An Overview of Conceptual Programming

Roger T. Hartley

Computer Science Department, New -Mexico State University

Abstract

Conceptual Programming is a term meant to convey a similar idea to that of Logic Programming, but at a higher level of representation. Programming with concepts, as presented here, has all the advantages that many knowledge representation schemes have in dealing with declarative knowledge i.e. explicitness, naturalness, expressibility, and transparency. It also gives procedural knowledge these same features, using the same representational technology, thus providing uniformity of expression for both forms of knowledge. This paper presents an overview of an augmented version of John Sowa's conceptual graph theory; a recipe of neccessary items for building a conceptual programming system; and an outline of the actions of an interpreter for the system. Conceptual programming is intended for work in knowledge engineering. A case is made, however, for its inclusion in the repertoire of general-purpose computing language systems.

Introduction

Conceptual programming is a term meant to convey a similar idea to that of logic programming, but at a higher level of representation. One of the differences between the two concepts (a recurringtheme in this paper) is that whereas logic programming represents knowledge at a logical (or at best at an epistemological level, as described by Brachman, 1979), conceptual proaramming represents it at a higher level. The metric here is similar to that used to compare all computer languages i.e. high and low level, but the issues are different. All logical level languages (including all general purpose computer languages) impose a rigid systematization on the programmer. Conceptual programming (CP), is less of a straight-jacket when it comes to translating ideas into programs. However, it must, if it is to be a regular language, provide some framework for the programmer. This framework is the subject of this paper.

The other major difference between CP and logic programming (referred to generically by 'Prolog') is that CP computes in open worlds, whereas Prolog computes in closed worlds. Thus although CP allows definition of typed concepts and follows a kind of proof procedure, it is not just typed Prolog. Open world computations (Hewitt, 1985) are fundamentally different from their closed world cousins and allow for differing interpretations of terms used, handling of incompleteness and inconsistency, None of these are allowed in Prolog or the standard general-purpose Ianguages. Failure to acknowledge these issues has led to the brittleness of expert systems (Buchanan, 1982). Such systems fail to say anything sensible when pushed beyond the limits of their domain.

The advantages to be claimed for CP over conventional programming systems are a more natural transition between problem statement and working program; a more integrated representation of procedural knowledge; and a tolerance to inconsistency in definition and use of terms. These claims, if vindicated, will make CP a better tool for the real-world engineering of expert systems.

The following issues are addressed:

- to investigate the nature of open-world computation and to design a methodology to make it viable in a computer svstem
- to implement an experimental prototype for conceptual programming
- to investigate the application of CP to expert system problems

CP as a computer language

During the history of computing, attention has focused from time to time on the so called *computational paradigim.* This is an attempt to capture, in a sentence. not

only the nature of computation, but also the ways in which it is embodied in real language systems. Four main paradigms may be discerned in currently used sys-

- assignment of values to a memory, typified in many old' languages and the Algol, Pascal, Ada, Modula group.
- application of a function to its arguments, exemplified in Lisp and its variants.
- response to messages passed between objects, as in the Simula, Smalltalk, Flavors group.
- proving- a theorem by displaying a model for it, as embodied in Prolog

All of these paradigms are logical-level i.e. their semantics are only to be understood in abstract terms. There is no *necessary* connection with the real world which they are abstracting and representing. CP, however, is tied to the world both through its use of terms and through its methodology of accepting new terms. The principle followed may be called *propositional coherence.* Propositions are considered to be the conceptual counterparts of logical relations between values and or types. Coherence is the conceptual counterpart of procedurality (related concepts are flow-of-control and program-state, although there is no simple one-to-one identification to be made). In the following section we will show that coherence is the open-world extension of provablity in a closed world.

Comparison with Prolog

The most useful way to describe CP is to compare it with Prolog and to show that Prolog is just one special case allowed by it. Operationally, Prolog is a theorem prover over a closed world consisting of a set of a restricted sort of logical expression. The theorem is proved (assuming that it is not present already) by displaying a model which is consistent with the world. The world is equated with a set of facts assumed to be true. In order to generate a model, a 'query' is presented to the system. The presence of variables in the *query* indicates that the theorem (the variables are actually existentially quantified) can only be proved if instantiation of the variables is achieved. If no direct match with the world model is possible, Prolog back-chains through a set of Horn clauses (essentially implications) until either a match is achieved or failure occurs. The set of instantiations forms the model which satisfies the query. The mere success of the back-chaining and matching procedure is sufficient grounds (in a closed world) to state the query as a theorem. If no model was generated, then the 'negation as failure' rule (again a consequence of the closed-world assumption) is -rounds to state the negation of the query as a theorem. 'Resatisfaction' may produce alternative models, but these merely further support the theorem.

Open-world reasoning, on the other hand, is forced to relax these requirements and assumptions. Provability in a strict logical sense is impossible when both queries and conclusions can be questioned. To replace it a notion of 'coherence' is needed which involves an evaluative procedure. The basic architecture of CP is however the same as the Prolog case. We still have a query and still present it to the world for testing. Now, however, the query is subject to interpretation, according to alternative meanings of the terms used. A query is an arbitrary proposition (just as in the Prolog case), but it may be of the form 'assuming P1, P2, P3... is C?' where the Ps and C are all propositions. The set of interpretations form *contexts* in which the query makes sense. Since queries are often incompletely specified a context needs to be established before the query can be answered.

The world consists of a set of *typed* propositions. These are not just true or false, but may be attitudinal in the same sense that modal logics try to encompass other propositional types. The next step is to instantiate the alternative interpretations in an attempt to generate models which support the query. 'Models are alternative pictures of the real world generated in accord with the constraints that the real world provides. The complexity of the form of these models is in stark contrast to the truth theoretic denotations of logical expressions in a closed-world system. All of these models may support the query in the sense that the query has a projection in the model. In this case the query may be asserted as a fact and incorporated into the world. Note that Prolog does not usually incorporate proved theorems into the system, although it is easy to do so if required. The models may however contradict the query and form grounds for asserting the negation of the query. Between these two extremes lie cases where some models support and some contradict the query. There must be therefore an evaluation carried out of the models in the light of the query to see whether there are grounds for asserting the query, its negation or accepting it as a belief, or possibility. The strength of the link between the assurnptions and the conclusion in the query is important here because the conclusion may be accepted if, for instance. the query is of the form 'assuming P1, P2, P3... is it *possible* that C'. On the other hand if the link is one of *necessity* then the assertion may not be made.

The structure of a CP program

The terms used in a conceptual representation svstem need definition. Following Sowa (1984), there are two mechanisms for definition. Concept types can be defined in an Aristotelian way, i.e. through a set of necessary and sufficient conditions by which the type may be distinguished from all other subtypes of its parent type;

they can also be defined using schemata, Which express alternative contexts in which a term can gain meaning. An example of an Arisotelian definition is of a pen as a writ ing-i nstrument which contains ink. A schematic cluster for hammer, however, would contain descriptions of the use of a hammer to hit nails into wood; to help in constructing machinery; to help demolish a house and so on. The Aristotelian definition of hammer as an instrument with a metal head and a wooden handle used to hit things does not capture all of these shades of meaning.

The terms thus defined form an incomplete lattice of types, ordered by the relation 'subtype'. The topmost concept (written ⊤, pronounced 'top') is that type of which all others are subtypes. It is the universal type. The bottommost concept (written ⊥, pronounced 'bottorn') is the type which is a subtype of all others. It is often called the absurd type since there are no individuals in the world which are of this tvpe.

Procedurality is represented in a CP program by the integration of relevant actors into the type definitions (Hartley, 1985). They express the necessary information for instantiation of concepts in order to produce concrete models of the world. A definition of the concept of giving would, for example, show how an object passes between two people. The relevant actor would deinstantiate the state of one person having the object and instantiate the state of the other person possessing it (See figure 1 for the graphical and linear notations for this example). Actors can express causality such as this but can also express the inferences necessary for problemsolving. The lattice of type definitions, including causality and inference-making actors comprises the terminological component of CP. The closest analogy with standard languages is that with declarations of types in a strongly-typed languages. In particular, the specification and implementation of abstract data types using the 'package' construct in Ada is similar. The difference, as pointed out before, is that types may be defined by mentioning different contexts in which they occur and not just constructed out of more abstract types.

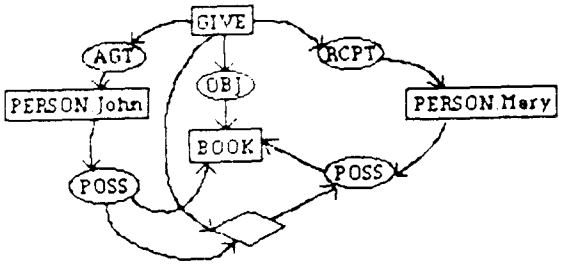

Figure 1a. John giving Mary a book

```
<>-
  <- [STATE: (PERSON: John] -
             (POSS) -> [BOOK: * b]],
  <- [EVENT: [GIVE] -
             (AGT) -> [PERSON: J ohn]
             (OBJ) -> [BOOK: * b]
             (RCPT) -> [PERSON: Mary]
  <- [STATE: [PERSON:Mary ] -
             (POSS) -> [BOOK: * b]].
```

Figure 1b. The Linear form of Fig. 1a.

As well as a set of definitions, the system has a component which handles the incremental assertion of arbitrary propositions. This assertional component initially contains a set of canonical descriptions, some definite (involving individuals) and some indefinite (involving sets of individuals). Whenever a new proposition is presented as a query and its related models are evaluated, the proposition is typed (fact, belief, possibility, etc.) and added to the assertional component. At any one time, the contents of this component can be taken to represent the sum total of knowledge possessed by the system. It is feasible that they can also be time-stamped and the passage of real time will make some of the assertions 'true-! n-the-past' rather than 'true-now'. In any case, it is appropriate to call such a knowledge base persistent in that it can quite naturally exist from one use of the system to another. There is no need, therefore to start every application use of CP from scratch. It is possible to use previous information or results of previous sessions in a new area.

Execution of a CP program

In order to illustrate the operation of a CP interpreter a simple example will be presented. Although it displays many of the features of the conceptual programming methodology, it is a closed-world -example which could be programmed in Prolog. The crucial aspects of open-world programming i.e. the evaluation of the models generated from query interpretations and the revision of the query in the light of it, is too long to display here.

The example concerns a fragment of knowledge about the nature of paint and whether or not it will peel off a wet surface (following Mike Coombs). Only two types of paint will be represented: oil paint which does peel off and acrylic paint which does not, because 'it absorbs any surface water. Clearly there is scope here for uncertainty (how much water can acrylic paint absorb? can oil paint tolerate a little moisture?) but this is not handled in the example in order to keep it simple. The definitions and interpretations are presented in graphical form since graphs are easier to read.

Firstly, the definitions of oil paint and acrylic paint. Oil paint is defined as peeling off a wet surface (figure 2)

and acrylic paint -,Ls absorbing surface water (figure 3). Note that the actors produce the correct event (peeling, off absorbing) as Ion,, as their input,-, are instsantiated.

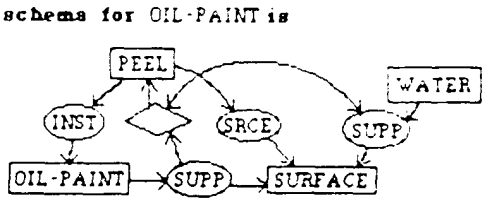

Figure 2. Definition of oil paint

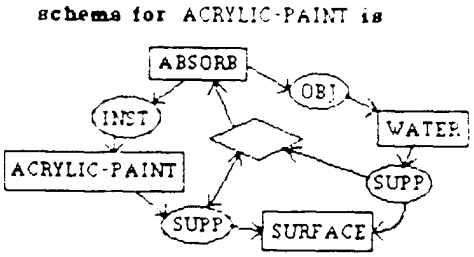

Figure 3 . Defini t ion of acryl is paint

The remaining definition is that of painting a surface. Here It is a simple schema which mentions a brush as an instrument (figure -1).

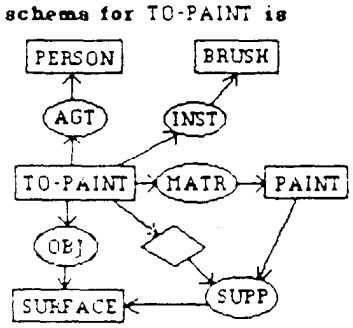

Figure 4. Definition of painting a surface.

The query which will be presented to the system is "When painting a wet surface, will paint stay on it?" It is in two parts. an assumption, namely -painting a wet surface" and a necessary proposition namely --a painted surface" The assumption will be used to form the query interpretations i.e. to put the query in some meaningful context. The actual question ("will the surface remain painted?') will be used to evaluate the models generated from these interpretations. The propositional mode of the question (necessity, possibility etc.) will determine the style of evaluation. The query may be represented linearly as:

```
[QUERY] -
  (PART) -> (ASSUMPTION:[SURFACE:*s] -
                (SUPP) <- [WATER]
                (OBJ) - [TO-PAINT]],
  (PART) -> [NECESS ITY:[ PAINT] -> (SUPP) -> [SURFACE:*s]].
```

The first step is to interpret the query in the light of any assumptions which may need to be made. By joining the assumption referent to the definition of TO-PAINT we get Figure 5. Subsequent separate joins to the definitions of OIL-PAINT and ACRYLIC-PAINT produce the interpretations in figure 6.

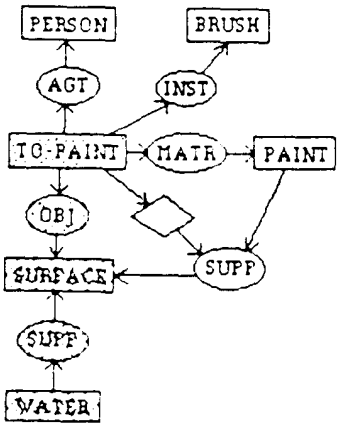

Figure 5. Assumption join.

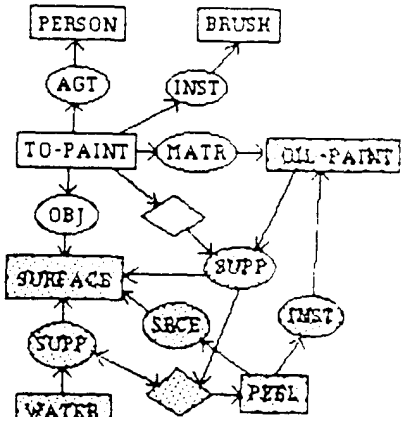

Figure 6a. The interpretation for OIL-PAINT

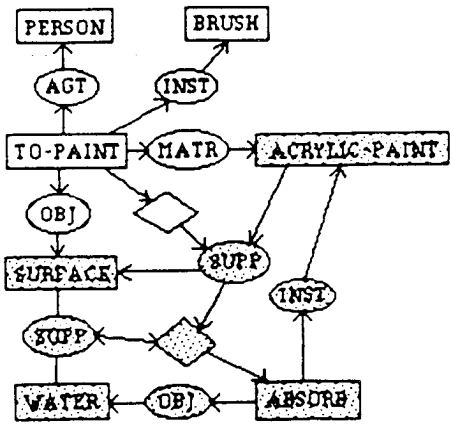

Figure 6b. The interpretation for ACRYLIC-PAINT.

The next step is to instantiate the interpretations to produce models. In order to do this a strategy has to be employed. The strategy needed here is simply one of making sure to create a wet surface before painting it. This is the definition in figure 7.

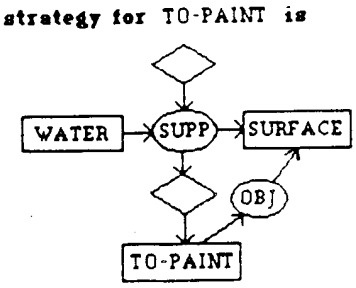

Figure 7. The strategy for painting.

The strategy is joined with the intepretations to produce a 'program' which will generate models when the actors contained in it are initiated. The join for OIL-PAINT is shown in figure 8. The one for ACRYLICPAINT is similar. There is only one actor with no preconditions. It immediately instantiates the state of a surface having water on it. This in turn starts up the painting actor which then paints the surface. This in turn causes the peeling actor to fire, and it 'removes' the paint from the surface. The whole model is a sequence of events and states each having reference to concrete surfaces, agents, paints and brushes. The ACRYLIC-PAINT program goes through the same sequence, but the model shows how the water on the surface is absorbed, allowing the paint to stick. These models are (briefly):

For OIL-PAINT:

- SURFACE s (has) WATER w
- PERSON p TO-PAINT t SURFACE s (using) BRUSH b (with) OIL-PAINT o
- OIL-PAINT o PEEL (from) SURFACE s
- SURFACE s (has) WATER w

For ACRYLIC-PAINT:

- SURFACE s (has) WATER w
- PERSON p TO-PAINT t SURFACE s (using) BRUSH b (with) ACRYLIC-PAINT a
- ACRYLIC-PAINT a ABSORB ab WATER w (on) SURFACE s
- ACRYLIC-PAINT a (on) SURFACE s

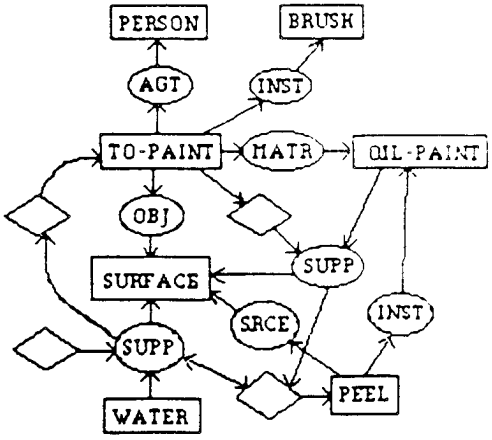

Figure 8. The program for OIL-PAINT.

The final step is to 'answer the question' by joining the available models to the question part of the query. In the case of OIL-PAINT this produces the empty graph (indicating a contradictory model) whereas the ACRYLIC-PAINT model produces :

[ACRYLIC-PAINT: a] -> (SUPP) -> [SURFACE: s]

indicating a supporting model. Since the requirement is one of necessity, the answer is effectively 'no' (cf. the Prolog 'no'). However, if the requirement was one of possibility, the answer would be 'yes' This last stage of model evaluation gives CP great scope for open-world computations.

Conclusions

Several issues have emerged as a result of preliminary thinking in the areas of improving expert system tools, open-world computation and knowledge representation using conceptual graphs . They include:

- should the intepretation procedure be defineable or is it governed only by the pre-set definitions of actors?
- is Sowa's maximal join operation adequate as a graph merge or is it just one of a family of merge procedures?
- is evaluation of models defineable or is it completely determined by the query type?
- how are standard computations to be incorporated?

e.g. arithmetic, symbol manipulation

- how does CP relate to explanation-based expert system work? i.e. are CP models explanations?
- are propositional types (fact, belief, possibility, assumption, necessity etc.) primitive or can they be defined through schematic clusters relating to query types?
- is the evaluation of instantiated query interpretations the only way to compute in open worlds?
- how does CP relate to truth maintenance and nonmonotonic reasoning?

We believe that if the answers to some of these questions are forthcoming- then conceptual programming can become a viable tool for knowledge engineering. We are already working on three such applications where CP forms a central pillar around which the application is built. One concerns the design of a tutor for novice Prolog programmers who generate misconceptions of the language's procedural semantics (Coombs, Hartley &, Stell, 1986). A second concerns the integration of numeric information from multiple sources and its interpretation for the purpose of hypothesis generation and testing. This forms a part of the Science Workbench project at the Computing Research Laboratory (Conley et al., 1985). The third project is to incorporate the open-world nature of the CP execution cycle (query interpretation, model generation, model evaluation) into a knowledge acquisition system in which mcomplete and inconsistent information can be handled.

References

Brachman, R.J. (1979). On the epistemological status of semantic networks. In Associative Networks, N. Findler (Ed.). New York: Academic Press.

Buchanan, B.G. (1082). New Research on expert systems. In Machine Intelligence 10, J.E. Hayes, D. Michie and Y.-H. Pao (Eds.), pp. 269-209, Chichester:Horwood.

Conley, W., Slator, B., kriderson, M. and Sitze, R. (1985). Designing and prototyping a scientific problem-solving environment: The NMSU science workbench. In Research data mana-ement in the ecological sciences. W. Michener (Ed.), University of South Carolina Press.

Coombs, M.J. Hartley, R.T. & Stell, J.L. (1986). Debugging user conceptions of interpretation processes. MCCS-86-46, Computing Research Laboratory, New Mexico State Universitv.

Hartley, R.T. (1085). Representation of procedural knowledge in expert systems. 2nd IEEE conference on Al applications (in press).

Hewitt, C. (1985). The challenge of open svstems. BYTE, April 1985.

Sowa, J.F. (1984). Conceptual structures. Reading, Mass.: Addison Wesley