Conceptual Programming with constraints

Daniel P. Eshner & Roger T. Hartley
Knowledge Systems Group
Computing Research Laboratory
New Mexico State University
Box 30001
Las Cruces, New Mexico 88003

1. Introduction

This paper discusses the integration of constraint actors to the Conceptual Programming (CP) environment (Hartley 1988) developed at CRL. The CP system is an implementation of Conceptual Graphs (Sowa 1984) with an added layer called procedural overlays. These overlays allow the system to represent temporal and causal information using a temporal calculus and related time maps (similar to Allen's temporal relations (Allen 1983) and Dean and McDermott's (Dean and McDermott 1987) time maps. When executed, each procedure overlay produces a time map showing the relationships specified by the action. These inherently localized time maps are combined into a global time map by a further extension to CP called PRE (Eshner and Hartley 1988). Constraint overlays implement the original actors in conceptual graphs with two additions: each actor may behave like a formal constraint on a state or concept referent as well as a function, and constraint actors may take as input a state as well as a single concept referent. The next section briefly summarizes CP, procedural overlays and constraint overlays. The integration of constraint overlays and procedural overlays is then presented in an example that illustrates their use.

2. Conceptual Programming

CP is an environment for knowledge representation based upon selected parts of conceptual graph theory, with extensions to the representation of temporal events. Retained from the theory in Conceptual Structures are:

1. the hierarchy of conceptual types;
2. definitions of these types according to Aristotelian principles or through a schematic cluster
3. preservation of the coherence of knowledge structures through canonical operations
4. individuals of type name, number, i-mark and sets of these
5. functional actors
6. knowledge structures as graphs (CP has a graphical user interface)

Not included are:

1. prototypes or compound individuals;
2. co-reference links and quantificational contexts;
3. logical inference
4. definitions of relations
5. control marks
6. type expansion or contraction

The additions to the base CG theory are in two areas. Firstly, a natural extension to the idea of temporal relations into a complete action/event calculus. This work stemmed from Allen's seminal work on

temporal relations, but this has been modified to fit into conceptual graphs as an extension to the use of actors for dynamic events. Whereas the usual functional actors are executed to produce relationships between concepts and their referents, procedural actors in CP are executed to produce temporal relationships between events and the states that mediate them. In order to achieve this, CP provides a set of temporal relations which cover all of possible relationships between the time interval of an event and that of a state that enables it or is enabled by it (Figure 1). A network of temporal actors and their relations can be added to a definition as a 'procedural overlay'.

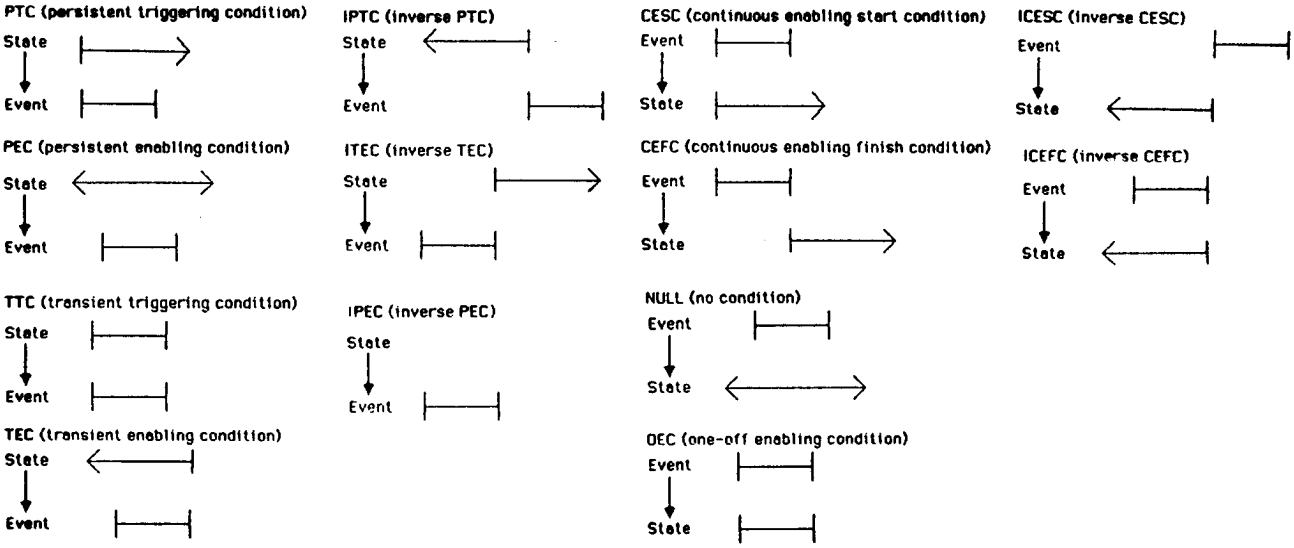

Figure 1. The set of actor relations and their time charts.

The second extension allows functional actors to be considered as full-blown constraints, and not just functions. This gives CP the capability proposed for the next version of Prolog as a constraint satis tion language. Constraint actors can also enable and be enabled by states, these providing a bri between a temporal (or procedural network) and the constraint network. Constraints are, like temp actors, added to definitions in a 'constraint overlay'. The example presented below illustrates both ex sions integrated into a single environment.

3. **A bouncing ball with naive physics**

In this example, we are trying to simulate a bouncing ball when dropped from a certain height. behavior should be one of repeated smaller bounces until the ball rolls (or comes to a halt). There four events involved: stop, fall, bounce, and rise. Associated with these events are five states: station falling, bouncing, rolling and rising. The graphs are broken up by event, with each event containing two states it mediates as well as the two overlays (if they exist) associated with it. An example graph bounce with a procedural overlay is shown in Figure 2. This event follows the state of falling and res in either a state of bouncing or one of rolling. The physical object involved (physobj to be restricted ball) has an attribute of bounce-energy (the kinetic energy produced as the ball hits the ground
deforms) which enables one of the two states, but not both. The relations contained in the graph that connected to the actor create the time chart shown in Figure 3. In Figure 4 we see the constraint ove for the same graph. The constraint compute-speed-f computes the falling speed of the object given height and is enabled by the state of falling. The second constraint computes the bounce energy from speed and is enabled when the speed changes. Height and speed are both attributes of the physical obj (the ball).

When the graphs for all four events are joined together, they form a program (which is executed to produce the local time maps) for a bouncing ball. Procedurally, the program is a cycle and shown in figure 5. Informally, the state of being stationary enables the fall event which asserts the

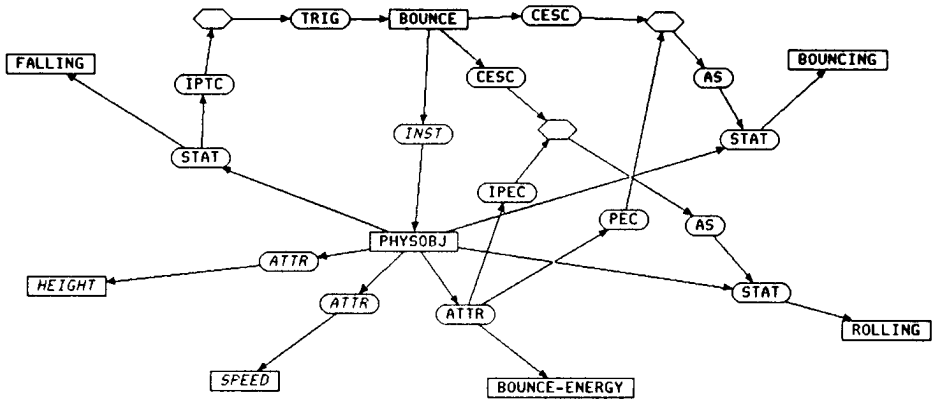

Figure 2. The **graph** for the event 'bounce' with procedural overlay.

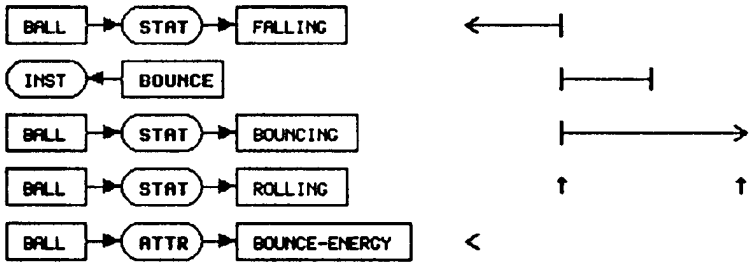

Figure 3. The time chart created from the procedural actors in 'bounce'.

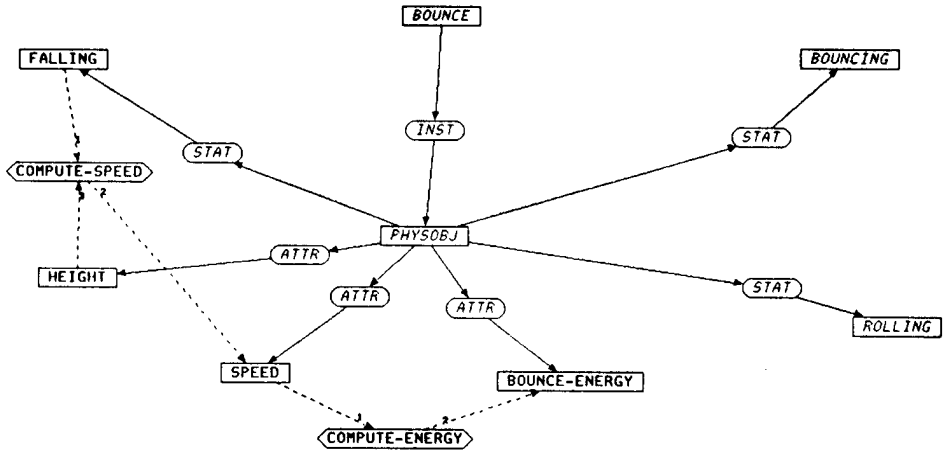

Figure 4. The graph for the event 'bounce' with constraint overlay.

falling state. The falling state enables the bounce event which, depending on the bounce-energy of the object, asserts the rolling or bouncing state. From the figure we see that the actors for rolling and bouncing are connected by inverse relations to the bounce-energy state. If the bounce energy state is enabled it fires the bouncing actor, otherwise the inverse is enabled and the rolling actor fires. If the bouncing state is asserted, it enables the rise event which asserts the rising state. This rising state enables the stop event which completes the cycle by asserting the stationary state. Each of the four event procedural overlays produces a local time map when the program is run. These are all combined by the PRE system to produce the global time map shown in figure 6.

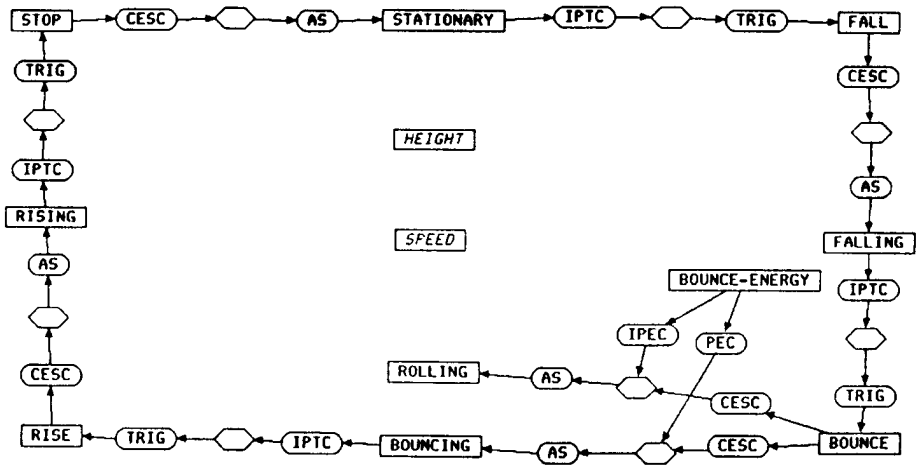

Figure 5. The program cycle for a bouncing ball.

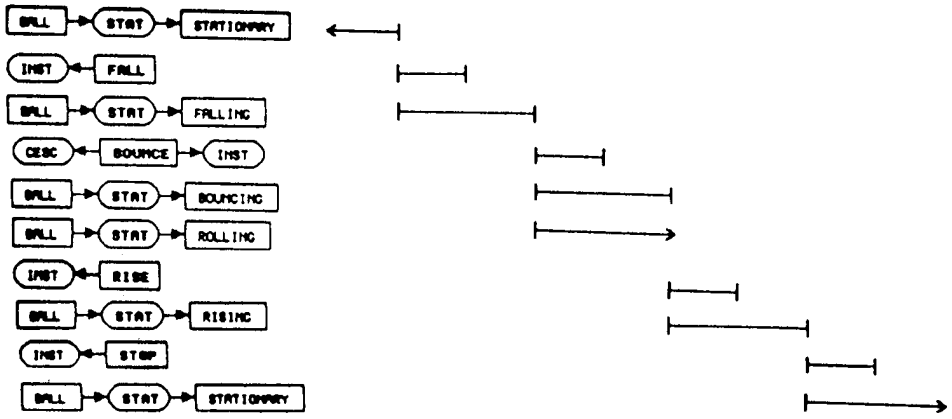

Figure 6. The global time map for a bouncing ball.

Figure 7 shows the constraint network formed by joining the constraint overlays from the event graphs. In this figure the compute-height constraint is enabled by a speed and the stationary state to produce a height. This value for height and the state of falling enable the compute-speed-f constraint to produce a speed. This new value for speed immediately enables the compute-energy constraint since no state is used as input. The state of rising and the value for speed enable the compute-speed-r constraint to compute a new speed value. This does enable the compute-energy constraint to produce a new bounce-energy value which in this case is changed as the constraint network continues to fire by the compute-speed-f constraint. We can see, by following both the constraint network and the procedural network that the values of the attributes changes as we travel around the cycle. The value for speed is decreasing every time the compute-speed-r constraint is fired due to the loss of energy in bouncing. Eventually, the speed reaches a point at which the bounce-energy state is disabled. At this point in the procedural network the actor for rolling is enabled.

It should be noted that the constraints in the above example are behaving like functions and not like constraint satisfiers. A major issue we are facing in implementing true constraints is that of refraction, that is of a constraint firing repeatedly as its inputs change. Normally, a constraint will fire if its referent inputs have changed and if its state inputs are enabled. In a true constraint with no states as inputs, as the constraint compute-energy in figure 7, the constraint may fire repeatedly as the referent value, in this case speed, changes. In the bouncing ball example, all constraints except compute-energy are enabled by states. We could possibly combine the constraints compute-speed-r

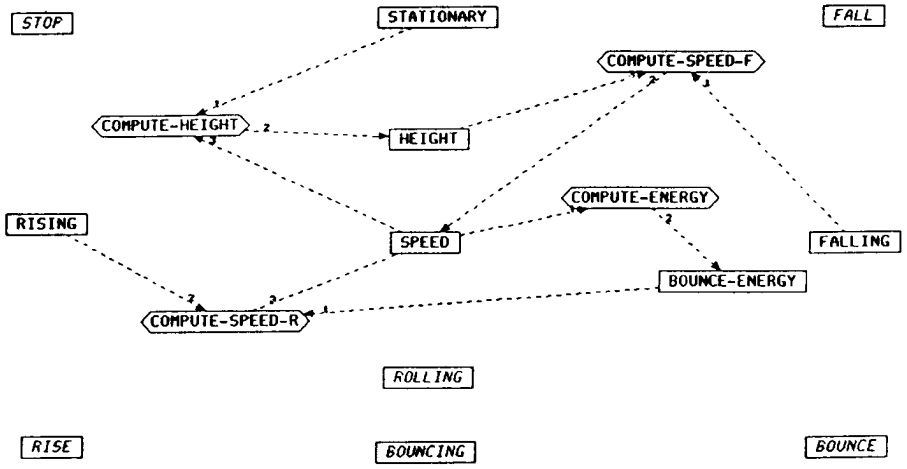

Figure 7. The constraint network produced for a bouncing ball.

and compute-energy into a true constraint satisfier that would give the value of either speed or bounce-energy given the other. We would, however, run directly into the problem of refraction which is solved by breaking them apart and enabling the speed computation separately throughout the rising state.

4. Conclusion

We have shown the two additions to the original CG theory, constraint and procedural overlays, and their integration into the CP environment. Combined with the PRE environment this gives us full capabilities for representing temporal, causal and functional information as well as giving us the ability to do constraint satisfaction. The implementation of these tools along with the canonical operations provided with CG theory is nearly complete. We have been using this environment now for model generative reasoning (mgr), planning, and information fusion.