The Conceptual Programming Environment, CP: Reasoning Representation using Graph Structures and Operations

Heather D. Pfeiffer † and Roger T. Hartley ‡
Reasoning Architectures Group
New Mexico State University
Las Cruces, NM, USA 88003-0001

Abstract

The Conceptual Programming environment, CP, being developed at the Computing Research Laboratory (CRL), is a complete knowledge representation visual programming environment for use with both dynamic, open-world, problem solving (weak) applications and static, closed-world, scientific analysis (strong) applications. CP is based upon a graphical methodology of visualization derived from John Sowa's conceptual graph theory. In this paper, we present the formal basis for the CP system, a discussion of the basic graph operations in the system, and display the visual elements of the environment. The CP environment is a 'working' representation system, and makes a good foundation for several applications.

1 Introduction

The Conceptual Programming environment, CP, is an ongoing project at CRL. The CP system is a knowledge representation development environment within a graphical visualization framework ([18], [10]). Although most problem solving systems in Artificial Intelligence (AI) are based on some form of predicate logic, and many scientific analysis systems use some kind of pattern recognition, there has recently been some work on systems that are based on graph structures, and operations on those structures. In the CP system all knowledge is represented by graphs and operations (mappings) performed on those graphs. In contrast to a problem solving logic-based approach, such as STAR [14], that reduces a set of predicates displayed by graphs using resolution, or a scientific data analysis task, such as PROTEAN [12], that employs pattern generation using metarules in a "control knowledge base", we present an approach that uses a constructive technique based on the actual graphs displayed and their graph transformation operations [9].

CP graphs are patterned after Sowa's conceptual structures ([20], [19]) using the operations defined for conceptual structures on the graphs. *Conceptual graphs*, as defined in Sowa's book, express declarative knowledge using concepts, relations, and actors 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 to actors, both syntacticly and semanticly, where they perform more like "functional relations" ([17], [15], [4]). 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; where as, within the spatio-temporal domain an ontology for objects, events, states and processes is provided within this same framework.

An example of a problem solving system built on top of the CP environment has been developed here at CRL. 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, MGR".

†Computing Research Laboratory
‡Computer Science Department

2 Representation

2.1 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 ([20], [19]) 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 [13] instead of FOPC.

A *conceptual structure* $\mathcal{CS}$ is a connected multilabeled bipartite oriented graph [6]. The two colors of nodes in a conceptual structure are called *concepts* and *relations*. 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 ([6], [16], [19]). 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.

2.2 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 graphicly 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. These 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. Figure 1 shows a graph, a simple one-actor overlay and the graph that results from laying the overlay on the first graph. Following Sowa, the concept nodes are presented as squares (or rectangles) and relation nodes as circles (or ovals).

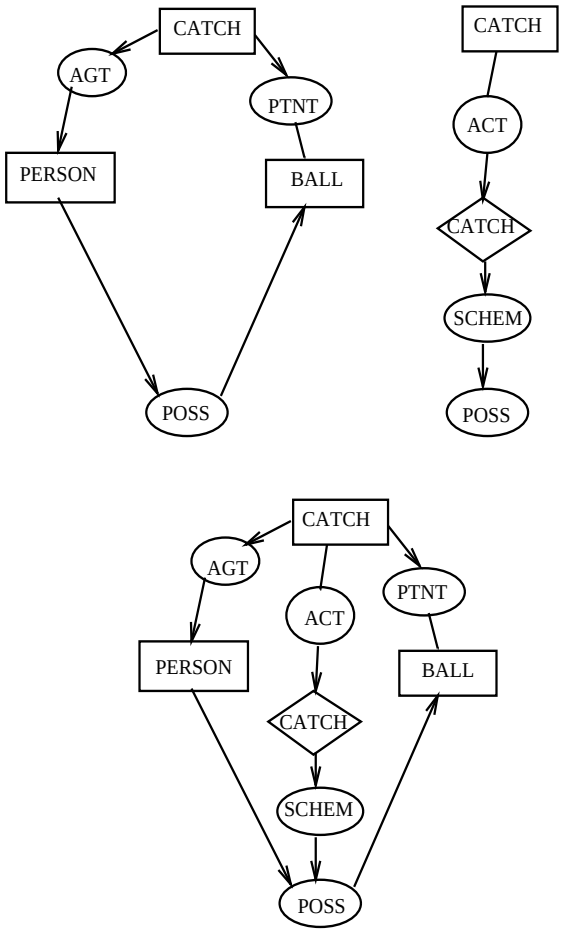

Figure 1: A conceptual graph, an actor overlay and the overlaid graph.

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 [15].

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 ([17], [4]). 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* [11]. This time chart is similar in use and meaning to the time maps of Dean and McDermott [3] and have correspondence to Allen's relations [1].

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.

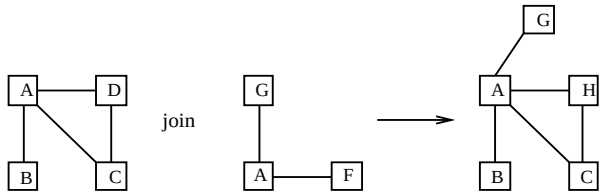

Figure 2: Maximal join ($M_b\ (D,\ F) = H$).

3 CP Graph Operators

Using the basic representation given above, CP has two operators, *join* and *project*, that can manipulate the graphs and overlays by following the rules laid out by the type hierarchy [9]. These operators are duals (i.e., union and intersection), therefore, the description of join is, in some sense, the dual of the description of project. Let us start by describing the join operator.

3.1 Join

Sowa's join operation is defined on concept types only. The job of the binary operation join is to merge two graphs at a single point where both graphs contain the same concept label, or a subtype. CP join, M_b, is always *maximal*, i.e., labels *may* be restricted by replacement with a label of any subtype, and graphs *will* be merged on the maximum number of nodes. An example is given in Figure 2. The functionality of join over a set of graphs $\mathcal{G}$ is:

$$\textit{maximal join}\colon \mathcal{G} \times \mathcal{G} \rightarrow 2^{\mathcal{G}}$$

There can be more than one maximal join, hence the powerset notation on the set of all graphs $\mathcal{G}$. Join is a binary operation but multiple graphs can be joined by composing it with itself. Unfortunately, there is good reason to believe that join is not commutative when semantic considerations come into play [16], but for now we will assume there is no problem.

Since restrictions are allowed, it is clear that two nodes are joinable as part of a maximal join operation if they contain types that have a maximal common subtype. See papers ([18], [16], [9]) for examples.

3.2 Project

Project is the inverse of the operation join. It is based on the same idea of merging two graphs, but this time taking the minimal common supertype of the

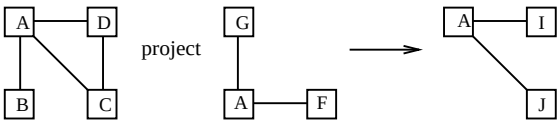

Figure 3: Projection (note: $M_p\ (D,\ F) = I$).

concepts, M_p. Just as join is maximal, so project is as well. If join is likened to set union, in that all nodes not joinable are just left alone, and come along for the ride, then project is like set intersection. All nodes that are not projectable are simply dropped from the resultant graph, along with their associated relation nodes.

Project's functionality is the same as join:

$$\textit{maximal project}\colon \mathcal{G} \times \mathcal{G} \rightarrow 2^{\mathcal{G}}$$

The example in Figure 3 shows projection with the reduced form of graphs. These operand graphs are the same as in Figure 2.

Since join and project are duals, we will not waste space by repeating the same arguments that we presented for join.

The following set of correspondences are sufficient to indicate how project works:

Join	$\longleftrightarrow$	Project
Max. Subtype	$\longleftrightarrow$	Min. Supertype
Union	$\longleftrightarrow$	Intersection

4 CP - Bouncing ball example

The CP environment has been under construction for five years and exists in its full form on Symbolics machinery using Symbolics Common Lisp, and in an updated, yet not complete, form on Sun machinery using Allegro. The following example is presented in order to give one a visual feel for the system.

4.1 Conceptualizing the problem

In order to use the CP system, the analyst must first conceptualize the problem they wish to reason about. Let's take the example of a bouncing ball. When a ball is bounced, it goes through a set of stages, moving in a repeated cycle through these stages until the height of the bounce becomes minimal, such that, the ball now rolls or halts. These stages are dependent on the height and speed of the bouncing ball and can be broken down as follows: 1) the ball starts at a certain height (suspended), 2) when dropped, the ball starts falling, 3) when the ball hits the ground

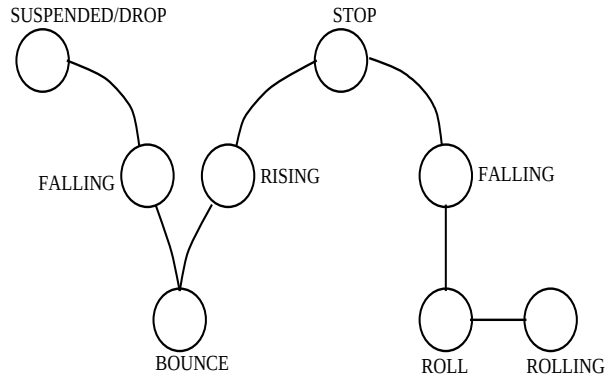

Figure 4: Conceptual diagram for a bouncing ball.

it bounces which causes it to start rising, 4) when it reaches a particular height it stops rising and starts falling, and 5) when not enough height or speed for the ball to bounce, it starts rolling until halt.

Using Figure 4 as a reference diagram, there are four events involved in bouncing a ball: drop, bounce, stop, and roll. Linked to these events are four states: suspended, falling, rising, and rolling. For each event involved in bouncing a ball, a definitional (conceptual knowledge) graph would need to be created within the system. A graph would also have to be created to describe the actual object being bounced, the ball.

The interrelationships of all the events and states will need to be graphically made by entering time overlay graphs for each event. Also any runtime overlay graphs must be created to compute values needed in order to cycle through these time relationships. All this definitional information is needed to describe the environment of a bouncing ball.

4.2 Program execution

Once the analyst has done a conceptual analysis of the problem, their graphs should be entered into the CP environment. The CP environment has a built in loader and editor. In order to use the CP environment, see the CP manual [8].

In order to do any reasoning about a bouncing ball, one must have a belief or at least an assumption that there is a ball and at what height it is suspended at. The CP system calls this 'factual' information, and the analyst should enter this information as a *fact*. Figure 5 is an example of a graph that could be selected as the belief that one has a ball that is in the state of suspension at the height of 10 units.

Given the fact selected, the graphs (both definitional and overlays) for all four events are selected and

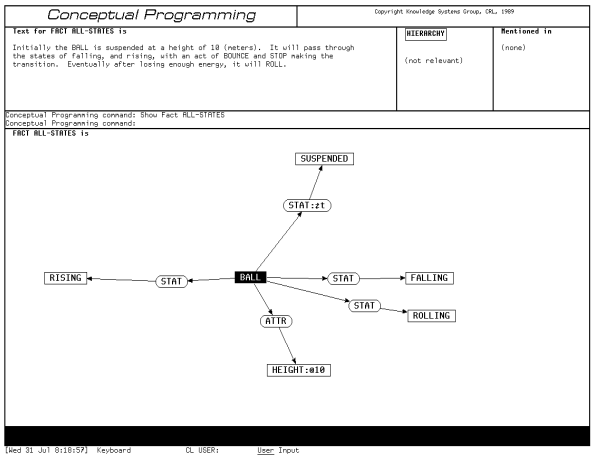

Figure 5: Fact graph for a bouncing ball.

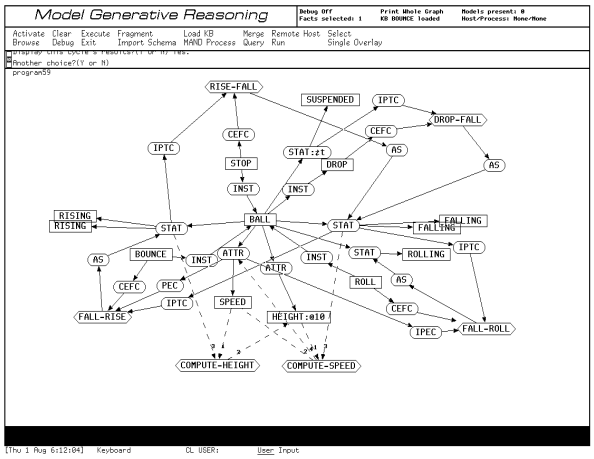

Figure 6: CP program for a bouncing ball.

joined together to form a program (a CP *program*) for a bouncing ball (see Figure 6).

As described in the introduction to this paper, CP is an visual environment for entering the representation of information about the data and knowledge to be reasoned about. It also has two operations, join and project, built into its functionality. CP, however, does need a controlling application for performing the actual reasoning. For the bouncing ball, we will use MGR, mentioned in the introduction, as the reasoning application.

As explained in the conceptual description of the problem, the CP program discussed in the above section has a cycle in it of falling and rising with the 'stop' event at the top of this cycle. The reasoning system will instruct CP to execute this program. Informally, the state of being suspended enables the 'drop' event to assert the state of falling. From there the falling state enables a bounce or roll event. If a bounce event is asserted, then the ball enters the state of rising until the stop event is enabled. The stop event asserts the state of falling which will then either continue the cycle or enable a roll event. If a roll event is entered the state of rolling will be asserted which will end the cycle and complete the program.

During the execution of the program, all actors can be repeatly 'fired' or executed until completion of the program. On completion of all firings, a model of a bouncing ball is produced. Through evaluation of the produced model and the time chart given for the spatio-temporal actors, an explanation can be made about the bouncing of a ball.

5 Conclusion and enhancements

In this paper we have presented a representation system based on graph structures along with the graph operations that are defined over these graph. The representation visual programming environment, the Conceptual Programming environment, CP, has been used in several applications including robot task planning [5], qualitative physics ([17], [4]) and genetic sequence mapping [7]. Systems built using this representation visual programming environment are intended to be used in real time, dynamic situations and so detail has been given to the basic structures and operations.

The system is currently in the processing of being fully moved and updated on to SUN workstations and also being enhanced in order to run in parallel on a parallel architecture machine, such as a Sequent.

Work is also being done on the entrance and editing process of graphs. Because conceptualization of a problem is sometimes difficult, special purpose domain interfaces are being attempted. One such application, for use by the Army, enters unit information for intension analysis [2]. Work is also proceeding on improving the CP editor in order to enhance the user interface for entering and editing graphs. These enhancements will hopefully be completed shortly.

Acknowledgements

Input for the original specifications and design of the CP environment was given by Michael J. Coombs here at CRL. The authors would like to thank him for

his review of this paper and for his input and support of this work.

References

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

[2] M.J. Coombs, R.T. Hartley, and H.D. Pfeiffer, "Developing Computing Technology for Modeling Enemy Intentions using Environmental and Doctrinal Information," *Memoranda in Computer and Cognitive Science*, MCCS-90-184, CRL, New Mexico State University, Las Cruces, 1990.

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

[4] 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.

[5] D.P. Eshner, R.T. Hartley, C. Lennox, and M. Moya, "Conceptual representation for robot task planning," J.F. Sowa, N.Y. Foo, and A.S. Rao (Eds.), *Conceptual Graphs for Knowledge Systems*, New York, NY: Addison Wesley, 1990.

[6] 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.

[7] C.A. Fields, H.D. Pfeiffer, and T.C. Eskridge, "Knowledge representation and control in **gm1**, and automated DNA sequence analysis system based on the MGR architecture," *International Journal of Man-Machine Studies*, 34, pp. 549-573, 1991.

[8] R.T. Hartley, "CP - The Manual," *Memoranda in Computer and Cognitive Science series*, MCCS-88-127, CRL, New Mexico State University, Las Cruces, 1988.

[9] 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.

[10] 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.

[11] 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.

[12] B. Hayes-Roth, B. Buchanan, O. Lichtarge, M. Hewett, R. Altman, J. Brinkley, C. Cornelius, B. Duncan, and O. Jardetzky, "PROTEAN: Deriving protein structure from constraints," *Proc. of AAAI-86*, Los Alos, pp. 904-909, 1986.

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

[14] Lockheed Missiles and Space Company, Inc., *Air-Land Battle Management Program Decision Aids for Battle Management Planning*. Austin, 1989.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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