

The Conceptual Programming Environment, CP: Time, Space and Heuristic Constraints

Heather D. Pfeiffer[†] and Roger T. Hartley[‡]

Reasoning Architectures Group
New Mexico State University
Box 30001/3CRL
Las Cruces, New Mexico 88003-0001
Phone: (505) 646-5782 and 646-5181
email:hdp@nmsu.edu and rth@nmsu.edu

Abstract

The Conceptual Programming environment, CP, being developed at New Mexico State University is a complete knowledge representation 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 describe a new internal representation and explain the mechanisms for the operators relating to the latest aspects of constraint processing for time, space and heuristics within the CP environment.

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 framework (Hartley, 1986; Pfeiffer, 1986; Hartley and Coombs, 1990). The graphs within the system are patterned after Sowa's conceptual structures (Sowa, 1984; 1990), in which we present an approach that uses a constructive technique based on the actual graphs displayed and their graph transformation operations defined over the conceptual structures (Hartley and Coombs, 1990; Pfeiffer and Hartley, 1989; Eshner and Hartley, 1988).

CP graphs are actually extended forms of Sowa's *conceptual graphs*. Conceptual graphs, as defined in Sowa's book, express declarative knowledge using concepts, relations and actor nodes, and linked together through the edges. CP also expresses declarative knowledge, but introduces a representation for expressing procedural and constraint knowledge, *overlays*, through extended features to actors. These features are both syntactic and semantic, and make the actors perform more like "functional relations" (Eshner and Hartley, 1988; Pfeiffer

[†]Computing Research Laboratory

[‡]Computer Science Department

and Hartley, 1989). It is through *overlays* that declarative knowledge (definition graphs) can be constrained by time, space and heuristics.

Our extensions to Sowa's conceptual graph theory introduce an "overlay" level that 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; within the spatio-temporal domain an ontology for objects, events, states and processes is provided within this same framework. The CP environment has incorporated into the processing of graphs, operations to handle and constrain internal processing much like "heuristic rules" (metarules) when required by this level.

The overlay level was introduced in order to allow the CP environment to implement a broad scope of mechanisms for different applications. The need for this layer of processing came about because it was desired that CP be able to efficiently handle both weak problem solving applications (Coombs and Hartley, 1987; 1989; Coombs et al, 1986; 1990; Eshner et al, 1990) and strong scientific analysis applications (Eshner and Hartley, 1988; Fields et al, 1991). With a totally unconstrained system, language analysis (or parsing), mechanisms for processing declarative knowledge are sufficient. However, with more complex, noisy, and novel tasks, such as path planning, mechanisms for procedural knowledge (time/space) now becomes necessary. When the task requires some sort of numerical analysis as well as being complex, as in sequence analysis in genetics, mechanisms for constraint knowledge (physical) comes into play. Through the overlay level we are able to handle all these different kinds of applications.

In our first system design, the overlay level was not an integrated part of the system. It was actually designed and added on in pieces. This eventually produced the "band-aid" effect. To eliminate major problems and greatly enhance the CP environment, we have completely rewritten the system. The following describes some of the new internal representation implemented, and how this overlay level is represented and processed in the new system.

2. Representation

Representation within the CP system has been modified to give a more concise and efficient internal structure. It has taken on two principles of *object oriented programming* (Sethi, 1989), 1) encapsulating the data being stored with the operations being performed on that data, and 2) creating the concept of a "vanilla graph" object and having all CP graphs be extensions of this object. Figure 1 represents the inheritance structure of the CP graphs; where as, at each *object* in the structure, some data encapsulation does occur. Each object, however, is built on the vanilla definition of a *graph* with each object lower in the inheritance structure able to apply more and more specific syntactic and semantic processing. This object oriented approach has not only improved the implementation of the CP system, but has "cleaned up" the design specification of each object represented in Figure 1.

2.1. Storing CP Graphs as Database Tuples

General database theory is constantly examining the internal representation of the storage structure in order to decrease access time and space without losing information. By evalu-

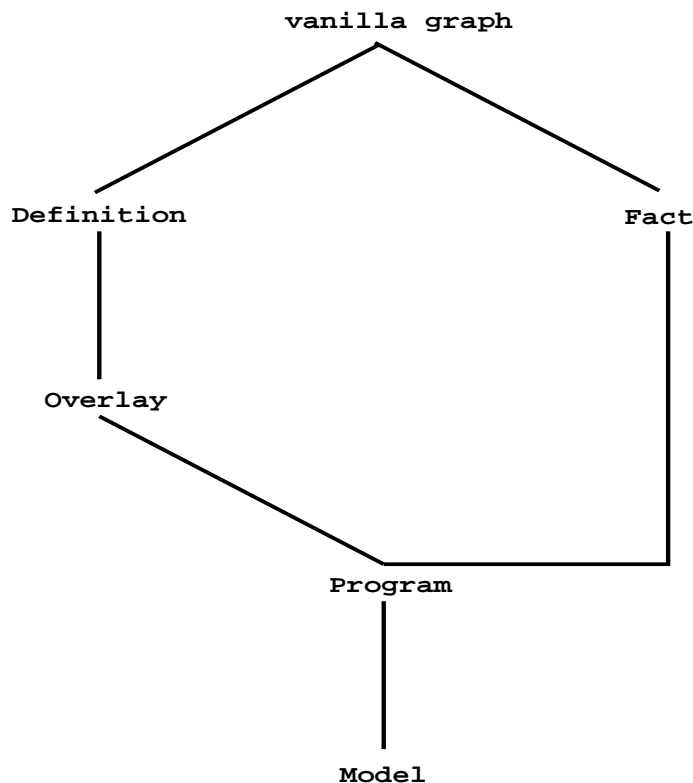


Figure 1: Inheritance structure for CP graphs.

ating these enhancements (Ullman, 1988) from the perception of stored knowledge, we have made major changes in the storage of the CP graphs in memory and disk. The graph in figure 2 was stored in the original CP system as 20 pairs of nodes, two for each link (forwards and backwards). Not only did this waste space, but increased processing time for the operators by adding extra computation in order to evaluate whether two graphs were available for joining or projection (Hartley and Coombs, 1989). The new representation breaks a CP graph up into concept-relation-concept or actor-relation-concept ‘node’ triples (3-tuples). Each node is uniquely stored with its property list holding the concept, relation or actor *label*, type and referent, for each available node. Through this new representation, the graph in figure 2 can now be stored as 5 triples, greatly reducing space requirement. This representation reduces the computational time needed for the operators by taking advantage of the internally stored “crc” and “arc” relationships. This representation has also allowed the implementation of some internal cache hashing functions that quickly retrieve the information from the nodes for the operators.

2.2. Extending Conceptual Graphs with Overlays

Sowa has shown how unknown objects (concept nodes with no individual field) can be computed by an *actor* node that corresponds to a function in standard logics. In our extensions

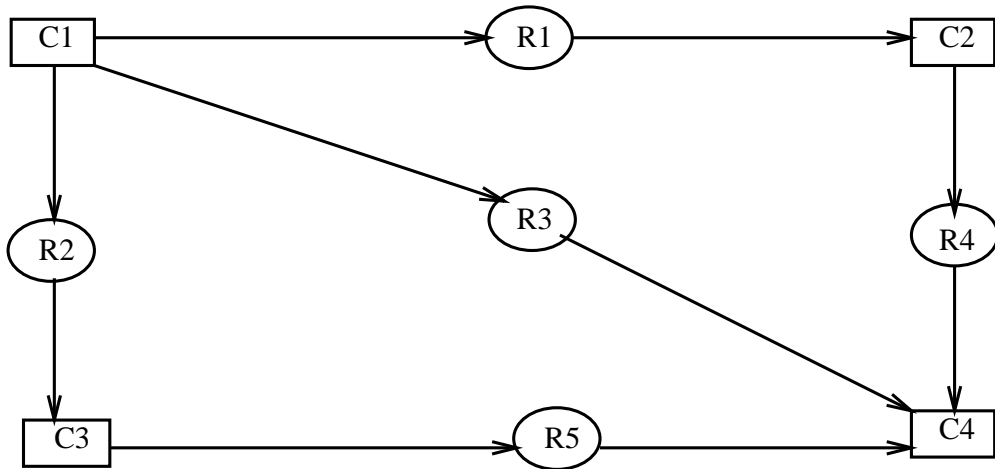


Figure 2: A conceptual structure, with four concepts, five relations, and nine labels.

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. The overlays in CP are seen as a new level of graphs that live on top of the basic graph definitions.

Within this level, there are actually two sublevels, *spatio-temporal* domain, and *feasibility-runtime* domain. The spatio-temporal sublevel uses qualitative actors, and the feasibility and runtime sublevel uses quantitative actors as described above. This level requires a syntactic extension to conceptual graphs in order that the diagrams not become too confused and thus lose their force. An *actor* within CP graphs can best be thought of as a “functional relation”, where there is a semantics (performed by the procedure) being represented graphically between two objects. The relation 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. The CP actors however, can be run forwards or backwards, or operate as constraint checkers, just as a Prolog rule can. In this manner an actor can compute a missing relation.

Temporal actors compute missing temporal relations, and spatial actors compute missing spatial relations. Figure 3 gives in time chart form (see Section 3.1) examples of input temporal relations, and figure 4 gives examples of output temporal relations. The same notions can be applied to create spatial actors (see Hartley, 1991 for examples).

Feasibility and runtime constraint checking on the nodes in the graph can be performed by the addition of heuristical overlays. These constraints allow CP to deal more efficiently with scientific and numeric data. These are designed as heuristic actors, computing *quantitative* constraints, propagating through a system of values and variables. Through the actual

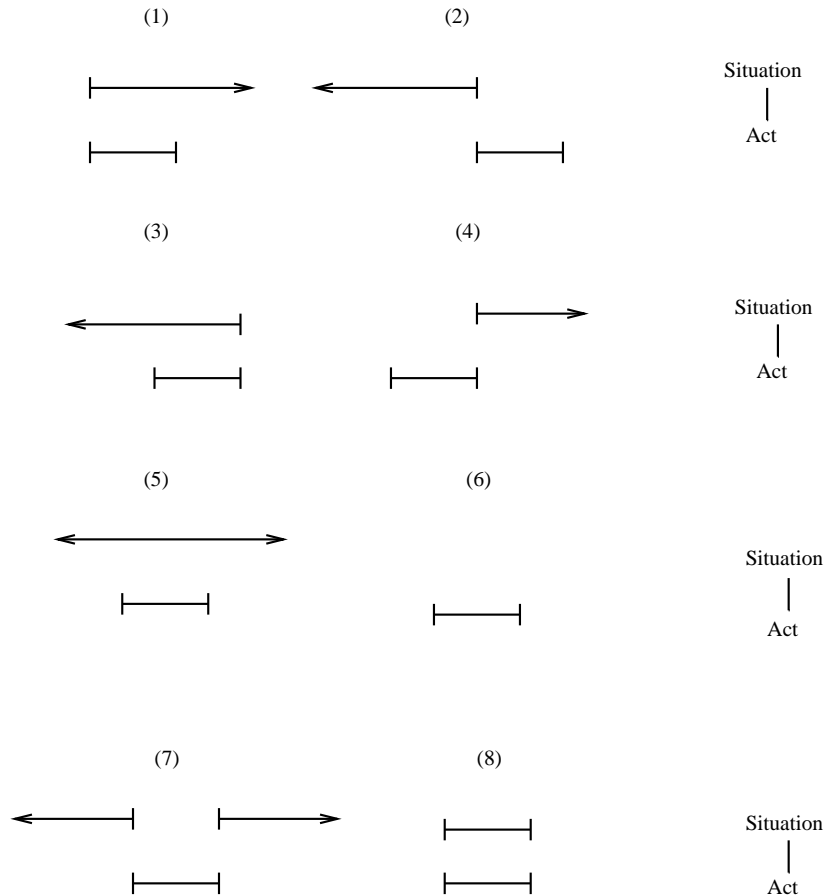


Figure 3: Time charts for the input temporal relations. (1) is a triggering enablement where the situation triggers the act and persists after the act finishes. (2) is the inverse where the absence of the situation is the trigger. (3) is a temporary enabling condition where the situation that enables the act disappears at the end of the act. (4) is again the inverse. (5) and (8) are permanent enabling conditions where the situation (or its absence) persists even though the act terminates. (7) shows an interrupt enablement where the situation is interrupted by the act, but resumes after the act finishes. (8) is the inverse of this, but can also be seen as the situation and act coinciding in their start and end points.

procedure of the actor (free form Common Lisp code), an heuristic can be applied during the functioning of the operator (see Section 3.2).

3. Processing

The processing level of CP consist of its operators, *join* and *project*. The time, space and heuristic actors are all executed in conjunction with the presence of their overlays during either a join or project operation (Hartley, 1991; Hartley and Coombs, 1989; 1990; Pfeiffer and Hartley, 1990). Success or failure of the operation depends on whether the network of actors ‘fires’ successfully. The actors therefore provide a layer of semantic checking over and above the preservation of canonicity guaranteed by the operation.

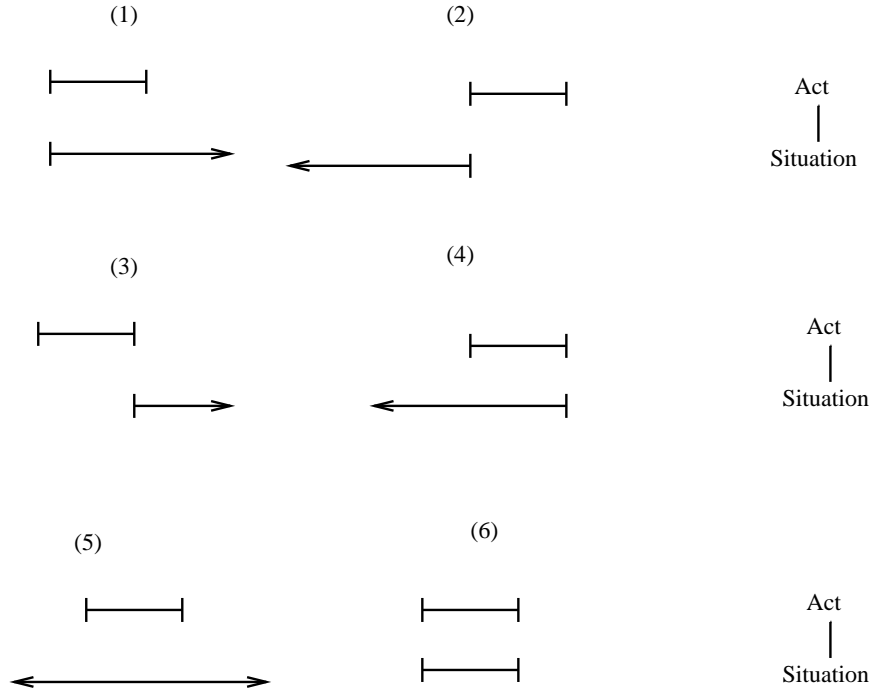


Figure 4: Time charts for the output temporal relations.

3.1. Time and Space Constraints

One set of rules in CP are represented as spatio-temporal actors, whose sole job is to act as confluence points for the knowledge structures that have to be related. As discussed above 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.

The ontology for time and space draws its inspiration from the decomposition of a conceptual graph into its (mostly) binary relations. Each relation is of one of four kinds: temporal, which links two *acts*; spatial, which links two *objects*; property, which links an object or an act to one of its properties; and case, which links acts to objects. Each of the first three kinds of relation can serve as input to or output from an actor.

In the temporal domain, inputs are partial states and schematics, since these are exactly what is expected to change in time. A *state* is an object and its properties; a *schematic* is a set of objects and their spatial relationships. 'Partial' indicates one binary relationship taken from a number of them. Each temporal actor has an act as input, at least one partial state or schematic as input and one as output. The crucial part of the whole idea of the overlay comes, however, with the temporal relationships that the act bears to the inputs and outputs. We have appealed to simple ideas of causality to analyze the possible relationships. These relationships are stored with the actor as a *time chart*, similar to Dean and McDermott's time map (Dean and McDermott, 1987). Notice that each of the interval pairs from Figure

3 (except 6 and 7) have correspondence to Allen's relations (Allen, 1985).

Similar ideas can be used 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. A *process* is an act and its properties. A *chronicle* is a set of acts and their temporal relationships. A spatial actor contains a *space map*. Figures 5 and 6 show example graphs in each domain.

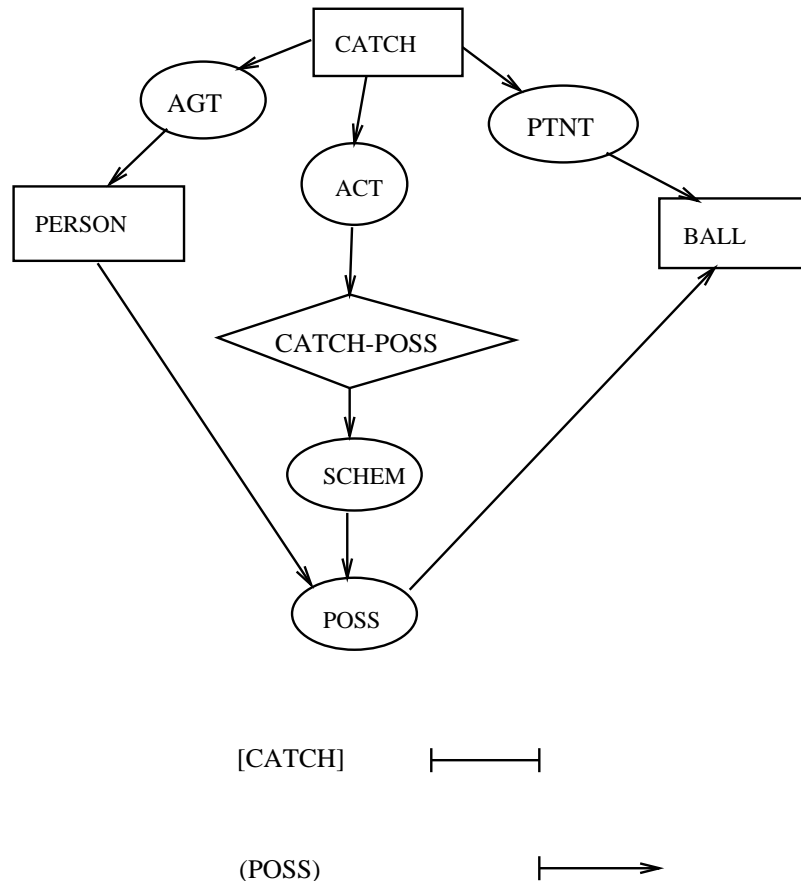


Figure 5: An example of a temporal actor

Time and space come together canonically in the *square* graph shown in Figure 7. Here all properties are omitted, leaving only temporal, spatial and case relations. If the temporal relation was absent, it could be computed using the spatial one, and *vice versa*, as Figure 8 shows.

Processing networks of these actors amounts to propagating the constraints expressed by their time charts and space maps in a consistent manner. Successful constraint propagation results in a successful graph operation. Failure causes the operation to fail as well.

3.2. Heuristic Constraints

Another set of rules in CP are represented as heuristic actors. These also execute in conjunction with the CP *operators*. These heuristics are set to operate like *before* and *after* functions

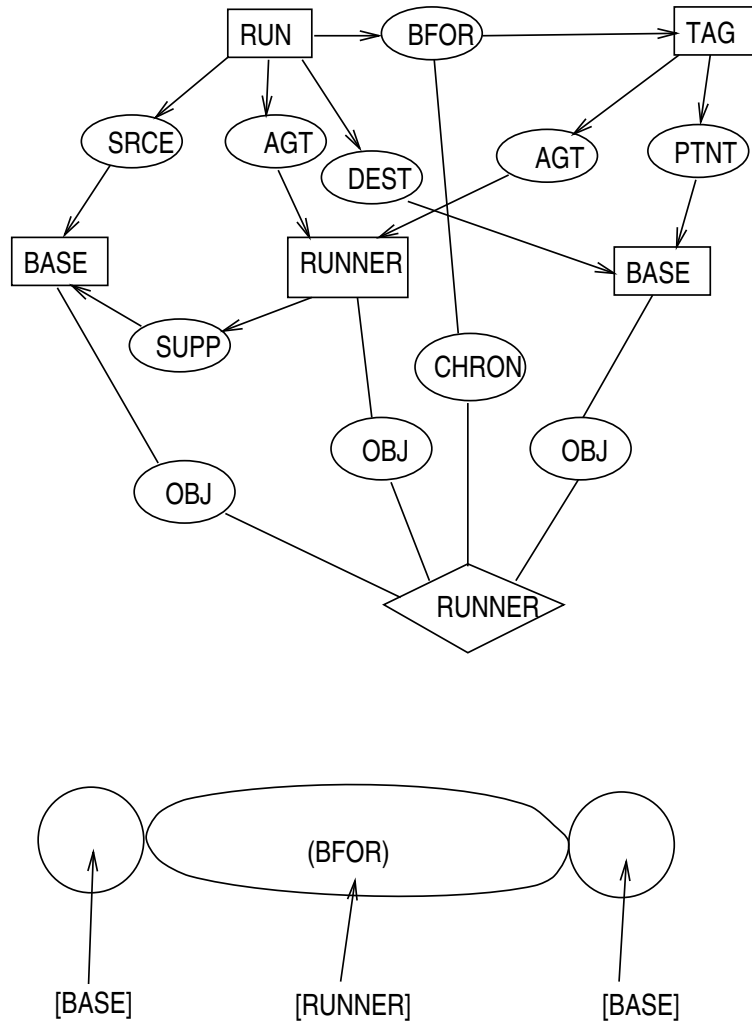


Figure 6: An example of a spatial actor

on the operators. The values of the concepts and/or relations connected to the actor act as input, output or input/output parameter to the corresponding actor function. Figure 9 is a code example for a heuristic *after* function.

4. Conclusion and Future Enhancements

The new rewrite of CP is currently being tested and will soon be incorporated into the existing MGR applications, in particular the tactical situation development system described in (Coombs and Hartley, 1989). The work on integrating spatial, temporal and constraint actors will also be implemented at this time.

Two other pieces of work take the CP environment as a basis. One concerns the parallelization of the MGR architecture, which involves parallelization of the basic CP operations. As a general framework we are implementing the operations as generators, so that where multiple results can occur (for instance with the operation join), they are produced in preferred order, one at a time.

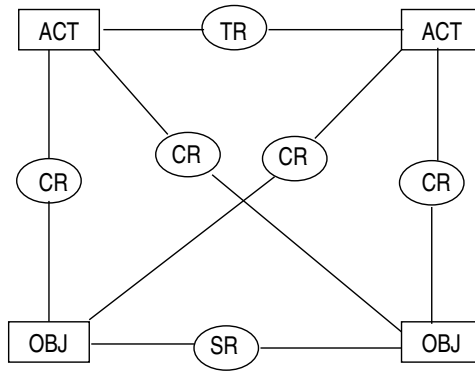


Figure 7: The canonical square graph

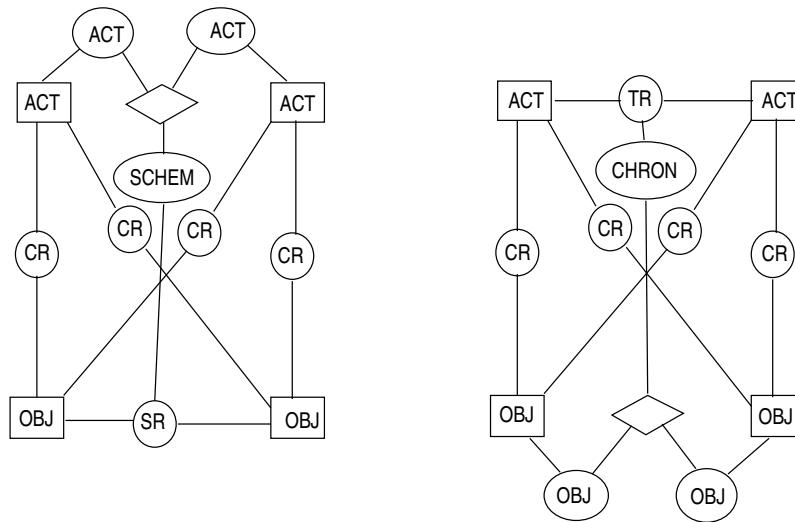


Figure 8: The canonical graph with actors supplying missing relations. On the left a temporal actor defines an implicit relation between the acts; on the right, a spatial actor defines an implicit relation between the objects.

This technique allows multiple generators to be working on different parts of a problem at the same time, and yet still ensure exhaustive search of the solution space. The other relevant piece of work is more theoretical in nature. We are engaged in an effort to axiomatize the CP operations and structures formally, with a view to generalizing a range of problem solving architectures over these structures. Eventually this will lead to a general theory of problem solving based on graph theoretic ideas. CP will develop alongside the refinement of these ideas.

<p><i>Conceptual Programming</i></p>	<p>Copyright Knowledge Systems Group, CRL, 1989</p>
	<p style="text-align: center;">Figure 9: Code for "COMPUTE-HEIGHT" actor.</p>
<p>Conceptual Programming command: Show Graphs STOP Conceptual Programming command:</p>	
<p>Constraint for actor COMPUTE-HEIGHT is</p> <pre> (DEFACOR COMPUTE-HEIGHT CONSTRAINT-OVERLAY FALL ((CONCEPT SPEED) (CONCEPT HEIGHT) (STATE STAT)) ((DECLARE ((SPEED NUMBER) (HEIGHT NUMBER) (STAT STATE))) (WHEN (AND (EQUAL STAT 't) SPEED) (ASSIGN-VALUE HEIGHT (/ (* 0.75 (* SPEED SPEED)) 64)) (FORMAT T "in compute-height, computing height = "A from speed = "A" HEIGHT SPEED))) </pre>	
<p>10</p>	
<p>[Tue 18 Jun 2:47:57] rth CL USER: User Input</p>	

References

- [1] Allen, J. (1985). Maintaining Knowledge about Temporal Intervals. *Communications of the Association of Computing Machinery*, 26(11), pp. 832-843.
- [2] Coombs, M.J., R.T. Hartley and H.D. Pfeiffer (1986). Conceptual Reasoning in Mobile Intelligent Robots. *Report 1. to Sandia Mobile Robot Project*, CRL, New Mexico State University, Las Cruces, NM.
- [3] Coombs, M.J. and Hartley, R.T. (1987). The MGR Algorithm and Its Application to the Generation of Explanations for Novel Events. *Memorandum in Computer and Cognitive Science, MCCS-87-97*, CRL, New Mexico State University, Las Cruces, NM.
- [4] Coombs, M.J. and R.T. Hartley (1989). Design of a software environment for tactical situation development. *Proceedings of the US Army Symposium on Artificial Intelligence Research for Exploitation of the Battlefield Environment*, El Paso, TX.
- [5] Coombs, M.J., R.T. Hartley and H.D. Pfeiffer (1990). Developing Computing Technology for Modeling Enemy Intentions using Environmental and Doctrinal Information. *Memoranda in Computer and Cognitive Science, MCCS-90-184*, Computing Research Laboratory, New Mexico State University, Las Cruces, NM.
- [6] Dean, T.L and D.V. McDermott (1987). Temporal data base management. *Artificial Intelligence*, 32, pp. 1-55.
- [7] Eshner, D.P. and Hartley, R.T. (1988). Conceptual Programming with Constraints. *Proceedings of the Third Annual Workshop on Conceptual Structures*, Minneapolis, pp. 3.1.2-1 - 3.1.2-6.
- [8] Eshner, D.P., Hartley, R.T., Lennox, C. and Moya, M. (1990). Conceptual representation for robot task planning. In: Sowa, J.F., Foo, N.Y. and Rao, A.S. (Eds.), *Conceptual Graphs for Knowledge Systems*. New York, NY: Addison Wesley.
- [9] Fields, C.A., H.D. Pfeiffer, and T.C. Eskridge (1991). 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.
- [10] Hartley, R.T. (1986). The Foundations of Conceptual Programming. *Proceedings of the First Rocky Mountain Conference on AI*, Boulder, pp. 3-15.
- [11] Hartley, R.T. and Coombs, M.J. (1989). Reasoning with Graph Operations. *Proceedings of Workshop on Formal aspects of Semantic Networks*. In: J. Sowa (Ed) *Formal Aspects of Semantic Networks*. New York, NY: Addison-Wesley.
- [12] Hartley, R. T. and Coombs, M. J. (1990). Conceptual programming: Foundations of problem solving. In: J. Sowa, N. Foo, and P. Rao (Eds) *Conceptual Graphs for Knowledge Systems*. New York, NY: Addison-Wesley.
- [13] Hartley, R.T. (1991). A Uniform Representation for Time and Space and their Mutual Constraints. In: F. Lehmann (Ed) *Special Edition on Semantic Networks in Artificial Intelligence, Computers and Mathematics with Applications*.

- [14] Pfeiffer, H. D. (1986). Graph definition system. *Proceedings of the Second New Mexico Computer Science Conference*, Las Cruces, pp. 97-103.
- [15] Pfeiffer, H.D. and Hartley, R.T. (1989). Semantic additions to Conceptual Programming. *Proceedings of the Fourth Annual Workshop on Conceptual Structures*, Detroit, pp. 6.07-1 - 6.07-8.
- [16] Pfeiffer, H.D. and Hartley, R.T. (1990). Additions for SET Representation and Processing to Conceptual Programming. *Proceedings of the Fifth Annual Workshop on Conceptual Structures*, Boston&Stockholm, pp. 131-140.
- [17] Sethi, R. (1989). *Programming Languages Concepts and Constructs*. Reading, MA: Addison Wesley.
- [18] Sowa, J.F. (1984). *Conceptual Structures*. Reading, MA: Addison Wesley.
- [19] Sowa, J.F., Foo, N.Y. and Rao, A (1990). *Conceptual Graphs for Knowledge Systems*. New York, NY: Addison Wesley.
- [20] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, Volume 1. Rockville, MY: Computer Science Press.