# Final Report: Active Knowledge Structures for Natural Language Processing

Yorick Wilks, Michael Coombs,
Roger T. Hartley, Dihong Qiu
Computing Research Laboratory
New Mexico State University
Las Cruces, NM 88003



**Abstract**

This report is the final one for contract N00014-89-J-1558. We present a reworking of the ideas in the ViewGen belief management system in terms of the structures and reasoning mechanisms contained in the Conceptual Programming system, CP which is based on Sowa's conceptual structures. Each of the constructs in ViewGen is shown in its CP form, and the major operations, ascription and percolation are shown as CP operations.

## 1 ViewGen and CP

ViewGen is a dynamic beliefs management system. It generates multiple belief environments from different points of view. The basic inference mechanism in ViewGen is default reasoning. That is, unless there is evidence to the contrary, the agents are assumed to have the same beliefs as the system. CP is a general knowledge representation system. It allows data, as *facts*, and domain knowledge, as *definitions*, to be separated, but it does not allow for nested beliefs. That is, CP only allows reasoning from one point of view, instead of allowing multiple viewpoints, as ViewGen does.

An integrated reasoning system has been designed which combines CP and ViewGen, and a prototype system has been implemented. Below is a discussion of the architecture of the system, and the representation and organization of belief and knowledge. First, we present an overview of the capabilities of Conceptual Programming and give a simple example of the representations it uses.

# 2 An overview of Conceptual Programming

The Conceptual Programming environment, CP, is an ongoing project at NMSU. The CP system is a knowledge representation development environment within a graphical visualization framework (for an overview, see [Pfeiffer and Hartley, in press], or [Hartley and Coombs, 90]). In the CP system all knowledge is represented by graphs and operations (mappings) performed over sets of those graphs. The CP representation presents an approach that uses a constructive technique based on the actual graphs displayed and their graph transformation operations [Hartley and Coombs, 89].

CP graphs are patterned after Sowa's conceptual structures ([Sowa et al, 90], [Sowa, 84]) using the operations defined for conceptual structures on the graphs. *Conceptual graphs*, as defined in Sowa's book, express declarative knowledge using concept, relation, and actor nodes, and link the total context together through the edges. CP also expresses declarative knowledge, but introduces a mechanism for expressing procedural knowledge through extended features for actors, both syntactically and semantically, where they perform more like "functional relations" ([Hartley and Pfeiffer, 91], [Pfeiffer and Hartley, 89], [Eshner and Hartley, 88]). One of our extension to Sowa's conceptual graph theory is the addition of an "overlay" level. This level allows for both feasibility-runtime domain support and spatio-temporal domain support. Within the feasibility-runtime domain, heuristics and constraints are introduced in the conceptual structures framework; whereas, within the spatio-temporal domain an ontology for objects, events, states and processes is provided within this same framework.

A simple example is the conceptual graph for the sentence *The girl took off a blue coat.* The graph is:

```
[TAKE-OFF] -
    (AGT) -> [GIRL:#]
    (PTNT) -> [COAT] -> (ATTR) -> [WET].
```

Details of the syntax of the linear representation of conceptual graphs can be found in Sowa's book (*op cit*, but briefly, concept labels are enclosed in square brackets, and relation labels in parentheses. The symbol '#' indicates a definite individual (corresponding to 'the' in the sentence). The arrows indicate the direction of the relation. Here the act 'take-off' has two conceptual cases, and agent (AGT) and a patient (PTNT). Wetness is an attribute (ATTR) of the coat.

An example of a problem solving system built on top of the CP environment has been developed here at NMSU. In this system, the process of solving a problem is one of constructing a CP graph, called a *model*, out of graphs that can be thought of as data, definitions and previously created models. Thus, at all times a partially completed model holds relevant data. The graph operations are used to create and update the actual models. Because solutions are found by generating models during the reasoning process, the general approach has been termed "Model Generative Reasoning, or MGR".

# 3 CP Representation

## 3.1 Basis of the representation

CP is a knowledge representation environment visualized and operated on through graphs. These graphs are implementations of Sowa's conceptual structures ([Sowa et al, 90], [Sowa, 84]) and retain many of the features of conceptual graph theory. Although there exists a mapping from conceptual graphs to formulae in first-order predicate calculus, FOPC, the operations used in the CP system take advantage of the graphical representation. We therefore study the structure and operations on the graphs using graph theory [Harary, 69] instead of FOPC.

A *conceptual structure CS* is defined as a connected multilabeled bipartite oriented graph [Eshner and Pfeiffer, 90]. The two colors of nodes in a conceptual structure are called *concept* and *relation*. Each label in a concept node consists of two fields, the *type* field and the *referent* field. The type field is an element of the set of concepts defined in a type lattice ([Eshner and Pfeiffer, 90], [Pfeiffer and Hartley, 90], [Sowa, 84]). The referent field contains the individual specialization (if any) for the type field. Each label in a relation node consist of the single *relation* field. This relation field depicts the relationship between the adjoining concept nodes within the conceptual structure.

## 3.2 Extensions to basic representation

Sowa has shown how unknown objects (nodes with no individual field) can be computed by an *actor* node that corresponds to a function in standard logics. This actor can best be thought of as a "functional relation", where there is a semantics (performed by the procedure) being represented graphically between two objects. Actor nodes of this kind are diamond-shaped boxes connected to concept nodes with dashed lines. In our extensions to conceptual graph theory, these actors are given the capability of computing 1) *quantitative* constraints in a Prolog fashion (i.e. of doing constraint propagation through a system of values and variables), and 2) *qualitative* constraints that propagate among moments in time when acts occur, and locations of objects in space. This inspired the overlays in CP. Overlays are seen as a new level of graphs that live on top of the basic graph definitions. The analogy here is of overhead slides being laid on top of one another to produce a complete diagram.

Within this level, there are actually two sublevels, *feasibility-runtime* domain, and *spatio-temporal* domain. The feasibility-runtime sublevel uses quantitative actors as described above. The spatio-temporal level uses qualitative actors. This level requires a syntactic extension to conceptual graphs in order that the diagrams not become too confused and thus lose their force.

The feasibility-runtime sublevel provides CP a mechanism for a system of constraints. The feasibility overlay works as a "heuristic" at a 'compile time' level of computation as opposed to a 'runtime' level. The graphs are created as overlays to a particular definitional graph. Each overlay contains at least one or more actors. These actors have a functional procedure associated with them and this function is executed when the actor is evaluated during the join operation. Runtime overlays work as "constraints" that imple-

ment Sowa's original actors in conceptual graphs with two additions: 1) each actor may behave like a formal constraint on a state or concept referent as well as a function, and 2) constraint actors may take as input a state as well as a single concept referent. Like feasibility heuristics, runtime constraints focus on the quantitative functional relations. For feasibility-runtime actors these inputs and outputs may be any relevant concept or relation in the graph [Pfeiffer and Hartley, 89].

Within the spatio-temporal sublevel, the main aim is at determining what things can be inputs and outputs to spatio-temporal actors. Three concept types are focused on: objects, acts and properties. The relationships between these three entities types are the basic inputs and outputs to the spatio-temporal actors ([Hartley and Pfeiffer, 91], [Eshner and Hartley, 88]). These relationships can be made explicit by interpreting the constraints expressed by the actor and its connections (inputs and outputs, roughly speaking) just as a rule in a rule-based system can be thought of as an implicit relation between its left and right-hand sides. 'Firing' the rule computes the relation.

Rules in CP are represented at this spatio-temporal level. The actors sole job is to act as confluence points for the knowledge structures that have to be related. All of the actors are constraint-like in that they can operate forwards or backwards. However, temporal actors are often regarded as operating forwards, in the direction of time. Thus, inputs to an actor are pre-conditions for the actor's firing, and outputs are post-conditions. In the temporal domain, inputs are partial states and schematics, since these are exactly what is expected to change in time. Each temporal actor also has an act (really a process) as input. With the crucial part of the whole idea within the temporal relationships being how the act relates to the inputs and outputs. The input and output relations are represented by a *time chart* [Hartley, in press]. This time chart is similar in use and meaning to the time maps of Dean and McDermott [Dean and McDermott, 87] and have correspondence to Allen's relations [Allen, 85].

We can apply the same notions to create spatial actors corresponding to the temporal actors just discussed. Where the temporal overlay placed partial states or schematics in temporal relationship, the spatial one places partial processes or chronicles in spatial relationships. Each spatial rule will be represented by an actor corresponding to an object. Whereas the temporal actors are directional, according to the forward flow of time, there is no such constraint on spatial actors. Through the use of spatial and temporal actors, CP is able to operate over space and time.

# 4 The architecture of the integrated system

The architecture of the integrated system contains two components (see Figure 1). One component corresponds to the reasoning engine based on CP, and the other corresponds to ViewGen. The basic strategy is to use Viewgen as a "bookkeeping" system, rather like the ATMS, to maintain the current state of belief space and to generate the new nested belief environment according to need. The problem solver, MGR, can be used in any belief environment to generate new assertions or hypotheses from old ones. The generated hypotheses are

beliefs in that environment. These beliefs can be propagated by ascription and percolation, the operations of ViewGen.

There are three kinds of inference in the current system: ascription, percolation, abduction. Each of them is an independent inference engine. All of them operate on nested belief environments. The initial belief environment can be built by the user or by the inference engine. The function of the ascription inference engine is to build the initial or new nested belief environments. The abduction inference engine is a problem solver; it also can be used in building the initial belief environments. The percolation inference engine propagates the hypotheses generated by abductive inference from the inside of a nested belief environment to the outside. In the the current version of ViewGen, there is no percolation inference, so this is an addition. In the future, we can build other inference engines as well, such as deduction or inheritance inference. In principle any problem solving or inference technique can be incorporated into the ViewGen superstructure as needed.

# 5 The representation of belief and knowledge

In Viewgen, a belief is represented by the language known as FOLSE. (First Order Logic with Sets and Environments) [Ballim, 86]. In the integrated system, both belief and knowledge are represented through Conceptual Programming (CP) structures [Hartley and Coombs, 88]. Following, we will discuss how to use conceptual graphs in belief ascription and reasoning.

## 5.1 The representation of beliefs in CP

There are three kinds of objects in ViewGen: simple belief , environment, and atypical belief. We will discuss each briefly.

### 5.1.1 Simple belief

A simple belief is represented by first-order logic predicates in FOLSE. Because there is a simple mapping from first-order to conceptual graph [Sowa, 84], it is easy to represent a simple belief by a conceptual graph. For example, the belief that the earth is round can be expressed as:

```
[EARTH] -> (CHRC) -> [SHAPE] -> (ATTR) -> [ROUND]
```

### 5.1.2 Environment

In order to express the environments of ViewGen , we introduce the "environment" concept type . The type label is ENVIRONMENT, and its referents are one or more graphs. It is one kind of "context", or embedded graph of which GRAPH, PROPOSITION, and SITUATION are others. They are all subtypes of universal type T (top). For example, we can represent "John believes the earth is flat" as following:

```
[ ENVIRONMENT:
    [PERSON:John] <- (EXPR) <- [BELIEVE] -> (PTNT) -
        -> [PROPOSITION: [EARTH] -> (ATTR) -> [FLAT]]]
```

Here, the proposition represents a simple belief. In fact, we can express nested beliefs too. Following is the representation of belief "The system believes the earth is round and believes John believes the earth is flat" .

```
[[SYSTEM] -
    (BELIEVE-ABOUT) -> [EARTH] -> (STMT) -> [[EARTH] -> (ATTR) -> [ROUND]]
    (BELIEVE-OF) -> [PERSON:John] -
                        (BELIEVE-ABOUT)-
                            -> [EARTH] -> (STMT) -> [[EARTH] -> (ATTR) -> [FLAT]]]
```

Here, we have distinguished the conceptual relations "BELIEVE-ABOUT" and "BELIEVE-OF". This is because environments consist of two basic types , viewpoints and topics. See the ViewGen papers (*op cit*) for detailed discussion. From the above description, we can get the general form of environment:

```
[ENVIRONMENT:
    [SYSTEM] -> (BELIEVE-ABOUT) -> [TOPICS1] -> (STMT)-
                [PROP: {G11, G12, ...}]
             -> (BELIEVE-OF) -> [AGENT1] -> (BELIEVE-ABOUT) ->
                [TOPICS2] -> (STMT) -> [PROP: {G21, G22, ...}]
            ... ...
             -> (BELIEVE-OF) -> [AGENT1] -> (BELIEVE-OF)-
              ...  -> [AGENT N] -> (BELIEVE-ABOUT] -> [TOPICS N]-
               -> (STMT) -> [PROP: {GN1, Gn2, ...}]
]
```

### 5.1.3 Atypical belief

The atypical belief in ViewGen is represented by lambda-expression and lambda formulas. Because there is a lambda expression mechanism in conceptual graphs, again the correspondence is clear. For example, "L's phone number" can be written as:

```
    (LAMBDA L) [PERSON:*L] -> (POSS) -> [PHONE] -> (ATTR) -> [NUMBER:#]
OR
  PHONE-NUMBER-OF L =
    (LAMBDA L) [PERSON:*L] -> (POSS) -> [PHONE] -> (ATTR) ->
[NUMBER:#].
```

The $\lambda$-functions in ViewGen consist of one basic evaluation relation and three complex evaluations: the basic evaluation "Val-for" expresses the agent's beliefs about the value of $\lambda$-expression; the complex relation "Comp(X,Y)" expresses the fact that there are two

competing evaluation relations X and Y for the expression; "Spec(X,Y)" expresses that the evaluations derived from the evaluation relation X are considered "better" by those agents involved in X than the evaluations derived from Y. In addition , it indicates that the agents involved in Y do not believe the agents involved in X to have such "better" values; "SpecK(X,Y)" which is the same as "Spec(X,Y)" except that it states that the agents involved in Y believe that the agents involved in X have a better evaluation, although they don't know the actual evaluation). We can define the above evaluation relations in a similar fashion to the above example.

We can then give a representation for a specialist belief:

Assume there is a lambda-expression "(LAMBDA L) immed-type-of L" and Z = ((LAMBDA L) immed-type-of L) (thalassemia), we can describe the relevant lambda formula as:

```
[ [[SPECIALIST] -> (VAL-FOR) -> [HYPOCHROMIC]]-
   -> (SPECK) ->
  [ [[HIGH-MED-INFORM-PERSON] -> (VAL-FOR) -> [GENETIC-DISORDER]]-
     -> (SPECK) ->
    [[AVG-MED-INFORM-PERSON] -> (VAL-FOR) -> [DISEASE]]-
  ]
]
 -> (SPEC) ->

[ [[AVG-EDUCATED-PERSON] -> (VAL-FOR) -> [GREEK-PROVINCE]]-
   -> (SPEC) ->
  [[AVG-PERSON] -> (VAL-FOR) -> [GREEK-PLACE]]
]
```

This expresses the various levels of specialized knowledge of thalassemia, depending on the type of person. Medically trained people believe it to be a genetic disorder; people with some medical knowledge believe it to be a disease, and everybody else thinks it is something in Greece.

## 5.2 Belief Ascription

The detailed strategies of belief ascription have been discussed by Wilks and Ballim in their book "Artificial Believers" [Wilks and Ballim, 91]. Here, we only discuss how to ascribe belief by using conceptual graphs, that is, to implement ViewGen structures in conceptual graphs. Generally, there are three main tasks in ViewGen: setting up an initial environment, transforming the environment, and pushing one environment inside another.

### 5.2.1 Generating the initial environment

In order to set up the initial environment , there are two subtasks we need to execute: to decompose the goal environment and match the existing environment.

The decomposition entails generating relevant degenerate environments and arranging them according in preferred order. After decomposing, we have:

$$Gf \rightarrow Gf_1, Gf_2, \ldots, Gf_i$$

By matching the above $Gf_k (k = 1, \ldots, i)$ with the environment description, we can find the corresponding content of each degenerate environment and set up the initial environment. Here, by using conceptual graphs to represent environments, the change in criteria only changes the number and the order of subgraphs $Gf_k$, it has little effect on the whole system.

### 5.2.2 Transformation

As we have discussed above, ViewGen uses lambda expressions and lambda formulae to express an atypical belief. Generally, ascribing a $\lambda$-formula to an agent requires altering the formula, which involves changing the function table for the formula. In their book, Wilks and Ballim have discussed the transformation of $\lambda$-formula and give a recursive transformation function. However, because there are only four basic evaluation relations defined in $\lambda$-formula, the transformation function is restricted in its use. Because CP has a mechanism for defining different evaluation relations, we can generalize the transformation function by using the power of conceptual graphs.

By checking the transformation condition and the agent mentioned, we can transform the basic belief function. However, complex functions need a special method of transformation. Wilks and Ballim have given the transformation method for some complex evaluation relations. In conceptual graphs it is possible to define new relations, and an associated transformation method. When ascribing a complex relation, the system decides whether or not the relation is defined by the user. If it is, the system finds the corresponding transforming method and changes the relation. Otherwise, the system transforms the relation in pre-set ways.

### 5.2.3 Pushing one environment into another

After we have set up the initial environments and transformed them regarding to the agents, now the task is to push the content of degenerate environments and produce the final environment. The main problem in pushing a belief into an environment is finding the counter evidence, that is, finding whether inconsistencies between source environment and target environment exist. Ballim adopted a method which is like de Kleer's ATMS methodology. By using justification and assumption sets and keeping a record of propositions that are inconsistent with each other, the system can determine whether a new proposition is valid by checking that there are no inconsistent pairs in the assumption set .

Using the above method, we can find the inconsistency between "the earth is flat" and "the earth is not flat". But, for two propositions that have no surface inconsistency, such as "the earth is flat " and "the earth is round" , it is necessary to identify the mutually exclusive properties between flat and round in order to find the inconsistency. By using the canonicality of conceptual graphs and the knowledge of the conformity of individuals to

types in conceptual graph, inconsistency between propositions can be found. In addition, we plan to use CP's constraint actor mechanisms to perform this sorts of consistency checks.

## 5.3 A Conceptual Graph Based Language for Belief Ascription and Reasoning

In [Wilks and Ballim, 91], Wilks and Ballim define two kinds of language: $L_{int}$ and $L_{ext}$. The former is one kind of three-value logic, and the latter is a representation method , incorporating the notion of viewpoints. Because the two languages are different, when we build a system for both belief ascription and reasoning, confusion is possible and they can be difficult to implement. Here, we try to formally define a unified conceptual graph based language for both belief ascription and belief reasoning.

Because $L_{int}$, $L_{ext}$, and FOLSE (the language used by Viewgen) are based on first-order logic, and there is simple corresponding relation between first-order logic and conceptual graph, we can define a unified language based on conceptual graphs.

### 5.3.1 Representation language BRL

Definition: If u is an atomic conceptual graph, that is, u has no nested contexts, $\neg$ is the (two-valued) negative relation, and @ is the unknown relation, which is a monadic operator that provides a mapping from three-value logic to two-value logic, then:

1. $\neg u$, @u, $\neg$ @u are atomic wffs of BRL.

2. If $\alpha$ is an agent, and v is an atomic wff as in 1, then $[\alpha] \rightarrow (Bel) \rightarrow [v]$ is an atomic belief wff of BRL.

3. If $\alpha$ is an agent and u is a belief wff, then $[\alpha] \rightarrow (Bel) \rightarrow [u]$ is a belief wff of BRL, and if v is also a wff of BRL , then $[\alpha] \rightarrow (Bel) \rightarrow [\{u, v\}]$ is a belief wff of BRL.

Here, the form $[\alpha] \rightarrow (Bel) \rightarrow [u]$ is a contraction of the description:

$$[\alpha] \leftarrow (EXPR) \leftarrow [BELIEVE] \rightarrow (PTNT) \rightarrow [PROPOSITION : u]$$

By this definition, we have defined a language for viewpoints.

As in language $L_{ext}$ [Ballim, 86], we can define contradictoriness and contrariness between viewpoints in this language.

Definition: p, q are atomic conceptual graphs. If p is not of the form @q nor of the form $\neg$@q ($p \neq q$) then p and $\neg$p are said to be contradictory across viewpoints.

Definition: p, q are atomic conceptual graphs. If p is of the form @q or of the form $\neg$@q, then p and $\neg$p are said to be contrary across viewpoints.

Definition: U is a set of belief wffs in BRL. If the agent in the outermost nested context is System, and th System believes all the members of U, then U is known as an environment.

Definition: U is environment. P is set of atomic wffs in the same nested context. If no p and $\neg p$ are both in P, then environment U is consistent.

Definition: U, V are environments, p,q are atomic wffs in them respectively. If U,V are consistent, and no contradictory between p,q, then U, V are consistent under ascription, denoted by A(U,V).

### 5.3.2 Belief Ascription

Before discussing the detail mechanism of belief ascription, we will define the basic idea of ascription first.

Basic Ascription rule:

Let u, w are atomic belief wffs as following

$$[\alpha] \rightarrow (Bel) \rightarrow [p]$$

$$[\beta] \rightarrow (Bel) \rightarrow [q]$$

If U,W are consistent under ascription (U contains p, and W contains q, and p and q are consistent), then

$$\underline{[\alpha] \rightarrow (Bel) \rightarrow [p], [\beta] \rightarrow (Bel) \rightarrow [q], A(U,W)}$$

$$[\alpha] \rightarrow (Bel) \rightarrow [\beta] \rightarrow (Bel) \rightarrow [\{p,q\}]$$

This is the simple rule for belief ascription. In fact, ascription is a more complex operation because of the checks for consistency and handling of special knowledge. For a detailed discussion see [Wilks and Ballim, 91].

### 5.3.3 Belief percolation

Definition: U,V are two belief environments, $p_i$ are atomic wffs in $U, i = 1, \ldots, m$, $p'_j$ are atomic wffs in $V, j = 1, \ldots, l$. U,V are said to be assertively consistent if for any $i$ and $j$, $p_i$ does not contradict any $p'_j$ .

Assume U,V are two belief environments, $\alpha_0, \ldots, \alpha_{n-1}$ are the agents in U, and $\alpha_0, \ldots, \alpha_{n-1}$, $\alpha_n$ are the agents in V. $p_i$ are atomic wffs in the innermost context of $U, i = 1, \ldots, m$ $p'_j$ are atomic wffs in the innermost context of $V, j = 1, \ldots, l$. If there is a $p'_k$ in the innermost context of V which is not of the form $\textbf{@}p$, nor of the form $\neg\textbf{@}p$, $\neg p'_k$ is not in $\{p_i\}$, and $p'_k$ is assertively consistent with $\{p_i\}$, then we can percolate $p'_k$ as follows:

$$U : [System] \rightarrow (Bel) \rightarrow \ldots [\alpha_{n-1}] \rightarrow$$

$$[Topic] \rightarrow (STMT) \rightarrow [\{p_1, \ldots, p_m\}]$$

$$V : [System] \rightarrow (Bel) \rightarrow \ldots [\alpha_n] \rightarrow$$

$$\underline{[Topic] \rightarrow (STMT) \rightarrow [\{p'_1, \ldots, p'_l\}]}$$

$$W : [System] \rightarrow (Bel) \rightarrow \ldots [\alpha_{n-1}] \rightarrow$$

$$[Topic] \rightarrow (STMT) \rightarrow [\{p_1, \ldots, p_m, p'_k\}]$$

# 6 The organization of beliefs and knowledge

In the prototype system, we used the knowledge structures of Viewgen and CP. We use a simple two-level table to represent the nested belief environments of agents. Each element in the first level table is a pair (Agents content). Agents is a list of agents. The list represents the nested relation among the agents. For example, (System Mary Susan) represents "System BELIEVE-OF Mary BELIEVE-OF Susan BELIEVE-ABOUT..." Content is list of pairs. For each pair, the first element represents the topic, the second element is the real beliefs under the topic. Those beliefs are divided into three categories: facts (assertions assumed to be true), definitions (domain knowledge), and models (hypotheses derived from facts and definitions). Thus, the belief environment looks like:

```
( (( system) ((T1)((facts (f1 ...))
                   (definitions (d1...))
                   (models (m1...))
                  )
              (T2)((facts (f2...))
                   (definitions (d2...))
                   (models (m2...))
                  )
              ))

  ((system agent1) ((T1) ((facts (f11...))
                          (definitions (d11...))
                          (models (m11...))
                         )
                   ))
)
```

In the prototype system, all beliefs, including facts, definitions, and models are represented by CP graphs. We only put the name of the graph in the two-level table. All of these graphs are stored in a structured knowledge base so that ascription and percolation inferences can be made as needed.

# 7 The Implementation of Ascription and Percolation

Here we briefly sketch the mechanisms that implement the two main ViewGen operations. They are both based on the relevant CP operations, making the integration of ViewGen and CP even closer than simply using CP's style of representation.

## 7.1 The implementation of Ascription

As described above, ascription is a copy operation with restrictions. Since each nested environment in ViewGen is internally consistent (much like an ATMS environment), then

any new assertion must also be assertively consistent with the set, as described above. CP can check for *canonicality* which is essentially type consistency through the *maximal join* operation. A new assertion is thus joined to the set of assertions in an environment. If the join succeeds, then the assertion is added. If the join fails (either through some type incompatibility, or through a negation check) then the assertion is not added. Further, more semantic checks, such as necessary with specialized knowledge can be made through CP's feasibility constraints (see [Pfeiffer and Hartley, in press]) but lack of space prevents any further discussion.

## 7.2 The implementation of percolation

Percolation is the dual of ascription in that it promotes assertions from inner to outer nested environments. The dual of maximal join in CP is *maximal project.* It also preserves canonicality, but in addition is truth preserving. Thus percolation is guaranteed to only promote true assertions, whereas ascription can produce assertions that are only abductively true. Again constraints can modify percolation just is they can be used with ascription.

For a simple example of these mechanisms, refer to Figure 2. Here the system is diagnosing a disease presented by a patient Tp. The system is assumed to know facts s_fact1 and s_fact2, and observes symptoms s_obs1, s_obs2 and s_obs3. Beacuse this is insufficient to diagnose the disease, the system searches for expert advice, and finds that two doctors, A and B, can diagnose the disease, but at the moment the observations needed for diagnosis are unknown. The system believes that Doctor A knows A_fact1 and A_fact2, and believes hypotheses A_hyp1 and A_hyp2 about the disease in question. Correspondingly, the system holds beliefs about Doctor B's beliefs, B_fact1 and B_hyp1. Following is the initial environment, described in BRL:

$$[SYSTEM] \leftarrow [Tp] \rightarrow (STMT)-$$
$$\rightarrow [\{s\_fact1, s\_fact2, s\_obs1, s\_obs2, s\_obs3\}]$$
$$[SYSTEM] \leftarrow (BEL) \leftarrow [DrA] \leftarrow (BEL)-$$
$$\leftarrow [Tp] \rightarrow (STMT) \rightarrow [\{A\_fact1, A\_fact2, A\_hyp1, A\_hyp2\}]$$
$$[SYSTEM] \leftarrow (BEL) \leftarrow [DrA] \leftarrow (BEL)-$$
$$\leftarrow [DrB] \rightarrow (STMT) \rightarrow [\{B\_fact1, B\_hyp1\}]$$

Through ascription inference we can push s_fact1, s_fact2, s_obs1, s_obs2, s_obs3 into Dr A's environment. Suppose now that s_fact2 is insonsistent with A_fact1 and A_fact2. A'a nested belief environment is then:

$$[SYSTEM] \leftarrow [DrA] \leftarrow (BEL)-$$
$$\leftarrow [Tp] \rightarrow (STMT) \rightarrow [\{s\_fact1, s\_obs1, s\_obs2, s\_obs3, A\_fact1, A\_fact2, A\_hyp1, A\_hyp2\}]$$

Similarly, for Dr B, if s_fact1 is inconsistent with B_fact1, we get:

$$[SYSTEM] \leftarrow (BEL) \leftarrow [DrA] \leftarrow (BEL)-$$
$$\leftarrow [DrB] \rightarrow (STMT) \rightarrow [\{s\_fact2, s\_obs1, s\_obs2, s\_obs3, B\_fact1, B\_hyp1\}]$$

By using abduction, Dr A believes that A_hyp1 is a possible explanation for these observations. Dr B believes that B_hyp1 is an explanation. Suppose that A_hyp1 and B_hyp1 are not inconsistent. These hypotheses may then percolate to the system's environment, resulting in:

$$[SYSTEM] \leftarrow (BEL)[Tp] \rightarrow (STMT)-$$
$$\rightarrow [\{s\_fact1, s\_fact2, s\_obs1, s\_obs2, s\_obs3, A\_hyp1, B\_hyp1\}]$$

Thus the system can use A_hyp1 and B_hyp1 as alternative diagnoses for the patient's disease.

# 8 Conclusion

We have shown how the essential structures in ViewGen, i.e. nested belief environments, can be represented in CP's graph language. In addition, the major operations for reasoning about nested beliefs can be implemented with CP's graph operations.

We have implemented a prototype of the integrated system that extends the single-environment type of reasoning found in MGR to ViewGen's multiple environments. This prototype, that shows how modern military intelligence might have been used in the English civil war ([Coombs and Hartley, 88], is implemented in CommonLisp, and runs on UNIX workstations. In the partially fictitious scenario, the system reasons over the King's beliefs about his opponent's actions, and vice versa.

# References


[Allen, 85] J. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the Association of Computing Machinery*, 26(11), pp. 832-843, 1985.

[Ballim, 86] Ballim,A. 1986, A proposed language for representing and reasoning with nested beliefs, MCCS-86-77, CRL, New Mexico State University.

[Coombs and Hartley, 88] Coombs,M.J. and Hartley,R.T. 1988, Design a software environment for tactical situation development, MCCS-88-144, CRL, New Mexico State University.

[Dean and McDermott, 87] T.L. Dean, and D.V. McDermott, "Temporal data base management," *Artificial Intelligence*, 32, pp. 1-55, 1987.

[Eshner and Hartley, 88] D.P. Eshner, and R.T. Hartley, "Conceptual Programming with Constraints," *Proc. of the Third Annual Workshop on Conceptual Structures*, Minneapolis, pp. 3.1.2-1 - 3.1.2-6, 1988.

[Eshner and Pfeiffer, 90] D.P. Eshner, and H.D. Pfeiffer, "A Graph Theoretic Basis For Problem Solving," *Fifth Rocky Mountain Conference On Artificial Intelligence (RMCAI-90)*, Las Cruces, pp. 161-166, 1990.

[Harary, 69] F. Harary, *Graph Theory*, Reading, MA: Addison-Wesley, 1969.

[Hartley and Coombs, 88] Hartley, R.T. and Coombs,M.J. 1988, Conceptual Programming: Foundations of problem-solving, MCCS-88-129.

[Hartley and Coombs, 89] R.T. Hartley, and M.J. Coombs, "Reasoning with Graph Operations," *Proceedings of Workshop on Formal aspects of Semantic Networks*, J.F. Sowa (Ed), *Formal Aspects of Semantic Networks*, New York, NY: Addison-Wesley, 1989.

[Hartley and Coombs, 90] R.T. Hartley, and M.J. Coombs, "Conceptual programming: Foundations of problem solving," J.F. Sowa, N.Y. Foo, and A.S. Rao (Eds), *Conceptual Graphs for Knowledge Systems*, New York, NY: Addison-Wesley, 1990.

[Hartley, in press] R.T. Hartley, "A Uniform Representation for Time and Space and their Mutual Constraints," special edition on Semantic Networks in Artificial Intelligence (ed. F. Lehmann), *Computers and Mathematics with Applications*, in press.

[Pfeiffer and Hartley, 89] H.D. Pfeiffer, and R.T. Hartley, "Semantic additions to Conceptual Programming," *Proc. of the Fourth Annual Workshop on Conceptual Structures*, Detroit, pp. 6.07-1 - 6.07-8, 1989.

[Pfeiffer and Hartley, 90] H.D. Pfeiffer, and R.T. Hartley, "Additions for SET Representation and Processing to Conceptual Programming," *Proc. of the Fifth Annual Workshop on Conceptual Structures*, Boston&Stockholm, pp. 131-140, 1990.

[Hartley and Pfeiffer, 91] H.D. Pfeiffer, and R.T. Hartley, "The Conceptual Programming Environment, CP: Time, Space and Heuristic Constraints," *Proc. of the Sixth Annual Workshop on Conceptual Graphs*, Binghamton, pp. 331-342, 1991.

[Pfeiffer and Hartley, in press] H.D. Pfeiffer, and R.T. Hartley, "The Conceptual Programming Environment, CP," J. Nagle, T. Nagle, L. Gerholz, and P. Eklund (Eds), *Current Research using Conceptual Structures*, Heidelberg, W. Germany: Springer-Verlag, in press.

[Qiu, 91] Qiu,D. 1991, Using conceptual graph as a unified belief representation approach in ViewGen system, ICYCS'91, Beijing, China.

[Sowa, 84] J.F. Sowa, *Conceptual Structures*, Reading, MA: Addison Wesley, 1984.

[Sowa et al, 90] J.F. Sowa, N.Y. Foo, and A.S. Rao, *Conceptual Graphs for Knowledge Systems*, New York, NY: Addison Wesley, 1990.

[Wilks and Ballim, 91] Wilks,Y. and Ballim,A. 1991, Artificial Believers, Lawrence A Erlbaum Associates, Hillsdale, NJ, 1991.

[Wilks and Hartley, 90] Wilks,Y. and Hartley,R.T. 1990, Belief ascription and model generative reasoning: joining two paradigms to a robust parser of messages. MCCS-90-180, New Mexico State University.

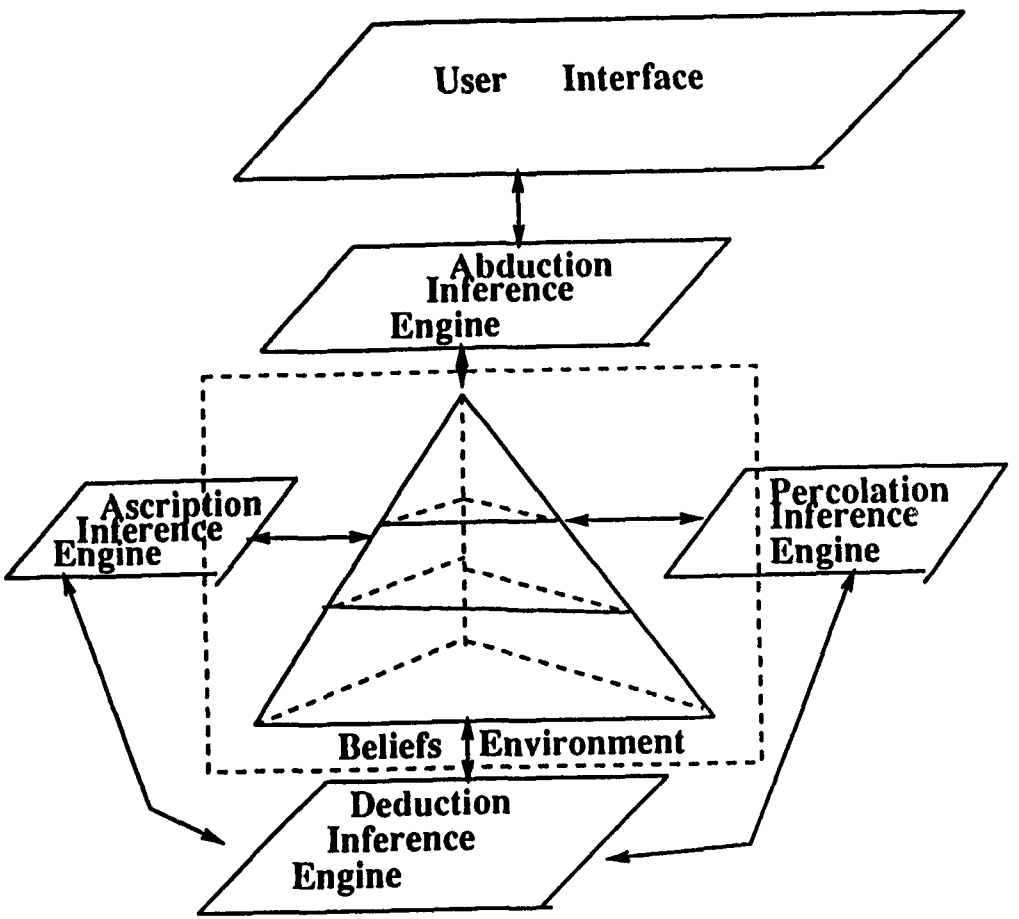


**Figure 1** Architecture for integrated system

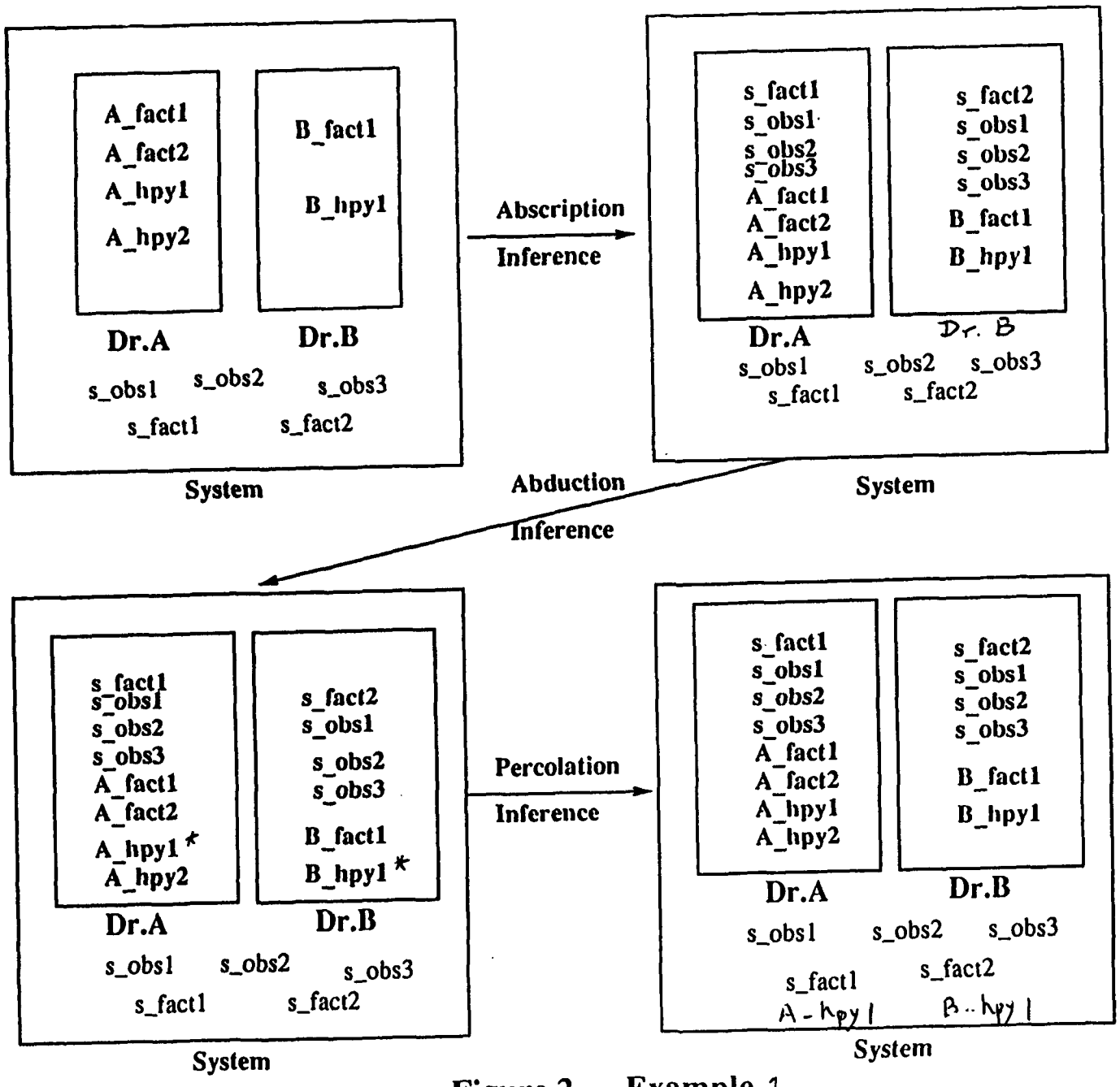


Figure 2 Example 1