A Unified Approach to Plan Representation

Daniel P. Eshner and Roger T. Hartley
Knowledge Systems Group
Computing Research Laboratory
BOX 3CRL
New Mexico State University
Las Cruces, NM 88003

ABSTRACT

Planning has traditionally used some form of predicate calculus as the basis for its representation. From the early days of STRIPS in 1970 to the recent planner TWEAK in 1987, domain knowledge and world models that represent the current state of the domain, have been represented as conjunctions of atomic formulae. Whereas this has been adequate for declarative knowledge, the representation of procedural knowledge (i.e. the actions and their effects that are reasoned about during planning) has been largely left to ad hoc schemes of a non-logical nature. In this paper we present a unifying framework based on a knowledge representation system with good expressive power for both forms of knowledge. Domain knowledge, actions and plans are all represented within the same formalism, giving distinct advantages over previous approaches.

1. Introduction

A planning system must be able to represent three things: the domain in which planning takes place, the actions an agent can take in the domain, and the plans that the agent formulates for behaving in the domain. Of the three problems, the representation of actions has received the most attention from the artificial intelligence (AI) community. This is perhaps because much of the power in a planning system is determined by the way in which actions and their effects are represented. The purpose of this paper is to address the above three representational problems within a unified framework. Section 2 discusses previous approaches to the problems. The Section 3 discusses the approaches in regard to procedural and declarative forms. Section 4 describes our Plan Representation Environment (PRE) that is based on the Conceptual Programming (CP) system (Hartley 1988). Finally, we conclude by relating PRE to previous approaches.

2. A Historical Look at Planning Environments

2.1. Domain Representations

There are two basic approaches that have been taken in representing domain knowledge1. The most common approach uses well-formed formulas (wffs) in the predicate calculus to represent all knowledge[1]. (Fikes and Nilsson 1971, Sacerdoti 1974, Sussman 1975, Sacerdoti 1975 and 1977, Tate 1974, Stefik 1980, Chapman 1987). The other approach, using knowledge representation languages based on semantic nets (e.g. KRL -- Bobrow and Winograd 1977), is equivalent in its expressive power to predicate calculus but differs in its epistemology. The use of these knowledge representation languages, the most common being Units (Stefik 80) and PEARL (Wilensky 83), has received attention from some of the more more recent planning work (Tate 1976, Wilkins 1983, Stefik 1980, Wilensky 1983, Mostow and Bhatnagar 1987, Broverman and Croft 1987). There are also some systems that combine the two approaches in order to represent both variant and invariant properties of objects (Wilkins 1983, Wilensky 1983).

The state of the world at any given time is usually represented by a set of propositions that define a world model. Actions (events, operations)[2] change the state of the world by adding and deleting propositions from the world model. It is natural to view the set of all possible world states as a problem space (Newell and Simon 1972). A plan can thus be viewed as a path through the problem space that ends in a goal state. Links from one state to another in this formalism are defined in terms of operators (actions). There are many approaches to finding optimal paths through this problem space, both in AI and in computer science. Korf (1987) discusses planning as search that quantifies the performance of AI techniques like the power of abstraction and heuristic search.

2.2. Action Representations

We will discuss action representations found in planning systems in light of the logical formalisms on which they are based. In order, we discuss those based on predicate calculus, situation calculus, possible worlds and temporal calculus. As discussed above, the domain representations are mostly based on the logical calculus used.

2.2.1. Predicate Calculus Based Approaches

Starting with STRIPS (Fikes and Nilsson 1971), actions are represented as a set of preconditions, an add list and a delete list. The add list is a set of propositions that become true once the action is performed. The delete list is a set of propositions that become false once an action is performed. Together the add list and the delete list may be thought of as the *postconditions* of the action. To represent knowledge that are domain specific, NOAH (Sacerdoti 1977) uses a language called SOUP (Semantics Of User's Problem). The set of SOUP functions have the preconditions and postconditions represented in procedural form. NONLIN (Tate 1976) has a similar language called TF (for Task Formalism) that is a declarative representation of the task specific actions. In ABSTRIPS (Sacerdoti 1974) literals within preconditions have a notion of criticality associated with them that discriminates among various levels of detail. These levels of

---

1 By domain knowledge we refer to objects in the world that are acted upon.

2 As is common in the planning literature, we use these terms interchangeably.

detail define an abstraction hierarchy. Literals that are more critical (those having a larger critical value) are included in a higher level of abstraction. Most major planning systems (Warren 1974, Tate 1974, Sacerdoti 1975 and 1977, Tate 1976, Vere 1983, Chapman 1987, Mostow and Bhatnagar 1987) use the same technique as does STRIPS except for four exceptions.[3]

In HACKER (Sussman 1975) actions are represented as programs that are debugged. There are no actions with general applicability in HACKER, only programs. In MOLGEN (Stefik 1980) operators introduce constraints. Changes to objects are dealt with by renaming them. Different names refer to the same object in different states. Wilensky's PEARL (Wilensky 1983), on which his planner PANDORA is based, has a representation of actions that is based on Schank's Conceptual Dependency (CD) theory (Schank 1975). PEARL uses a Result predicate to link an action to its results. In Wilkins' SIPE (wilkins 83) operators contain first order predicates that correspond to STRIPS preconditions and postconditions (postconditions are referred to as *effects* and can be negated) as well as predicates that indicate the purpose of the action. Each operator also contains a plot that specifies how the action is to be performed. Operators in SIPE also provide for constraint posting, resource specification, and the use of deduction to determine some of the effects of actions.

2.2.2. Situational Calculus Based Approaches

Starting with QA3 (Green 1969) we see the introduction of a situation calculus. This allows actions to be functions on states that yield new states. Although the planners we have discussed do not explicitly use situation calculus, actions have nonetheless been described as functions from one world state to another. This function, in the STRIPS formalism, is implemented with add and delete lists (or the equivalent) but was not uniform with the representation of the domain. Situation calculus was an attempt to support the passage of time without explicit time parameters. Manna and Waldinger (1987) present a version of situation calculus called plan theory. Actions are represented in plan theory as axioms that explicitly specify the new state of an object in terms of situation calculus. Pednault (1987) presents a language called ADL that is a combination of the STRIPS operator language and situation calculus. Syntactically, actions in ADL are represented like those in STRIPS. It is only the addition of situation calculus in ADL that differentiates it. Georgeff (1987) uses a generalized situation calculus in order to reason about actions in multiagent settings. In his model, events in a given state do not uniquely determine the resulting state. The formulation allows arbitrary temporal constraints on world histories not possible with ordinary situation calculus.

2.2.3. Possible Worlds Based Approaches

Moore (1977) presents a logic based on possible worlds which is an adaptation of modal logic (Kripke 1971). Ginsberg (1986) formally discusses the constructions off a

---

3 Some systems use different terminology to refer to the same approach. In NONLIN postconditions are referred to as *effects which have* a positive or a negative sign corresponding to additions and deletions to the world model.

possible world and relates the notion of possible worlds to that of counterfactuals. Informally, given a world model (a collections of facts) and a new fact, we want to construct the most similar possible world model in which the new fact holds. The set of possible worlds includes those in which the new fact does not invalidate any "protected" fact. Ginsberg represents actions in a possible worlds formalism much like STRIPS representations in that an action adds facts to the world. Instead of deleting facts as does STRIPS, however, consistency is maintained deductively using the constraints describing the domain in question (Ginsberg 1987).

2.2.4. Temporal Calculus Based Approaches

With DEVISER (Vere 1983) and the planner in Allen and Koomen (1983) we see the beginnings of temporal calculus based approaches to representing actions. In a temporal calculus the basic unit is a temporal interval. Allen (1981) defines nine primitive relations that can then relate these temporal intervals. Allen and Koomen (1983) use the STRIPS formalism with preconditions and effects being temporally qualified using temporal calculus. **In DEVISER Vere** attaches a duration expression to what he calls a relational production, the equivalent of the STRIPS formalism. The addition of durations on actions allows DEVISER to deal with the start time and durations of parallel plans. Lansky (1987) describes a representation based on temporal logic. This representation centers on events rather than states. A state is simply an *event history*. She shows how temporal logic constraints on these event histories can facilitate the representation of actions in parallel, multiagent domains. Actions are not represented in terms of the states they yield but rather the relationships between events which introduce constraints.

2.3. Plan Representations

As was stated earlier, a plan can be viewed as a path through a problem space. Since actions define the links between the nodes of the problem space, a plan is usually represented as a series of actions. In STRIPS (Fikes et al 1972) this series of actions was represented in a *triangle table.* A triangle table is a lower triangular array where rows and columns correspond to the operators of the plan. These tables are generalized into MACROPs which can be used for execution monitoring or later use in similar problems. In HACKER plans were simply programs. NOAH introduced the notion of a *procedural net* to represent actions and plans. The procedural net is a network of nodes that represent a particular action at some level of detail. Each node points to a body of code (SOUP code) that, when evaluated, causes new nodes that represent more detailed actions to be added to the net. The world model is then updated to reflect the effects of the more detailed actions using the add and delete lists. Nodes are linked in this way to form hierarchical descriptions of operations. Nodes are also linked at each level of the hierarchy in a partially ordered time sequence by predecessor and successor links. Using this procedural net and critics NOAH can delay commitments to a particular order and thus avoid backtracking.

NONLIN uses a similar structure called GOST (GOal STructure). GOST remembers the conditions on any node and the set of possible contributors to those conditions. This allows for simple detection and corrections of interactions between subgoals and for plan monitoring during execution, much like procedural nets in NOAH. DEVISER

also stored plans in a partially ordered network of activities. The final plan network resembles a PERT chart. From this chart a schedule of nominal start times for each activity is generated. TWEAK has at all times an incomplete (partially ordered) plan. Although this incomplete plan may resemble a procedural net, plans are not represented as a directed graph but as a partial order on the steps that make up the plan.

3. Procedural vs Declarative Knowledge

One way to view these approaches to representing knowledge is according to the procedural/declarative distinction.[4] By this we mean the *representation* of knowledge being either procedural or declarative rather than the knowledge itself being inherently procedural or declarative. With the exception of HACKER and SOUP codes in NOAH, the planning systems we have discussed have represented their domain knowledge declaratively. Most procedural information has been buried in the actual inference engine of the planner. For example, the information about a procedure in most systems is based on that of STRIPS (Fikes and Nilsson 71). That is, procedural changes to the world are designated by add and delete lists. The interpretation of these lists is left up to the inference engine thus burying the explicit procedural information of the change, HACKER, on the other hand, represents everything procedurally and thus objects do not exist outside of the procedures that act upon them. Although planning systems provide explicit procedures for the manipulation of declarative knowledge, there is no explicit procedural representation of change. As Sacerdoti describes the deficiency:

> The be aware of the goals and subgoals that the planner has decided to tackle, but it does not preserve any information about the computation that resulted in those decisions (Sacerdoti 1975, p. 214).

We believe a uniform environment should represent both types of knowledge explicitly in a formal way. Our approach using procedural overlays (see section 4.1) does exactly that.

4. The Plan Representation Environment

This section describes our environment for representing knowledge, procedures and plans in a unified framework based on the conceptual programming (CP) environment (Hartley 1987?). CP provides the environment for representing all domain knowledge and actions. The plan representation scheme resembles the procedural nets of NOAH with the inclusion of a temporal calculus (Dean and mcdermott 1987). We first briefly describe CP followed by a description of the plan representation.

4.1. The Conceptual Programming Environment

4.1.1. Declarative knowledge

CP represents knowledge using the conceptual graph formulation of John Sowa (Sowa 1984). Conceptual graphs are bipartite directed graphs with two distinguished node types: concepts and relations. Graphs are of two types, *facts* and *de initions.* Fact

---

4 See Winograd (Winograd 85) for a discussion of this topic.

graphs **represent objects and their relationships, and** have their counterpart in a STRIPS-like formalism as predicate calculus expressions. Definitions are graphs which serve as the meaning of the defined concept. These are embedded in a standard sub/super-type lattice. Although CP can represent definitions of both concepts and relations, only the concept definitions are relevant in planning. Furthermore, CP allows for an Aristotelian (genus and differentiae) form of definition, or a more domain specific form called a *schema.* There can be multiple schemata which express the meaning of a concept in different contexts. These schemata support the representation of both domain knowledge and actions in PRE.

4.1.2. Procedural knowledge

CP definitions are representations of static knowledge. In order to express procedural knowledge, an additional mechanism is needed. This is crucial in a planning domain since the representation of the effects of actions governs the workings of the planning system. Sowa describes actor nodes as behaving like functions embedded within a graph to provide concept referents (i.e. for instantiation). CP allows actors to be much more like "active concepts" which accept states as preconditions and events as triggers. Having been triggered, they assert states and enable further acts as byproducts of their activity. These actors may be used to express *causality,* involving states and events, and *inferences,* involving propositions. Since each actor of this sort is completely specified by its inputs and outputs, there is no need to label the actor box in the graph (see Figure 1). In effect, the actor merely acts as a confluence node for its inputs and outputs, not as a function as in Sowa's original formulation.

In the example in Figure 1 the declarative component of the graph expresses an assertion which in English can be written: "A person x is the agent of an act of giving a physical object, which is in x's possession, to a person y". The addition of an actor can express the change of possession from x to y. The actor is linked to its inputs and outputs via special conceptual relations which collectively express a factorization of the combination of states and events typically found in a theory of action (cf. Rieger's similar treatment in common-sense algorithms: Rieger 1976). All of these relation nodes and the actor nodes they connect to are collected in a special graph called a *procedural overlay.* The declarative and procedural components of a definition are thus stored separately, but in such a way that they can be used together (i.e. overlayed with common nodes in correspondence).

A state is defined canonically as a specialization of the graph [T] → (REL) → [T] and an event as an arbitrary number of joins (on ACT) of specializations of the graph [ACT] → (CASE-REL) → [CASE-TYPE] where CASE-REL encompasses any of the standard set of case relations (AGT, PTNT, INST, SRCE etc.) and CASE-TYPE is restricted to conform to the corresponding case. For example, the AGT relation is, canonically, [ACT] → (AGT) → [ANIMATE] and SRCE is [ACT] → (SRCE) → [LOCATION]. In the example, two inputs are needed: the state of possession of a physical object by person x (via the relation 'TEC') and the event involving x giving it to y, via the relation 'AS', and the de-assertion of the input state. 'TEC' denotes this transitory state. The graphs involving actors are actually abbreviated from graphs containing embedded graphs of type STATE and EVENT. However, the embedded level has been removed, unambiguously, since the nature of actor inputs and outputs are restricted.

The complete graph is given in Figure 2. It can be seen that Figure 1 is cleaner, as long as Sowa's original notation is extended to allow relations embedded in states to be connected to actor relations. The actor relations are necessary to type inputs and outputs according to their temporal relationships.

The epistemology of actors, together with their inputs and outputs is meant to be as simple as possible while still maintaining adequate expressiveness. States are to be thought of as intermediate between events; roughly speaking, states enable events and events cause states. Any actor linked to a relation (via one of the actor relations shown in Figure 3) has a state as input or output; an actor linked to a concept has an event as input or output. Input states can either be triggering, or enabling, and transitory or persistent, in all combinations. Triggering states are ones which cause events directly and the existence of the states is enough to start the event. For example, a switch being Con' causes current to flow as long as there is a current source and a circuit exists through the switch. On the other hand, enabling states have no direct causal link. To continue the electricity example, the presence of a voltage source enables the current to flow, but does not trigger it under normal circumstances. States can also be classified as transitory, meaning that the state disappears when some event stops it, or persistent, when an event has no effect. Thus the presence of a voltage source is persistent with respect to the current flow, but transitory with respect to the failure of the source (say a battery running down). It is also possible that the absence of a state can trigger or enable an event. In this way it is possible to handle negated conditions. For example, a solenoid may be magnetized thus inhibiting a switch from closing. As soon as the solenoid becomes unmagnetized, the switch closure event can begin.

Events are either continuous, when their effects continue after the event ceases, or oneoff when the effects terminate with the event. The flowing of a current in a wire causes a magnetic field to surround it, but as soon as the current stops, so does the field. This is one-off behavior. On the other hand the act of closing a switch causes the current to flow, but it remains flowing after the act has terminated. Ibis is continuous behavior. As with states causing or inhibiting events, events can cause the presence or absence of a state to occur. It is also possible to represent a "lagged response" which occurs on some forms of causality. In fact the current/magnetic field example is one of these, where the field takes a finite time to build, and cannot be thought of as coextensive in time with the current flowing. In particular, after the current is switched off the field takes a small amount of time to collapse again.

Figure 3 shows all the possible combinations of state and event causality, together with their canonical time maps. These simple diagrams show the temporal extent of states and events and their interrelationships. Time proceeds to the right, along the horizontal straight lines. A small vertical bar indicates a definite start or stop instant, whereas an arrow head indicates an indefinite instant. The vertical relationship between the instants (i.e. before, after or equal) characterizes the different relationships between the intervals. The full meaning and use of the actor relations in Figure 3 are described in (Hartley 1988), along with their translation into Allen's temporal relations (as in Allen 1983).

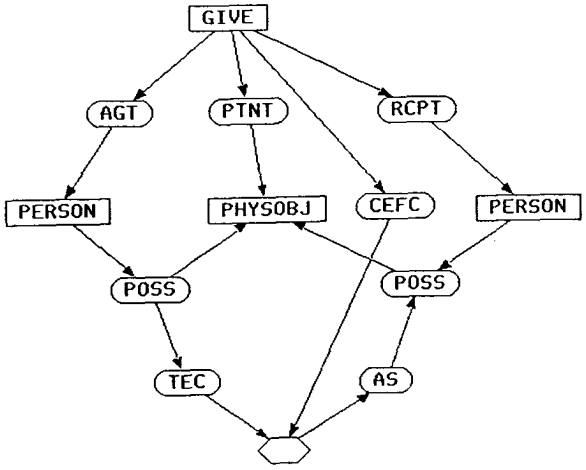

Figure 1. An example of an actor.

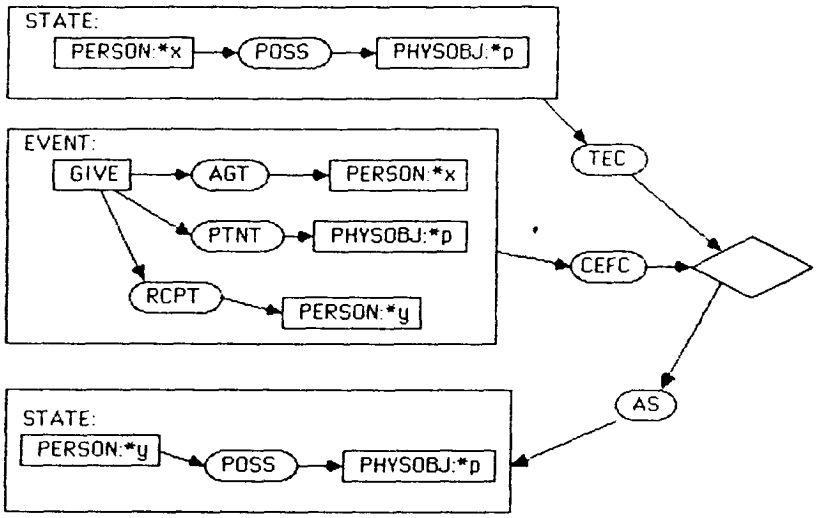

Figure 2. The expanded form of figure 1 in conventional notation.

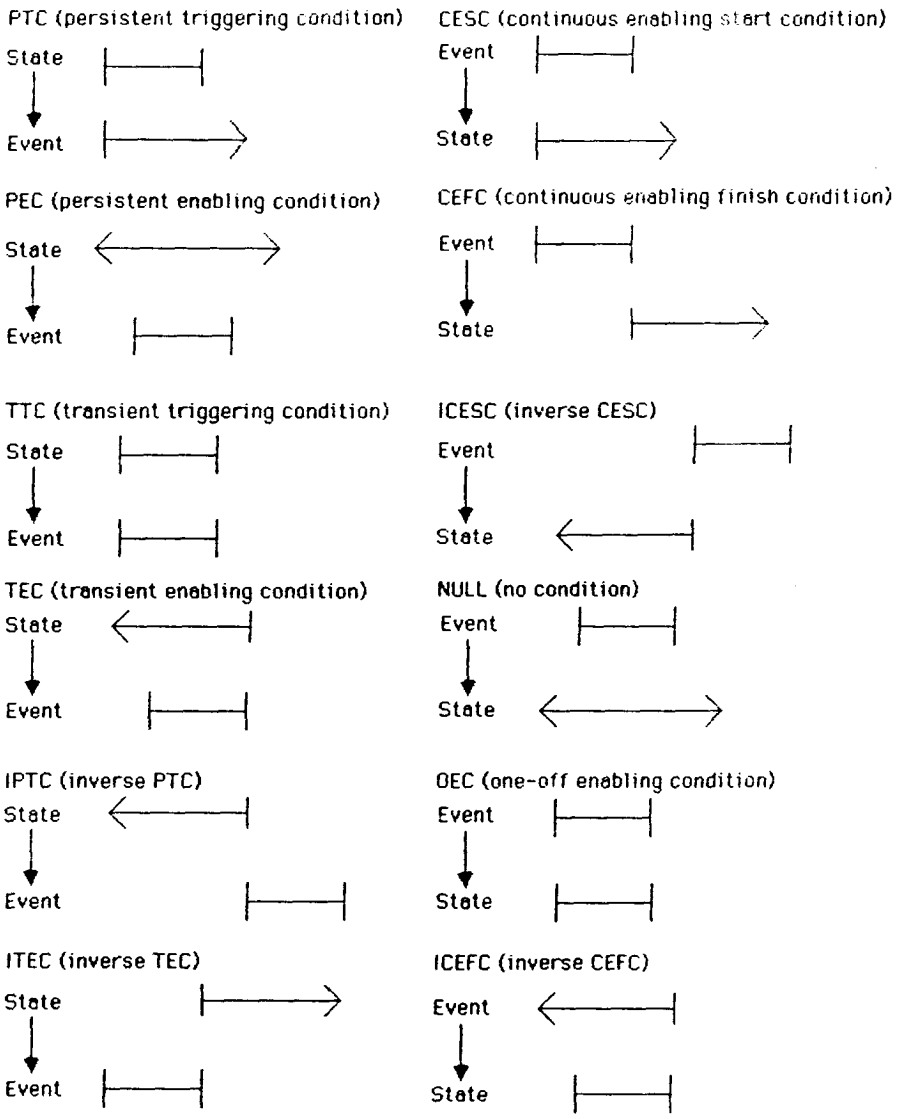

Figure 3. Actor inputs and outputs.

4.2. **Representing Domain Knowledge and Actions**

Both domain knowledge and actions are represented in PRE declaratively using schemata. Actions also have procedure overlays associated with them containing the actor

nodes mentioned above. The use of the coherent representational forms provided by CP gives PRE a uniform way to represent all of its knowledge. Also, representing the action in a declarative format gives PRE the advantage of storing this information in the plan tree for future reference. MOLGEN uses a similar approach based on Units but is the only other major planning system to do so.

4.3. Representing Plans

In addition to the ability to represent objects, agents, and the actions that can be performed, we must be able to group relevant facts into a world model.[5] This world model represents the state of the world at an instant of time. It can thus be thought of as a snapshot of the relevant aspects of the world at a given time. We must then be able to represent a series of these world models in a consistent way that will show the logical and temporal progression of a plan. To accomplish these aims we use a structure called a *plan tree.* This plan tree is essentially a subtree of the entire problem space. The nodes of the tree are called *snapshots* and represent possible world states. The links between snapshots are the actions that transform one world state to another. The linear sequence of snapshots in the plan tree that start from the node (the initial state) to a leaf of the tree that satisfies the goal defines the logical progression of that particular plan. The temporal progression of the plan is defined in a *global time map* (GTM). The three structures used in representing plans, i.e. snapshots, the GTM, and the plan tree, are described in more detail below.

Each snapshot contains a set of facts and a set of actions that operate on this set of facts. Facts are conceptual structures that contain information concerning objects, their current state, and their relationships. Actions are represented by schemata with an associated procedural overlay. This is the mechanism by which PRE represents both the declarative and procedural aspects each action. We can view each snapshot as representing the state of the world at that time instant as well as the relevant actions for transforming that world state to one more closely resembling the goal state. In this way a snapshot represents the decisions made at that instant for transforming the current world state in a goal directed manner.

The GTM represents the temporal progression of actions within a plan. This time map is a combination of all the temporal information contained in the relations between snapshots (i.e. between the world states). Each relation contains a local time map (LTM) that contains an incomplete representation of an actions temporal properties. Within each local time map some state intervals are possibly left indeterminate i.e. with either start time or end time, or both, unknown. The GTM may make these intervals determinate through the interactions between snapshots. The final time map resembles those found in (Dean and Mcdermott 1987).

The plan tree in PRE is represented as a conceptual graph with the nodes being snapshots and the relations between them being the local time maps mentioned above. Figure 4 shows the structure of a plan tree. Each branch from a node represents a different subset of possible actions being applied to the world state. This allows for both interacting goals and multiple agents. Since interacting goals may require a number of

---

5 Our use of the term "world model" is consistent with the use in predicate logic.

actions that interact through objects or through temporal constraints, we must allow for a set of actions to define a transformation to a new world state. Also, since multiple agents may require different actions or different arguments to the same action, again a set of actions must be allowed to transform world states. Since actions in CP form a type hierarchy PRE allows for hierarchical planning as the planning system may choose an action at the appropriate level of abstraction. This is a more formal way, in relation to ABSTRIPS, of representing and planning in abstraction spaces.

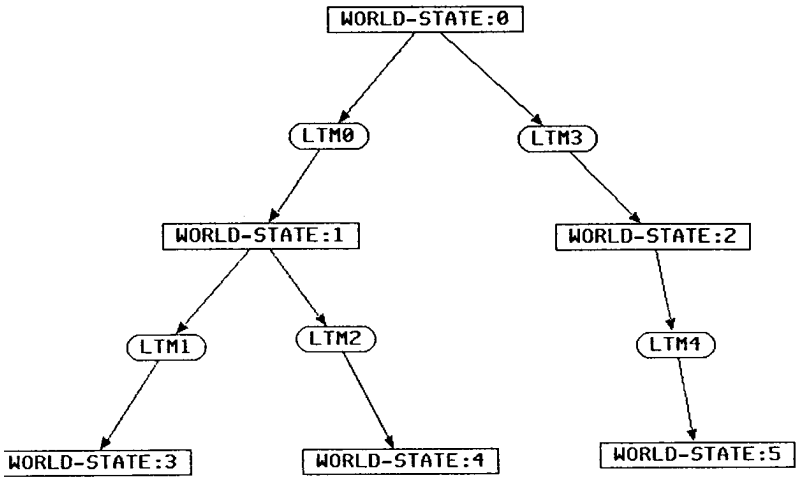

Figure 4. The structure of a plan tree.

5. **Relation to Previous Approaches**

Sowa has shown the notational equivalence of conceptual graphs to predicate calculus. Thus the representation of domain knowledge is really similar to that in all logic based systems. However, since actions are represented in the same framework, PRE has the advantage of using a unified representational structure. Planning systems other than MOLGEN (Stefik 1980) use two different representations for domain knowledge and actions. For instance, STRIPS-like systems use first order logic predicates for domain knowledge and a non-logical representation for actions. MOLGEN, although both domain knowledge and actions are represented within a frame based language, uses simple procedural attachment on frame slots to represent actions. CP allows complex interdependencies among objects using procedural overlays. It thus allows for easier

handling of complex time relationships that exist in events and event sequences. Another advantage to *PRE* is that plans are represented in **the plan** tree using the same formalism. This allows for the formal graph operations provided by CP to be used on entire plans, as well as on all other planning knowledge. Also, NOAH (Sacerdoti 1977) and NONLIN (Tate 1976) provide a separate language (SOUP and TF, respectively) for encoding domain specific knowledge. In *PRE,* this knowledge would also be encoded in CP and could be used interchangeably with domain independent knowledge. Most recent planning research deals with the complexities of temporal relationships between actions. PRE is able to make use of the temporal mechanisms in CP and thus temporal concerns are handled within the same unified framework.

6. Conclusion

We have presented a unified environment for representing domain knowledge, actions and plans using conceptual programming and shown its advantages over previous approaches. We believe much of this advantage is due to the fact that the system is created as a general representation environment as opposed to approaches stemming from the needs of a planning system. We are currently designing a general purpose planning system to take advantage of this environment. This future work builds on experience with a robot task planner for a mobile inspection robot (Eshner et al 1987). This planner utilized an early version of CP to good effect.

References

Allen, J.F., An interval-based representation of temporal knowledge, in: *Proceedings IJCAI-81*, Vancouver, BC (1981) 221-226.

Allen, J.F., Maintaining knowledge about temporal intervals, Comm. ***ACM, 26(11),*** (1983) 832-843.

Allen, J.F. and Koomen, J.A., Planning using a temporal world model, in: *Proceedings IJCAI-83,* Karlsruhe, F.R.G. (1983) 741-747.

Bobrow, D.G. and Winograd, T., An overview of KRL, A knowledge representation language, *Cognitive* Sci. 1 (1) (1977) 3-46.

Broverman, C.A. and Croft, W.B., Reasoning about exceptions during plan execution monitoring, in: *proceedings* AAAI-87, Seattle, WA (1987) 190-195.

Chapman, D., Planning for conjunctive goals, *Artificial Intelligence 32* (1987) 333-337.

Dean, T.L., and McDermott, D.V., Temporal data base management, *Artificial Intelligence 32* (1987) 155.

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

Fikes, R.E., and Nilsson, NJ., STRIPS: A new approach to the appplication of theorem proving to problem solving, *Artificial Intelligence 2* (1971) 198-208.

Fikes, R.E., Hart, P.E., and Nilsson, NJ., Learning and executing generalized robot plans, *Artificial Intelligence 3* (1972) 251-288.

Georgeff, M.P., Actions, processes, and causality, in: *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* Timberline Lodge, Timberline, OR (1987) 99-122.

Ginsberg, M.L., Counterfactuals, *Artificial Intelligence, 30* (1986) 35-79.

Ginsberg, M.L., Possible worlds planning, in: *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* Timberline Lodge, Timberline, OR (1987) 213-243.

Green, C.C., Application of theorem proving to problem solving, in: *Proceedings IJCAI-69*, Washington, DC (1969) 219-239.

Hartley, R.T., Conceptual Programming: Foundations of problem solving, in: Sowa, J.F., Foo, N.Y. and Rao, A.S. (Eds.), *Conceptual Graphs for Knowledge Systems* (Addison Wesley, New York, NY (in press)).

Korf, R.E., Planning as search: A quantitative approach, *Artificial Intelligence 33* (1987) 65-88.

Kripke, S., Semantical considerations on modal logic, in Linsky, L. (ed.), *Reference and Modality*, (Oxford University Press, London, 1971).

Lansky, A.L., A representation of parallel activity based on events, structure, and causality, in: *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* Timberline Lodge, Timberline, OR (1987) 123-159.

Manna, Z. and Waldinger R., A theory of plans, in: *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* Timberline Lodge, Timberline, OR (1987) 11-45.

Miller, D., Firby, RJ. and Dean, T., Deadlines, travel time, and robot problem solving, in: *Proceedings* IJCAI-85, Los Angeles CA. (1985) 1052-1059.

Moore, R.C., Reasoning about knowledge and action in: *Proceedings IJCAI-77,* Cambridge, MA (1977) 223-227.

Mostow, D.J. and Bhatnager N., Failsafe -- A floor planner that uses EBG to learn from its failures, Rutgers University Technical Report ML-TR-17, 1987.

Newell, A. and Simon, H.A., *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).

Pednault, E.P.D., Solving multiagent dynamic world problems in the classical planning framework, in: *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* Timberline Lodge, Timberline, OR (1987) 47-82.

Rieger, C., An organization of knowledge for problem solving and language comprehension, *Artificial Intelligence, 7* (1976) 89-127.

Sacerdoti, E.D., Planning in a hierarchy of abstraction spaces, *Artificial Intelligence 5* (1974) 115-135.

Sacerdoti, E.D., The nonlinear nature of plans, in: *Advance Papers IJCAI-75, Tbilisi,* U.S.S.R. (1975) 206-214.

Sacerdoti, E.D., *A Structure for Plans and Behavior* (American Elsevier, New York, 1977).

Schank, R.C., *Conceptual In*

.formation Processing (North Holland, Amsterdam, 1975).

Stefik, M.J., Planning with constraints, Ph.D. Thesis, Stanford University, Stanford, CA, 1980.

Sowa, J.S., *Conceptual Structures* (Addison Wesley, Reading Mass., 1984).

Sussman, G.J., *A computational model of skill acquisition* (American Elsevier, New York, 1975).

Tate, A., INTERPLAN: A plan generation system which can deal with interactions between goals, Machine Intelligence Research Memorandum MIP-R-109, University of Edinburgh, Edinburgh, 1974.

Tate, A., Project planning using a hierarchic nonlinear planner, Department of Artificial Intelligence Research Rep. No. 25, University of Edinburgh, Edinburgh, 1976.

Vere, S.A., Planning in time: Windows and durations for activities and goals, in: *IEEE Trans. Pattern Anal. Mach. Intell. 5* (3) (1983) 246-267.

Warren, D.H.D., Generating conditional plans and programs, in: *Proceedings AISB Summer Conference,* University of Edinburgh, Edinburgh (1976) 344-354.

Wilensky, R., Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding, *Cognitive Sci. 5* (1981) 197-233.

Wilensky, R., Planning and *Understanding: A Computational Approach to Human Reasoning* (AddisonWesley, Reading, MA, 1983).

Wilkins, D.E., Representation in a domain-independent planner, in: *Proceedings IJCAI-83* Karlsruhe, F.R.G. (1983).

Winograd, T., Frame representations and the declarative/procedural controversy, in: Brachman, RJ, Levesque, HJ. (eds.), *Readings in Knowledge Representation,* (Morgan Kaufmann, Los Altos, CA 1985).