithmic power in regard to the Montague-semantics-based approach and the traditional logic-based approach. We discuss the properties of the model itself but also its use for natural language understanding.

We present in the second section a Prolog-like resolution method which allows us to express a large amount of background knowledge in terms of conceptual graphs and to perform deduction on very large linguistic and semantic domains. The interpreter that we have developed is similar to a Prolog interpreter in which the terms are any conceptual graphs and in which the unification algorithm is replaced by a specialized algorithm for conceptual graphs. This conceptual graph machinery has been implemented in Prolog [7, 8].

The conceptual graph processor will be the main component of a general system for knowledge acquisition from texts that we are developing at the IBM Paris Scientific Center [9].

Conceptual graph model and natural language semantics

• *Concepts and conceptual relations*

Three types of objects are introduced:

- *Referents*, which denote individuals, values, or sets explicitly mentioned in the world or in the universe of discourse.
  Example: 'John', 3.14159, and {Paris, Tokyo} are referents.
- *Concept types*, denoted by a concept type label inside a box (or between brackets for a more convenient notation).
  Example: [PERSON], [COLOR], and [EAT] are concepts.
- *Conceptual relations*, denoted by a relation label inside a circle (or between parentheses for a more convenient notation).
  Example: (LOC), (AGT), and (INST) denote the conceptual relations "localization," "agentive," and "instrumental."

The set of concept types and the set of conceptual relations are assumed to be given (we return to this point later because these sets are not arbitrarily closed).

A concept can be restricted to a particular realization by introducing a referent.

Example: [PERSON:'Joe'], [CITY:{Paris, London}], [COLOR:'red'].

Concepts can be linked by conceptual relations to form a *conceptual graph* (see **Figure 1**). The formation rules are purely syntactic, and we assume that all conceptual relations are binary. At this level we do not make any assumption regarding the validity of the "meaning" of a conceptual graph.

• *The concept type hierarchy*

Concept types can be merged in a lattice whose partial ordering relation $<$ can be interpreted as a categorical generalization relation (in an Aristotelian perspective).

Thus, PERSON $<$ ANIMATE and EAT $<$ ACT can represent the facts that a person is animate and that to eat is an action. The top and the bottom concept types of the lattice are, respectively, UNIV (the universal entity type) and ABSURD (the "absurd" entity type).

In the preceding example, we say that the concept type ACT can be *restricted* to the concept type EAT or that PERSON is a *restriction* of ANIMATE.

The $<$ relation can be extended to concepts having referents, as suggested by the example

Example: [PERSON:'Joe'] $<$ [PERSON] and therefore [PERSON:'Joe'] $<$ [ANIMATE].

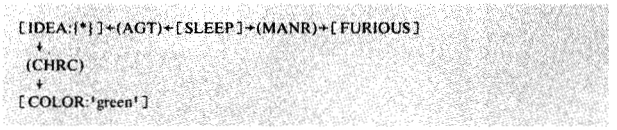

Figure 1

A conceptual graph corresponding to something like "(a set of) green ideas sleep furiously."

We say in this case that the concept ANIMATE can be restricted to the concept [PERSON:'Joe']. Two identical concepts having different referents are not comparable.

Given two comparable concept types, it is possible to exhibit their "least common generalization" and by duality their "greater common specialization." When two concepts are not semantically comparable, their least common generalization is UNIV and their greater common specialization is ABSURD.

Example:

- ANIMATE could be the least common generalization of ANIMAL and PERSON.
- SCIENCE_FICTION_ROBOT could be the greater common specialization of ARTIFACT and PERSON.

The definition of the relation $\leq$ on concepts can be extended to a partial relation on conceptual graphs:

- Let u and v be two conceptual graphs. Suppose that u contains a subgraph u' which is identical to v. In this case, the graph u is more "specialized" than the graph v (i.e., it expresses something more detailed than v). We say that $u \leq v$.
- By extension of this first case, if there is a subgraph u' of u that is isomorphic (modulo the $\leq$ relation on concepts) to v, then $u \leq v$. By "isomorphic" we mean that the graph u' can be obtained from v by restricting some of its concepts.

• *The join and generalization algorithms*

It is possible to build new conceptual graphs from a set of existing conceptual graphs by applying the following formation rules [6]:

- Copy a graph.
- Restrict a graph by replacing some concepts by restricting these concepts.
- Join two conceptual graphs, as follows (see **Figure 2**):
  - Form the maximal common overlap from the two graphs: This overlap is the maximal conceptual graph,

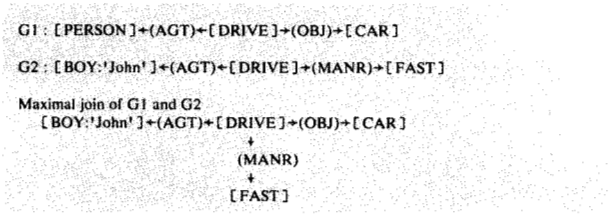

Figure 2

Example of maximal join of two conceptual graphs.

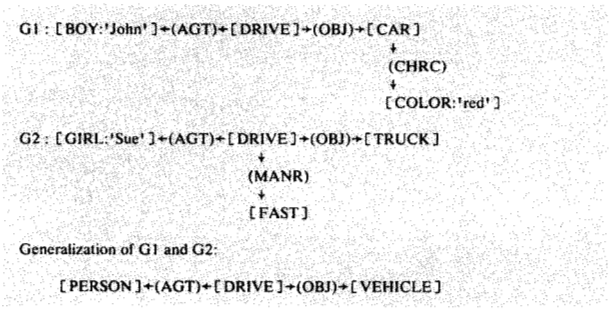

Figure 3

Example of generalization of two conceptual graphs.

which is the common restriction of two corresponding subgraphs of the two graphs. We require that it must be a connected graph.

- Attach the pending parts remaining in the two graphs to the maximal common overlap.

- Simplify a conceptual graph by suppressing the redundant occurrences of identical edges in a graph.

The notion of "maximal join" can be introduced if we add to the preceding rules that the maximal overlap used to form the join is maximal in two ways: It is maximal because it contains a maximal number of edges obtained by restriction from the two original graphs, but also because its concepts are the greater common specializations of the concepts coming from the original graphs.

We say that the join of two graphs fails whenever the maximal overlap degenerates into a single concept of type ABSURD (no comparable concepts in the two graphs).

The "maximal join" is not necessarily unique because it may happen that several maximal overlaps exist between the two graphs. Thus, the extension of the relation $\leq$ to conceptual graphs does not confer a lattice structure on the set of all conceptual graphs.

It is possible to write an algorithm which gives the maximal joins of two conceptual graphs. This algorithm is an extension of the unification algorithm because it gives as result a "greater common specialized graph," which can be built from two conceptual graphs.

It is possible to implement this algorithm in Prolog as a predicate which gives all the possible maximal joins using the backtracking facilities of this logic programming language [10].

Another interesting algorithm is the generalization algorithm (see **Figure 3**). It gives as result the least common generalized conceptual graph obtained from two graphs. This least common generalized graph can be built by forming a common overlap which contains the least generalized concepts obtained from couples of concepts coming from the two original graphs (instead of the greater common specialization of concepts in the case of join). We say that the generalization algorithm fails when its result is the [UNIV] degenerated graph.

This algorithm is very useful for acquiring general definitions from a set of particular examples. It is one of the algorithms to be used in the learning component of a knowledge system based on the conceptual graph model.

The maximal join algorithm and the generalization algorithm have been implemented in Prolog [7].

- *The abstraction operation and the conceptual graph model*

A typed formal model must have an abstraction operation to be a powerful candidate for representing natural language semantics. This abstraction operation allows one to define new concepts or relations by composing existing concepts and relations.

The second aspect of this abstraction principle is that it must be possible to distinguish between the individuals referenced in a particular discourse, the classes of individuals, and the properties of these classes.

This abstraction principle is obviously needed if we consider the definitions given in the dictionaries or the paraphrasing process that we can use to make the meaning of a discourse precise.

In the Montague semantics-based approach [3], the abstraction operator was introduced as the lambda abstraction operator because this semantic model is based on a particular interpretation of the typed lambda calculus used to denote natural language semantics. Thus, a particular "pen" can be denoted by the 0-order expression $\lambda()$[pen123], the class of all pens can be defined by the 1-order expression $\lambda(x)$[pen(x)], and this class is a particular subclass of the class noted by the second-order expression $\lambda(P)$[writing-artifacts(P)], where the functional variable P can be instantiated to the class "pen."

It is also possible to define the class of all musicians who play an instrument y by a logical form such as

$\lambda(x,y)$[person(x) $\wedge$ play(x,y) $\wedge$ music_instrument(y)]

or by the functional expression

λ(x,y)[play(person(x),music_instrument(y))].

The conceptual graph model also allows us to define new concepts and new conceptual relations using an abstraction operation. We give in **Figure 4** an example of definition statements for a new concept and for a new conceptual relation.

Whenever a definition is given, the new concept is added to the concept type lattice: In the example following, ART_SPONSOR < PERSON will be stored in the lattice.

The *contraction operation* takes as arguments a conceptual graph and a definition. It tries to replace a subgraph of the given conceptual graph by a single concept (or relation) using the definition of this concept. This contraction process also takes into account the restriction relation on the concepts of the conceptual graph in which the substitution is applied (see **Figure 5**).

The contraction algorithm applies a partial substitution when the graph to be contracted is more restricted, as shown in **Figure 6**. Thus, we do not lose the information which was contained in the original graph.

It is sometimes possible to perform several contractions on the same graph when this graph contains multiple occurrences of the definition graph. In this case, the contraction algorithm may be iterated on the graph obtained at each step.

Another useful algorithm is the *expansion* algorithm, which unfolds a single concept (or relation) occurring in a conceptual graph by replacing this occurrence with its definition graph.

• *From syntax to semantics using conceptual graphs*

A classical property of the formal models for natural language semantics used in AI is that they obey the compositionality principle. It is usually assumed that a representation of the semantics of an entire sentence can be built by combining the semantic representations associated with its components.

A classical approach consists of starting from a traditional syntactic tree to build the semantic representation of the sentence. A semantic form is assumed to be given (or selected) for each word in a *semantic lexicon*. An operator is defined to build the semantic form attached to a parent node of the syntactic tree. This operator takes as arguments the semantic forms attached to the son's nodes and gives as result a more complex composed semantic form.

If we consider the Montague semantic approach or related work such as [11], logical forms describing the semantics of the words in the lexicon can be represented as expressions in the lambda-calculus formalism. The operator which combines the logical forms to obtain a semantic representation for a sentence is in this case the lambda-evaluation operator (see **Figure 7**). The lambda expression of

```
type ART_SPONSOR(*x) is
[PERSON:*x]←(AGNT)←[GIVE]→(OBJ)→[MONEY]
                          ↓
                        (RCPT)
                          ↓
                       [ARTIST]
relation BROTHER(*x,*y) is
[BOY: *x]←(CHLD)←[PERSON: *]→(CHLD)→[PERSON: *y]
```

Figure 4

Definitions of a new concept and a new relation.

```
[MAN:'Smith']←(AGNT)←[GIVE]→(OBJ)→[MONEY]
  ↓                     ↓
(ATTR)                (RCPT)
  ↓                     ↓
[RICH]                [ARTIST]
Using the ART_SPONSOR definition, the contraction gives
  [ART_SPONSOR:'Smith']→(ATTR)→[RICH]
```

Figure 5

The contraction operation: a simple case.

```
[MAN:'Smith']←(AGNT)←[GIVE]→(OBJ)→[MONEY]
  ↓                     ↓
(ATTR)                (RCPT)
  ↓                     ↓
[RICH]                [PAINTER:'Jones']
The partial contraction gives
[ART_SPONSOR:'Smith']←(AGNT)←[GIVE]→(RCPT)→[PAINTER:'Jones']
  ↓
(ATTR)
  ↓
[RICH]
```

Figure 6

A partial contraction of a conceptual graph.

```
to love: λ(x,y)[love(x,y)]
John   : λ()[John]
girl   : λ(x)[girl(x)]
a      : λ(P,x)[P(x)]

      S      → λ(v)[love(John,girl(v))]
    /  \
  NP   VP    → λ(u,v)[love(u,girl(v))]
 /     /  \
'John' V   NP   → λ(x)[girl(x)]
      /    /  \
  'loves' DET  N
         /      \
       'a'      'girl'
```

Figure 7

Example of a logical-form-based approach.

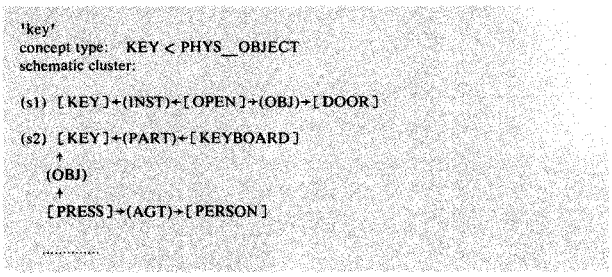

Figure 8

Example of schematic cluster for the word 'key'.

a parent node in the syntactic tree is obtained by evaluating a lambda expression of a son node with the other son expression as argument (the direction of this evaluation is indicated by ← and → on the figure). The logical form for this sentence could be made closer to predicate logic by rewriting it as

(F1): exists(v) [love(John,v) ∧ girl(v)]

but it is possible to obtain something like

(F2): exists(e), exists(v) [agt(John,e) ∧ obj(v,e) ∧ girl(v) ∧ love(e)]

where e is introduced as an "event" variable.

These variants depend on the complexity of the lambda expressions associated with the words in the semantic lexicon, but the compositionality principle taken into account by the lambda evaluation remains the same.

The important point now is that it is necessary to express *semantic constraints* in the lexicon.

For example, we would like to specify that the agent of "to think" must be a person. We would like also to define *semantic contexts* in order to handle more complex polysemic cases (example: "to run" in "John runs a mile" and in "the program is running very well").

The logical form approach is generally well based theoretically, but this approach makes it difficult to represent semantic constraints. These constraints are generally specified in the grammar rules by tests on semantic markers (such as "human" or "not human"). Thus, these tests appear as extraneous ad hoc specifications outside the lambda calculus or the predicate logic formulation for the semantics.

Another problem is the lack of flexibility stemming from the fixed arity assigned to the predicate symbols introduced in these models.

It is usual to introduce a predicate symbol to denote the semantics of a verb. The fixed arity of the chosen predicate symbol makes it difficult to represent multiple complement structures [leave(x,y) in "John left Chicago," but what is the predicate arity for "leave" in "John left Boston with Mary for New York by the highway"?].

Sowa, following Fillmore, introduces a set of semantic case relations such as agentive, object, instrumental, and so on. If we look at the logical formalism, it is also possible to introduce these case relations (such as in the F2 logical form given before), but special variable types must be introduced to represent events or possible worlds [5, 12].

Now, what about these problems using the conceptual graph model?

We can associate with a word a concept type and its place in the type lattice.

The semantic constraints can be associated with the words in a semantic lexicon by using what are called *canonical graphs.*

Example: With the verb "to think" one can associate the canonical graph

[PERSON]←(AGT)←[THINK]→(OBJ)→[PROPOSITION]

The possible semantic contexts are introduced by *schemata* which are also conceptual graphs. Thus, the "meaning by use" for a word can be represented by a set of schemata. This set is called a *schematic cluster* (see **Figure 8**).

Building a conceptual graph as a semantic representation of the meaning of a sentence obeys the compositionality principle as described for the logical form approach. Starting from a syntactic tree, the operator to be applied at a parent node of the tree is now the maximal join of the conceptual graphs associated with the son nodes [13]. Thus the join operation plays the same role as the lambda evaluation used in the logical form approach.

The initially selected graphs for the words of the sentence are either their canonical graphs or some schemata of their schematic cluster. We have assumed here for simplicity that there is a one-to-one mapping from the words to the concept types. We make this point more precise later in our discussion of the semantic lexicon.

Suppose that the concept type OPEN is associated with the verb "to open" and that the canonical graph for this verb is

(c1): [PERSON]←(AGT)←[OPEN]→(OBJ)→[PHYSICAL_OBJECT]

Then, the conceptual graph for the sentence "John opens the door with a key" will be

[PERSON:'John']←(AGT)←[OPEN]→(OBJ)→[DOOR]
↓
(INST)
↓
[KEY]

This graph is obtained by joining the canonical graph for "to open" and the (s1) schemata for "key" in Fig. 8.

Two important points must be emphasized.

The first one is that the join operator has taken into account the basic semantic constraints expressed by

canonical graphs. In the example given before, the canonical graph for "to open" encoded this kind of constraint.

The second one is that the *preference semantic principle* (as described by Wilks) is implicitly used by joining the schemata. In the given example this principle is applied by the selection of the (s1) schemata for "key." This selection is achieved in a natural way because of the mapping of the schemata with the current semantic context.

The problem of the arity of the predicate symbols has disappeared because we handle graphs instead of fixed-arity logical forms.

In the preceding example, a crucial point is the selection process of the right conceptual graphs in order to build a correct graph for the sentence. It may happen that a join fails in this process, either because the sentence is semantically not well founded or because a polysemic word occurs in this sentence. In the latter case, another choice can be made for the canonical graph or the schemata corresponding to the suspected polysemic word of the sentence. So, the process can backtrack.

This problem is closely related to the structure of the semantic lexicon: One polysemic word may correspond to a single concept type (as in the "key" example). In this case, the polysemy can be encoded as distinct graphs in the schematic cluster. It can happen also that a word corresponds to distinct concept types. For example, the verb "to open" could point to the concept type BEGIN and could have as a second canonical graph

(c2): [PERSON]←(AGT)←[BEGIN]→(OBJ)→
[COMMUNICATION_PROCESS]

If we consider a sentence such as "John opens the session by pressing the enter key" the process suggested above will give as result

```
[PERSON:'John']←(AGT)←[BEGIN]→(OBJ)→[SESSION]
                                ↓
                             (SRCE)
                                ↓
                          [PROPOSITION:
```

```
[PERSON:'John']←(AGT)←[PRESS]→(OBJ)→[KEY:'enter']
                                        ↓
                                     (PART)
                                        ↓
                                   [KEYBOARD]]
```

We do not give all the details about this example, but the background information used in the underlying join process would be the fact that SESSION < COMMUNICATION_PROCESS (assumed to be stored in the concept hierarchy) and the conceptual graphs (c2) and (s2).

In the case of second-order sentence structures (such as subordinate or completive constructions), we can use the concept type PROPOSITION, whose referent may be a conceptual graph. A set of unary conceptual relations on PROPOSITION concepts can be introduced too (see **Figure 9**).

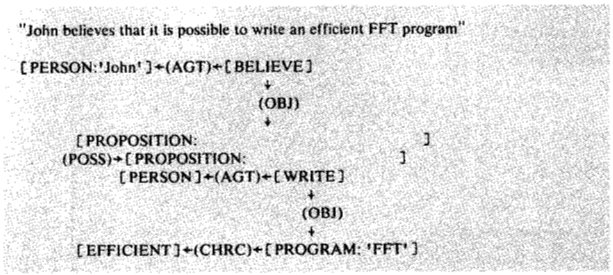

Figure 9

Conceptual graph for second-order phrase structures.

The conceptual graph model allows us to handle almost all of the traditional particular representation problems (tense, modalities, quantification, . . .). Some of these problems are discussed in Sowa's book [6].

Thus, it is possible to express a large quantity of information in the semantic lexicon in a flexible way. Of course, all the information describing a sample world must also be expressed as "pragmatic assertions or inferences" in a more complex formalism. We describe in the next section how to specify this knowledge using a Prolog-like formalism (i.e., a "Prolog" whose terms are any conceptual graphs).

Prolog-like deduction on conceptual graphs

- *Conceptual graphs and logic*

Before introducing the notion of deduction on conceptual graphs, we make more precise some points about the logical interpretation of a conceptual graph considered as an assertion.

As mentioned in the preceding paragraphs, the conceptual graph model is able to represent high-order logical relations which are difficult to represent in a simple first-order logical formalism.

If we consider only conceptual graphs without PROPOSITION concepts and without complex set referents, it is possible to associate a first-order logic formula ϕu with a conceptual graph u as follows [6]:

- A unary predicate is associated with each concept ci of u. The name of this predicate is the type label of the concept. The argument of this predicate is a variable symbol xi if the concept has no specified referent and the referent itself if it is a constant. The argument of the predicate is called the *identifier* of the concept ci.
- An n-ary predicate is associated with each n-ary conceptual relation r of u. The arguments of this predicate are the identifiers of the concepts linked by r in u.
- The logic formula associated with the graph u is

$\phi u = exist(x1, \cdots, xn)\, f.$

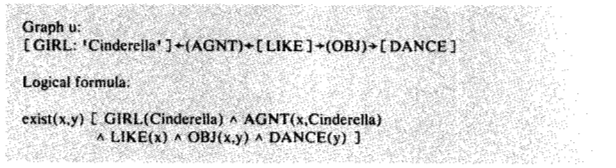

Figure 10

Logical interpretation of a simple conceptual graph.

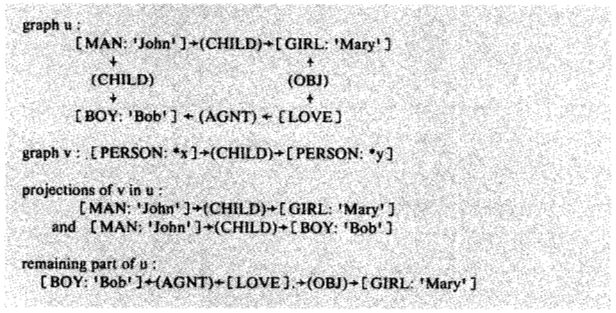

Figure 11

The projection operation.

The existentially quantified *xi* variables correspond to the concepts having no specified referents. The formula *f* is the conjunction of the predicates associated with the concepts and with the conceptual relations of *u*.

An example is given in **Figure 10**.

Sowa has extended this mapping into logic for higher-order conceptual relations on propositional concepts, for universal quantification, and for coreference problems. He uses an extension of Pierce's logical system adapted to conceptual graphs and gives the inference rules to perform deduction in Pierce's formalism. We do not use the same approach because the given set of inference rules, even if it is applicable by hand on little examples, seems too combinatorial to be used in practical systems. Therefore, we have studied the deduction method described in the last part of this section. This deduction method is based on the *matching* algorithm on conceptual graphs. The matching that we use is based on the projection operation which has been used by Sowa to formalize the join and the contraction operations. We now review this projection operation.

• *The projection algorithm*

The projection operation is closely related to the basic operation needed to introduce our Prolog-like deduction method for conceptual graphs.

The *projection* is defined as follows:

Let u and v be two conceptual graphs.

If $u \leq v$, there exists a subgraph u' of u isomorphic (modulo the restriction relation) to the graph v. The graph u can be viewed as the graph u' joined with some additional edges. The induced graph $u' = v \rightarrow u$ is called the *projection* of v in u. A graph v may have several distinct projections in the same given graph u, as shown in **Figure 11**.

If $u' = v \rightarrow u$ is a projection of v in u, we define the remaining part of u as the graph obtained by suppressing in u all the edges of u'.

The projection operation of v in u consists in exhibiting a subgraph u' of u which satisfies the following conditions:

1. The conceptual relations in u' and v are identical.
2. The concepts $c1, \cdots, cn$ of u' are some restrictions of the corresponding concepts $d1, \cdots, dn$ of v.
3. If a relation r links two concepts di and dj in v, then it also links the concepts ci and cj in u'.

Thus, we can define the *substitution* associated with the projection as the composition of the restrictions on the respective concepts of the two graphs. We note it by

$$\Theta = [(d1/c1), \cdots, (dn/cn)].$$

The structure of the implemented algorithm is given in **Figure 12**.

As mentioned, there are often many possible projections of a graph in another one, and the connectivity checking of the resulting graph is not included in Part 1 of the algorithm. In fact, we have implemented this algorithm as a nondeterministic Prolog predicate: If the obtained graph is not connected, a fail forces Prolog to backtrack inside the algorithm to find another solution.

• *The resolution principle for conceptual graphs*

We assume that we have

- A set of *clausal conceptual assertions* of the form

  G ← G1, G2, ···, Gn., where G, G1, ···, Gn are conceptual graphs.

  This clause can be interpreted as "to prove the assertion G, then try to prove the assertions G1, G2, ···, Gn.". If n = 0, the clause "G←." is a simple factual assertion.
- A "goal" clause of the form

  ← A1, A2, ···, An., where A1, ···, An are conceptual graphs.

  This goal clause will be interpreted as "prove A1 and A2 and ··· An."

The resolution mechanism is based on the following theorem, given in [6]:

If u and v are two conceptual graphs such that $u \leq v$, then $\phi u \supset \phi v$. (ϕu and ϕv are the logic formulas associated with u and v; $\supset$ is the logical implication.)

This result can be compared to the following implication of classical logic:

$$P(a) \wedge Q \supset exist\ (x)[P(x)],$$

where a is a particular constant. In our case, u is more specialized than the graph v. The graph u may contain more edges than v. Informally speaking, the subgraph u' of u, which is isomorphic (modulo restrictions on concepts) to v, plays the role of $P(a)$ in our analogy. The additional edges play the role of the other formulas Q included in the conjunction. Another point concerns the fact that the concepts of v will have more variables as referents than in the subgraph u'. Thus, informally, the logic formula corresponding to the graph v will be more "existentially quantified" than u. The preceding result depends also on the $\leq$ relation, which takes into account the lattice on the concept types.

Now, let us suppose that we have

a clause : G $\leftarrow$ G1, G2, $\cdots$, Gn
a goal : $\leftarrow$ A
a projection p of A in G.

Then G $\leq$ A, and therefore ϕG $\supset$ ϕA. This allows us to "erase" the goal A and to replace it with the subgoals

$\leftarrow$ G1, G2, $\cdots$, Gn.

The projection operation seems to be the basic algorithm to use in our deductive machinery. In fact, the operation that we need must be extended for the following reasons:

A Prolog goal $\leftarrow P(x)$. is not really interpreted as "prove the logical formula $exist(x)[P(x)]$" but is only a term to be instantiated by the solution for x found by the Prolog machinery. In the same way, we interpret a goal $\leftarrow$A. in our system as "try to find A or to exhibit a restriction of the graph A by deriving it from the set of the given assertions."

In other words, the concept types play the role of variables and the $\leq$ restriction relation between them plays the role of the instantiation. Thus, an instantiated solution of a "goal graph" in our system will be obtained via some restrictions on its concepts.

Suppose that a goal contains [PERSON:'John'] and that a clause head contains [BOY:*x].

The projection is too restrictive an operation because we would like to "unify" [PERSON:'John'] and [BOY: *x] in order to obtain [BOY:'John']. But these two concepts are not comparable by $\leq$ because of their referents, and therefore the definition of the projection cannot be applied.

To introduce a more general *matching* operation, we say that two concepts $c1$ and $c2$ are *compatible* if there exists a maximal nondegenerated common restriction $c3$ of $c1$ and $c2$ (i.e., $c3 \leq c1$, $c3 \leq c2$, and type(c3)$\neq$ABSURD).

```
-1-
IF v is reduced to a single concept d
  THEN find a concept c in u such that: c ≤ d ;
  ELSE
 FOR each occurrence of the relation r in v
 LOOP:
  IF r is not in u
   THEN fail
   ELSE B: suppress an occurrence of r in the set of relations occurring in u ;
    Let c1 and c2 be the concepts linked by r in u ,
        d1 and d2 the concepts linked by r in v ,
   IF c1 ≤ d1 and c2 ≤ d2
    THEN stack <r, c1, c2> in the result,
    ELSE Backtrack at the point B to find another
         occurrence of r in u.
 END LOOP

-2- Verify that the result stack is a connected conceptual graph.
  If it is not then backtrack inside -1-.

  The projection is in the result stack and
  the remaining part of u is the set of edges which
  remain in the graph u at the end of this process.
```

Figure 12

Projection algorithm.

This *matching* operation is defined as follows:

We say that a graph v can be matched to a graph u if there exists a subgraph u' of u such that

- The conceptual relations are the same in v and u'.
- If the concepts ci and cj (respectively, di and dj) are linked by the conceptual relation r in u (respectively in v), then, in the pairs (ci, di) and (cj, dj) the first and the second concepts must be compatible.

Using this definition, and if ei is the maximal restriction of ci and di, we obtain two substitutions on the graphs u and v:

$\Theta 1 = [(c1/e1), \cdots, (cn/en)]$ on the graph u,
$\Theta 2 = [(d1/e1), \cdots, (dn/en)]$ on the graph v.

Thus, a projection operation is a matching operation, but it is possible to match two graphs even if there is no projection between them, as in

[PERSON: 'John']$\leftarrow$(AGNT)$\leftarrow$[LIKE]$\rightarrow$(OBJ)$\rightarrow$
[ELEPHANT]$\rightarrow$(COLR)$\rightarrow$[GRAY]

and

[BOY: *x]$\leftarrow$(AGNT)$\leftarrow$[LIKE]$\rightarrow$(OBJ)$\rightarrow$
[ANIMAL: 'Jumbo']

This matching operation can be viewed as the operation used in the maximal join to build the common specialized overlap of two graphs, with the additional constraints that all the edges of the first graph must be included (modulo a restriction) in this overlap and that all joined concepts must be compatible.

```
Let P be the initial goal stack;
current goal stack ← P ;
loop while the current goal stack is not empty:
 Let "← A1,A2,...,Ap." be the current goal stack (conjunction of goals),
 Let "G ← G1,...,Gn." or "G ←" be a conceptual clause.
 current goal := "← A1",
 IF there exists a matching between G and A1
  THEN
   Let θ1 and θ2 be the substitutions associated respectively with G and A1
   by this matching;
   propagate these substitutions in the right side of the clause, if it is not
   empty (i.e., build θ1(G1),...,θ1(Gn) );
   propagate the substitutions in the remaining goal stack
   (i.e., build θ2(A2),...,θ2(Ap) ),
   and in P (i.e., build θ2(P) );
   new goal stack := ← θ1(G1), ... ,θ1(Gn),θ2(A2), ... ,θ2(Ap)
                or   ← θ2(A2), ... , θ2(Ap) if n=0 ;
end loop
result := stack P to which substitutions have been applied during the process.
```

Figure 13

Linear resolution algorithm for conceptual graphs.

```
A person is a citizen of Oz if and only if any of the following
conditions is true:
   1) This person is born in Oz.;
   2) One of his parents is a citizen of Oz;
   3) This person was naturalized in Oz;

clause (inference rule):
 [CITIZEN: *x]←(MEMB)←[COUNTRY: 'Oz']
     <= [PERSON: *x]←(AGNT)←[BORN]→(LOC)→[COUNTRY: 'Oz']

clause (inference rule):
 [CITIZEN: *x]←(MEMB)←[COUNTRY: 'Oz']
     <=    [PERSON: *x]←(CHLD)←[PERSON: *y]
         and [CITIZEN: *y]←(MEMB)←[COUNTRY: 'Oz'].

clause (inference rule):
[CITIZEN: *x]←(MEMB)←[COUNTRY: 'Oz']
<= [PERSON: *x]←(RCPT)←[NATURALIZE]→(LOC)→[COUNTRY: 'Oz']

clause (fact) :
 [PERSON: 'Tinman']←(AGNT)←[BORN]→(LOC)→[COUNTRY: 'Oz']
       +
   (CHLD)
       +
 [GIRL: 'Dorothy']


example of goal clause: "Who is a citizen of the Oz country ?" (P stack) :
     <= [PERSON]←(MEMB)←[COUNTRY: 'Oz'].

result : Two solutions (obtained by backtracking as in Prolog) :
    [CITIZEN: 'Tinman']←(MEMB)←[COUNTRY: 'Oz'].
    [CITIZEN: 'Dorothy']←(MEMB)←[COUNTRY: 'Oz'].
```

Figure 14

Example of resolution process.

This matching operation plays the role of unification in a Prolog interpreter.

• *A "top-down" resolution algorithm for conceptual graphs*

The preceding paragraphs allow us to define for the conceptual graph model a "top-down" linear resolution mechanism very similar to the Prolog mechanism (see **Figure 13**).

We give in **Figure 14** an example of deduction using our resolution method. We chose this "Oz" example because Sowa applied his set of inference rules to it in his book.

Conclusion

We had the feeling that the conceptual graph model could be a very powerful candidate as a core model of future systems integrating knowledge processing and natural language processing. Our practical experiments on the implementation of this model confirm this first feeling. The conceptual graph model is a general framework for expressing natural language semantics and designing very large semantic lexicons, but also a practical way to express a large amount of pragmatic information by assertions and clauses. All the algorithms are domain-independent and, unlike special-purpose knowledge models, the description of a semantic domain can be made through a purely declarative set of conceptual graphs.

Acknowledgment

We would like to thank John Sowa (IBM Systems Research Institute) for the fruitful contacts that we have had with him since we started to work on the conceptual graph model.

References

1. Charles J. Fillmore, "The Case for Case," *Universals in Linguistic Theory*, E. Bach and R. T. Harms, Eds., Holt, Rinehart & Winston, New York, 1968, pp. 1–88.
2. *Conceptual Information Processing*, Roger C. Schank, Ed., North-Holland Publishing Co., Amsterdam, 1975.
3. David R. Dowty, Robert E. Wall, and Stanley Peters, *Introduction to Montague Semantics*, D. Reidel Publishing Co., Dordrecht, the Netherlands, 1981.
4. Yorick A. Wilks, "Making Preferences More Active," *Artif. Intell.* **11**, 197–224 (1978).
5. Hans Kamp, "Events, Discourse Representations, and Temporal Reference," *Langages* **64**, 39–64 (1981).
6. John F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publishing Co., Reading, MA, 1984.
7. *VM/Programming in Logic (VM/Prolog)*, Order No. PO 5785-ABH, available through IBM branch offices.
8. Laurent Catach and Jean Fargues, "Deduction and Operations on the Conceptual Graph Model," *Research Report F087* (in French), IBM Paris Scientific Center, 1985.
9. Jean Pierre Adam and Jean Fargues, "Project KALIPSOS, General Overview," *Internal Research Report F074*, IBM Paris Scientific Center, 1984.
10. Alain Colmerauer, "Prolog in Ten Figures," *Proc. IJCAI-83*, pp. 487–499 (1983).
11. Michael C. McCord, "Using Slots and Modifiers in Logic Grammars for Natural Language," *Artif. Intell.* **18**, 327–367 (1982).
12. Robert C. Moore, "Problems in Logical Forms," *Technical Note 241*, SRI International, Menlo Park, CA, April 1981.
13. John F. Sowa, "Using a Lexicon of Canonical Graphs in a Conceptual Parser," *Workshop on the Lexicon, Parsing, and Semantic Interpretation*, CUNY Graduate Center, New York, January 1985. To be published.

Received July 25, 1985; revised August 9, 1985

Laurent Catach *Paris VI University, Artificial Intelligence Department, Paris, France.* Mr. Catach worked for IBM in 1985 as a visiting university student in the artificial intelligence group at the IBM Paris Scientific Center. He is a doctorate student at the Artificial Intelligence Department of the Paris VI University. Mr. Catach obtained his M.S. and his "Agregation," both in mathematics, from the Ecole Normale Supérieure.

Anne Elisabeth Dugourd *IBM France, Scientific Center, 36, Avenue Raymond Poincaré, 75116 Paris, France.* Ms. Dugourd is an engineer at the IBM Paris Scientific Center, where she has worked since joining IBM in 1982. She was a student of the Ecole Normale Supérieure, where she obtained her M.S. degree and her "Agregation," both in physics. She worked on the French Thesaurus linguistics project of the IBM Paris Scientific Center until 1984. Ms. Dugourd joined the artificial intelligence group at the beginning of 1985; she is working on the computational linguistics project of this group.

Jean Fargues *IBM France, Scientific Center, 36, Avenue Raymond Poincaré, 75116 Paris, France.* Dr. Fargues is a research engineer in the artificial intelligence group at the IBM Paris Scientific Center, where he has worked since joining IBM in 1983. He received the "Ingenieur" diploma from the I.D.N. French Engineering School in 1975, beginning work that same year in artificial intelligence as a research student at the Paris VI University. He obtained a thesis in 1978 for his work on the automatic synthesis of LISP recursive functions from examples. Dr. Fargues is "Docteur Es Science" in mathematics for his work on logic and deduction, defended as a doctoral thesis at the Paris VI University in 1983, supported by an IBM France research fellowship. His present interests include logic and computational linguistics, in connection with a project of the artificial intelligence group.

Marie-Claude Landau *IBM France, Scientific Center, 36, Avenue Raymond Poincaré, 75116 Paris, France.* Ms. Landau is an engineer in the artificial intelligence group at the IBM Paris Scientific Center. She joined IBM France in 1974. Ms. Landau received her degree of engineering from the Ecole Centrale des Arts et Manufactures, Paris, France, in 1970. From 1970 to 1974 she was employed by the Honeywell-Bull Company as an engineer in the systems architecture group in Paris. From 1974 to 1978 she worked at the IBM Development Laboratory, Essonnes, France, in the center of competence for VLSI design aids. Ms. Landau joined the Scientific Center in 1979. From 1979 to 1984 she worked in the image processing group, where she contributed to the IBM High-Level Image Processing System Program Offering (a software package to do image processing on an IBM 7350). Since 1984, she has worked in the artificial intelligence group at the Paris Scientific Center, where she contributed to the VM/Programming in Logic Program Offering. Ms. Landau is currently working on the computational linguistics project of the artificial intelligence group.