Contingent Plan Structures for Spacecraft

M. Drummond, K. Currie, and A. Tate
University of Edinburgh
Edinburgh EH1 1HN, United Kingdom

1. Abstract.

Most current AI planners build partially ordered plan structures which delay decisions on action ordering. Such structures cannot easily represent contingent actions. ~~This paper presents~~ a representation which can. The representation has some other useful features: it provides a good account of the causal structure of a plan, can be used to describe disjunctive actions, and it offers a planner the opportunity of even *less* commitment than the classical partial order on actions. The use of this representation is demonstrated in an on-board spacecraft activity sequencing problem. Contingent plan execution in a spacecraft context highlights the requirements for a fully disjunctive representation, since communication delays often prohibit extensive ground-based accounting for remotely sensed information and replanning on execution failure.

2. Introduction.

Plan generation isn't problem solving. Planning problems are physical realities which require physical solutions. Planning can only be construed as problem solving when it's part of a larger system which also addresses plan execution; only execution can realize the solution a plan specifies. We use this theme of plan execution to bring together some important issues in AI planning. We consider least commitment plan construction, the representation of teleological information, disjunctive plans, and contingent plan execution in realistically complex domains.

We begin in the next section by briefly discussing the way that most AI planners operate. Commonly used techniques include least commitment action ordering and object selection; we discuss both. Following this, in section 4, we describe an actual planner called O-Plan [1] which uses these techniques to good effect. We cover the essentials of O-Plan's search for an acceptable plan, leaving aside low level details. This discussion is used to show how O-Plan relegates the responsibility for reasoning about disjunctive actions to its search space management component. We argue that what a planner needs is a plan structure which is able to describe the disjunction of action implied by the choices encountered during plan construction. In section 5 we present a solution to the problem. A representation is given which has the properties we seek: it can be used to do least commitment plan construction; it explicitly represents teleological information; *and* it can describe disjunctive actions. Together these abilities allow our plans to be used for plan execution in realistically complex domains. To motivate this, section 6 places the ideas in the context of a spacecraft activity sequencing problem: planetary observation. This example causes us to reflect on the basic principle of least commitment problem solving in general, since it supports a form of least commitment reasoning which commits *even less* than current techniques.

The primary result of this paper is a representation we call *C-Plans*. We claim that the representation is suitable for use in sequencing the activities of automated spacecraft. Further applications-oriented research is required to substantiate this claim.

3. Current AI planners: least commitment plan construction.

An AI planner is given the responsibility of constructing a plan of action. Such a planner is given an initial state description, a set of goals, and a set of action schemas. The schemas are parameterized plans, suitable for solving limited problems. A plan produced by the system is an artifact built from individual operators, appropriately instantiated and ordered. This plan must be sanctioned by the system as a feasible means of achieving the given goals. In this section we examine briefly two of the main operations required to produce this plan: action ordering and variable instantiation.

3.1. Action ordering.

Early planners built totally ordered structures: a plan was a *sequence* of actions. This not only applied to the final plans produced by the system, but also to the partial plans that were built during search. With Sacerdoti's NOAH [3] system this all changed. NOAH built plans as *partial* orders on actions. This meant that it was possible for any two given actions to be unordered with respect to each other. The intuition behind this idea is that a partial order on actions characterizes very many total orders. Today such plans are often called *nonlinear*. It might seem that a system which builds nonlinear plans would be exponentially more efficient than one which builds linear, or totally ordered plans. Unfortunately this has never been proven. Only the intuition exists that nonlinear is better than linear; but this intuition is better than nothing. See Chapman [4] and Drummond [5] for more on this.

3.2. Object selection.

There is another sort of least commitment found in some planners which relates to the way that the objects referred to in plans are selected. Variable instantiation is the process of selecting constants to bind to variables. Each variable can be bound to a specific constant, or unbound, meaning that no constant has yet been selected as appropriate. But it is possible to operate in a more sophisticated way: we can *post constraints* on the permissible constants for any given variable. In this way, we constrain the possible bindings for a variable, rather than select one as correct outright. Information can be gathered during the process of plan construction which leads to the deletion of particular constants from the set of possibilities. The hope is that eventually the set of possibilities will be narrowed to one alternative, or reduced to the empty set, indicating that the constraints posted on the variable are so strict that there is no satisfactory object. This method of associating constants with variables is known as *least commitment object selection*. Its genesis was in Molgen [2].

4. A framework for doing this: O-Plan.

O-Plan is a modern planning system which owes many of its ideas to NonLin [6]. NonLin derives from NOAH, and extends it in many ways. For our purposes the essential contribution of NonLin is its completion of the search space of partial plans: NonLin could find plans that NOAH could not. This is because NonLin had plan modification operations available to it which defined its search space of partial plans so as to include plans that NOAH would never consider. O-Plan inherits its definition of the search space from NonLin. In the next section we consider the mechanisms used by O-Plan to search the space of possibilities. We explain how it keeps track of alternatives, and how it searches through the space of partial plans. We then go on to consider how this mechanism can be extended through a more flexible representation for plans.

4.1. Agenda-based partial plan search.

The planning components in the O-Plan framework employ various techniques to lessen the amount of potential search in any particular application. The techniques include least commitment variable binding, constraint cut-off, temporal coherence and various heuristic functions. O-Plan searches through a space of partial plan states, guided by these techniques, where each partial plan state is derived from the application of a plan modification operator to some current partial plan. This is essentially a search space of plan modifications, or operations whose application results in a new (partial) plan state.

An O-Plan Plan State is a structure of some detail and it includes the partial order of activities that is currently being built (essentially the plan so far), a log of effects and conditions asserted or required in the plan, the teleological information used during plan generation (and available thereafter), variables used during planning and, finally, information on outstanding tasks generated during the planning process. These pending tasks are collected together into agenda lists from which each task can subsequently be scheduled in some opportunistic fashion. This mechanism provides for a dynamic approach similar to that provided by "blackboard" based systems.

In practice there are two main agenda list types, one for task specifications which are fully instantiated, and one for task specifications where certain information has yet to be determined. A third "alternatives" agenda list is currently employed which should eventually disappear, but which has been used in the absence of a complete method for dependency-based plan repair.

Task selection is done under the control of a scheduler, which provides the opportunism for the overall process. This scheduler can be regarded as a "plug-in" module in the O-Plan system and therefore it can reflect various scheduling strategies. The scheduling of a task from the agendas causes a handler (or knowledge source) to process that particular task. The relevant handler is invoked by the type of the task scheduled hence the system is data driven by the tasks themselves. As well as changes to the current Plan State, the processing of a task generally results in the creation of new tasks or the amendment of existing tasks on the agendas.

When choices are made, the task handlers have the option to either post dependency information in the Plan State, or to simply spawn alternative Plan States via the alternatives agenda mentioned earlier. The former method has the advantage that it offers the potential for proper plan repair where only the affected parts of the current (partial) plan are stripped off after a failure, while useful parts are saved and the work done in producing them protected. We are researching how this can be done by using partial plans augmented with teleology information, although there are many outstanding problems.

Processing proceeds in cycles and finishes when all tasks have been processed or when there is reason for a particular task to terminate planning. In theory the handlers are independent of one another but they do have the ability to "poison" the current Plan State if they detect inconsistency or constraint violations. This is the time to backtrack, plan repair or simply give up. Search through the space of partial plans is therefore controlled by the scheduler which chooses the next best thing to do, using information provided by the tasks generated during planning. More detail of the O-Plan control structure can be found in [1].

4.2. The requirement for truly disjunctive structures.

O-Plan searches through a space of partial plans. When there's a choice that cannot be delayed, the current O-Plan task scheduler pursues *one* of the available options by incorporating it into the current Plan State. On failure, O-Plan may reconsider all previous Plan States on the alternatives agenda and pursue a previously ignored plan modification operation. In this way it follows a "one-then-best" search strategy as in NonLin.

An alternative approach is demonstrated by the following scenario. Consider that at some point during its search for an acceptable plan, the system identifies an outstanding goal, *G*. Assume that there are two action schemas which after analysis appear suitable for achieving *G*. The traditional approach says that this choice induces a bifurcation in the search space, each path considering one of the two possible actions. However if our developing plan is able to represent disjunction, such a bifurcation is unnecessary. *Both* possible actions (resulting from instantiating the schemas' variables) can be installed in the plan. The only requirement is that the plan record the fact that these two actions stand in a disjunctive relationship.

By the above discussion we aren't suggesting that a planner consider all possible options at each point in its search; such behavior is doomed to failure, since the number of options open will inevitably be huge. Much of the information needed for later planning also becomes uncertain in a plan with too much disjunction. However if the plan representation is able to describe disjunction, then the system will have the *option* of including action disjunction as appropriate.

Contingent plans are also necessary for doing realistic plan execution monitoring. When a plan is generated, it's unlikely that the generation component can guarantee what the world will be like when plan execution begins. To properly handle this we need disjunctive plans. The planner can produce plans which contain actions to deal with whatever contingencies it deems worth considering. Such a contingent plan must specify the conditions under which each of the planned actions is appropriate, to allow the execution component to correctly select which action to execute.

So: we would like to formalize a plan structure able to represent disjunction of action. But in doing this there's a trap to avoid. We could easily over-simplify the data structures used by a system such as O-Plan. It would appear possible to formalize a nonlinear plan as a partially ordered set. Mathematically all one requires is a set of actions and an ordering relation over that set. (See [4] for an example.) The ordering relation is required to be irreflexive and transitive, therefore asymmetric. The problem with such a simple formalization is that is fails to capture much of the information that O-Plan exploits during plan generation. In particular, it does not capture the *goal structure* of a plan [7]; that is, the causal structure that exists among the planned actions.

There are other requirements on the formalization that we won't consider in this paper. In particular, we won't address formalizing least commitment object selection. Data structures to support such operations are simple to formalize, but for ease of exposition, we won't do it here. It is straightforward to add this to the formalism we present.

5. Formalizing contingent plans.

We can borrow some notions and notation from Net Theory [8]. Not all the constructions that we need are part of net theory, so we'll have to add a few bits onto the basic framework. We won't motivate our additions; for a brief discussion, see [5], and for more extensive motivation [9]. Essentially, we use Condition/Event systems augmented with event occurrence preference orderings; we also identify the conditions and events of the system with predicates of a simple language. In this section, we'll proceed by informally defining the constructs of our plan language, building up the overall structure we require. The eventual goal is to define *C-plans*, or Contingent Plans, following on the arguments above. It is possible to be quite formal in defining these C-plans, but this paper simply explains and motivates them.

5.1. Basic C-plan structure.

A *proposition* is a functor applied to arguments. A *functor* is written in lower case, followed by its arguments in parentheses. *Arguments* are variables or constants; we allow infinitely many of each. Variables are written in upper case, constants are written in lower case. For example, both *on(a,X)*, and *skew-platform(15,left)* are propositions.

Propositions are identified with what we call *b-elements* and *e-elements*. A b-element is intended to denote a condition in the world, and can be true or false. For instance, the b-element *clear(c)* under a blocks world interpretation is true if and only if the block denoted by *c* has nothing on its upper surface. Propositions are also identified with e-elements. An e-element is intended to denote an action, the occurrence of which changes the holding of certain conditions.[1] For instance the e-element *move(a,b,c)* in a blocks world context might denote the action of moving the block denoted by *a* from the block denoted by *b* to the block denoted by *c*. Certain conditions must hold if this action is to occur; furthermore, when the action does occur, certain conditions in the world will no longer hold, and certain others which did not hold will begin to do so. For example, in the case of the block movement we might expect that *a* can only be moved from *b* to *c* if *a* is initially on *b*. Following the movement, *a* will be on *c*. We need to capture these condition-action relationships in our plan representation.

To do this we introduce the notion of a *flow relation*. A flow relation is a set of ordered pairs, each pair in the set ordering either a b-element and e-element, or e-element and b-element. The ordering of a b-element and e-element is interpreted as an *enable* relation. Thus, the holding of certain conditions is understood to enable the occurrence of certain actions. The ordering of an e-element and b-element is interpreted as a *cause* relation: actions can cause the holding of certain conditions. The flow relation describes the relationship between any given event and that event's enabling conditions and effects. It captures what O-Plan and NonLin call *Goal Structure*; the best dictionary word for this concept is probably *teleology*. We use the word to refer to the *reasons* for some event or condition being included in a plan. The flow relation of a net allows a formal analysis of which actions can be used to enable which other actions; this is essentially the reasoning that O-Plan performs to generate a plan. Other modern planners, such as SIPE [10] also include such information in their plan data structures.

We will refer to the b-elements which are ordered immediately before an e-element as that e-element's *preconditions*; similarly, we will refer to the b-elements ordered immediately after it as its *postconditions*.

Graphically we present b-elements as circles and e-elements as squares. Each circle is labeled with the proposition which is the b-element, and each square is labeled with the proposition which is the e-element. The flow relation is drawn as arcs from circles to squares and from squares to circles. If an arrow is to go from a circle to a square, and another from the same square to the same circle, we draw only one line, and use an arrow-head on each end of the line to indicate the two arcs.

One other ordering relation is needed to complete the basic C-plan structure. This is the *before* relation, used to constrain the way that a net can execute. Intuitively, the before order is a specification of which events must occur before which other events if a plan is to run to its intended completion. We often refer to the *before* relation as *execution advice*. Consider: the cause and enable orderings in a C-plan's flow relation describe what is causally possible. But in planning we are often interested in only one of generally many causally permitted execution sequences. Causal orderings will not always uniquely constrain a set of actions to describe just those behaviors

[1] We use the terms "action" and "event" interchangeably, as convenient.

which achieve a planner's overall goals.

A classic example of this occurs in blocks-world tower construction problems. For example: given the problem of creating a tower with block *C* on the bottom, block *B* in the middle, and block *A* on top, the plan construction reasoning must order the two required stack actions to reflect its overall goals. To see this, assume that all blocks are initially clear and on the table. If a plan calls for stacking *A* on *B*, and *B* on *C*, then *both* stack actions are enabled in the initial state. It is not an ordering enforced by *causation* that requires the stacking of *B* on *C* before *A* on *B*. Rather, it is the agent's intention regarding overall plan execution outcome that directs the sequencing of the two actions.

So a C-plan is defined by specifying a set of b-elements (which denote the conditions of interest in the domain being modelled), a set of e-elements (which denote the relevant actions), and an ordering relation on the members of these two sets (technically, the relation is bipartite, since it orders members of two different sets). The C-plan is augmented by giving some execution advice for causally underconstrained actions. This advice takes the form of an ordering relation on C-plan e-elements. To keep the graphical presentation of C-plans simple, we do not draw arcs between e-elements ordered in the execution advice. Instead, the ordered pairs are simply listed beside the net.

A simple blocks world plan basically compatible with what we have defined here can be found in [11].

5.2. C-plan projection.

We now have to say something about the *projection* of a C-plan. A projection is a structure which supports reasoning about the behaviors that a C-plan describes. First we must say something about the conditions under which events can occur and what changes they realize by occurring. Second we must build up the projection structure which describes the overall behavior of a C-plan, using the definition of individual event occurrence as a building block. E-element occurrence can be used as a "state generator" to create a state-space account of the behaviors permitted by a plan.

We will call an arbitrary set of b-element propositions a *case*. We interpret such a set of propositions as a partial description of a state of the world. If a proposition is in a case, then it is true; if it is not in the case, then it is false. Graphically, we present cases only in terms of C-plans -- when doing so, we place a dot (a *token*) inside each and only those circles labelled with propositions in the plan which are also in the given case.

We can use this idea of a case as a partial world state description to define when an individual e-element is enabled; that is, when the action it denotes is allowed to occur. To model this, we can say that an e-element is *enabled* in a case if and only if its preconditions are a subset of the case, i.e., if the enabling conditions of the event are true. We also require that none of the e-element's postconditions are already in the case, unless they are also preconditions. Further, we can specify how the world is changed under the occurrence of the action, by defining how an e-element's enabling case is modified to gain a successor. We can generate a new case through the *occurrence* of an e-element: the new case is defined to be the old one, minus all the e-element's preconditions, plus all the e-element's postconditions. The effects of an event are made true in the successor case, and the enabling conditions are made false. If a precondition is not made false by the occurrence of an action, one need only make the relevant b-element a postcondition of the e-element as well.

This definition of e-element occurrence can be used to build up a state-space graph structure which tells a story about the possible behaviors of a C-plan. Given an initial case and a C-plan, we can build up a *projection graph* as follows. The initial case is used as the starting node of the projection graph. E-elements of the given C-plan are repeatedly applied in non-terminal projection graph cases until there are no more cases in which any of the C-plan's e-elements have concession. Arcs leading from node to node in this graph are labelled with e-elements. An arc directed from one node α to another node β indicates that the e-element labelling the arc has concession in the case contained in α and under occurrence, produces the case contained in β.

The idea is that the graph structure defined in this way contains a given initial case as its starting node, and that each node in the graph contains a case reachable under e-element occurrence. With the interpretation of a case as a partial description of the world, the projection gives us a prediction of what a C-plan can do in terms of the possible world states it might give rise to. The initial case describes the "current" state of the world, and cases in the graph reachable from the initial case describe future possible world states. The arcs in the graph denote transitions from one world state to another, and these transitions can be realized through the actual execution of the actions that correspond to the e-elements labelling the arcs.

So the nodes of our projection graph contain cases, and the arcs are labelled with steps. We can map this structure onto the classical AI picture of planning as follows. The first node of our projection is the initial state given in the problem specification. In order to represent a solution to the problem, the plan's projection must give rise to a node which contains the required goals. We can say that a C-plan is a *potential solution to a planning* problem if it is applicable in the initial case of the problem, and under projection gives rise to a case which contains the given goals. Also, a particular case reachable under e-element occurrence in a partially developed C-Plan can be used for "Question Answering" operations in the planner during plan generation.

Using the idea of projection we can now say something more precise about a C-plan's execution advice. Recall the basic idea. Execution advice must contain the information required to remove harmful residual non-determinism. The advice should not restrict legitimately causally independent actions from occurring concurrently, but it should prevent planned actions from occurring in an order permitted by the causal structure of the plan but unintended by the planner. We can explain the meaning of a plan's execution advice by interpreting it as a guide to navigation through the projection structure. Basically, we say that a C-plan's execution advice is *sound* (with respect to a given problem specification) if and only if for all choice points in the projection, if there is *any* hope for success at the choice point, then either all choices lead to success, or for each choice point that could lead to failure, there is advice about another possible alternative, such that the suggested alternative *can* lead to success. In essence, when there is still hope for success the advice prevents the wrong sequencing choice from being made. To achieve this the advice must prescribe an order on e-elements which prevents certain paths through the projection from being considered at execution time.

It is possible to generalize the projection we have defined to deal with least commitment reasoning about action ordering. To do this, one need only say when a set of e-elements are causally independent, and use this definition to specify when sets of e-elements can be applied to a case, in bulk, to derive a successor. If this is done the arcs of the projection graph are labelled with *sets* of e-elements which describe the parallel occurrence of the denoted actions. This means that if some events are causally independent the e-elements which describe them can be applied as a set, and reasoning can continue from the resulting case.

6. A spacecraft activity sequencing example.

This section presents an example problem and its representation using C-plans. This problem would be difficult if not impossible to represent using the classic partially ordered structures found in systems like NOAH and NonLin.

The basic scenario for the example is as follows. While on a deep space mission, a spacecraft is to pass very close to the planet *Jinx*. Earth-based observation has determined that two weather systems obtain on Jinx: crystal clear skies and turbulent sand storms. While it isn't known exactly what conditions will hold when the spacecraft arrives, it is certain to be one of these two. So useful observations can be made regardless of the atmospheric conditions. If the atmosphere is unclouded, then visible light pictures should be taken. If a sand storm is in progress, then infrared pictures will be most effective.

The camera used for visible light and infrared pictures is the same, so it is impossible to take a visible light and infrared picture in parallel. An initialization step is required in order to prepare the camera for visible light or infrared work. Regardless of the sort of picture taken, a digital image is stored in a frame buffer on board. The frame buffer is only large enough to store one picture. Each time a picture is written to the frame buffer by the camera, a transfer operation must free the buffer by copying the information to an on-board tape storage medium. For this simple example, we do not address the problem of transferring the stored images back to Earth.

It would be nice to avoid specifying an observation programme rigidly in advance. Since Jinx is too far from Earth to permit the up-loading of an appropriate command sequence (using information gathered closer to the encounter) it is preferable to be opportunistic, and exploit the atmospheric conditions which obtain when the spacecraft arrives. During the period of contact, conditions may change, and the pictures being taken should reflect current opportunity.

From an AI planning perspective, the problem is to have a plan which represents the disjunctive observational requirement simply and economically. Notice that it is not a problem to have an on-board computer which runs a contingent program during the Jinx encounter phase. In principle, the program could be written in any language whatever, compiled, and up-loaded to the spacecraft well in advance. But for an AI planner the problem is one of representing the disjunction in a way that permits reasoning about a plan, since the plan will form part of a lager scenario with unexpected events and changing requirements. We give a C-plan which does this. It specifies what

each of the individual observation operations are and the conditions under which they are to be carried out.

The plan of figure 1 is projected in figure 2. The projection describes the behaviors that are possible for the plan. Each arc in the projection is labelled with an integer as used for each event in figure 1. Notice that for this example no execution advice is required. See [11] for an example of how this ordering relation is used.

The plan describes the following behaviors. While it doesn't matter what conditions obtain when the spacecraft arrives at Jinx, assume for the sake of argument that: the spacecraft camera is initialized for infrared work; that the weather on Jinx is clear; and that the frame buffer is empty. A case which describes these conditions is contained in the projection node *S1*. Two events are possible, as described by the C-plan's e-elements *clouding* (2) and *setup(vis)* (4). *Clouding* (2) denotes the event of the atmosphere becoming clouded by a storm. The *setup(vis)* (4) e-element denotes the action of configuring the camera to take visible light pictures. Similarly, the e-element *setup(ir)* (3) denotes the action of configuring the camera to take infrared pictures.

There are two tight cycles in the projection, one between *S2* and *S3*, and one between *S4* and *S5*. These cycles model the normal behavior of the plan during a period when the atmosphere is in a stable state. A transition from *S2* to *S3* models the action of taking a visible light picture, and a transition from *S3* to *S2* models the action of transferring the picture information from the frame buffer to tape. Likewise, a transition from *S5* to *S4* models the action of taking an infrared picture, and a transition from *S4* to *S5* models clearing the frame buffer to tape. All other transitions in the projection can be easily read as setup actions in response to changes in the planet's atmosphere.

7. Conclusions.

There is a relationship between the choices of action schema to achieve goals at plan generation time, and contingent plans which support flexible plan execution. Plan generation is reasoning about goals and the means to achieve them. Plan execution is about actually realizing these promised goals. If fast, efficient, and flexible AI planning systems are to ever exist, they must strike a balance between reasoning about disjunction in advance, and

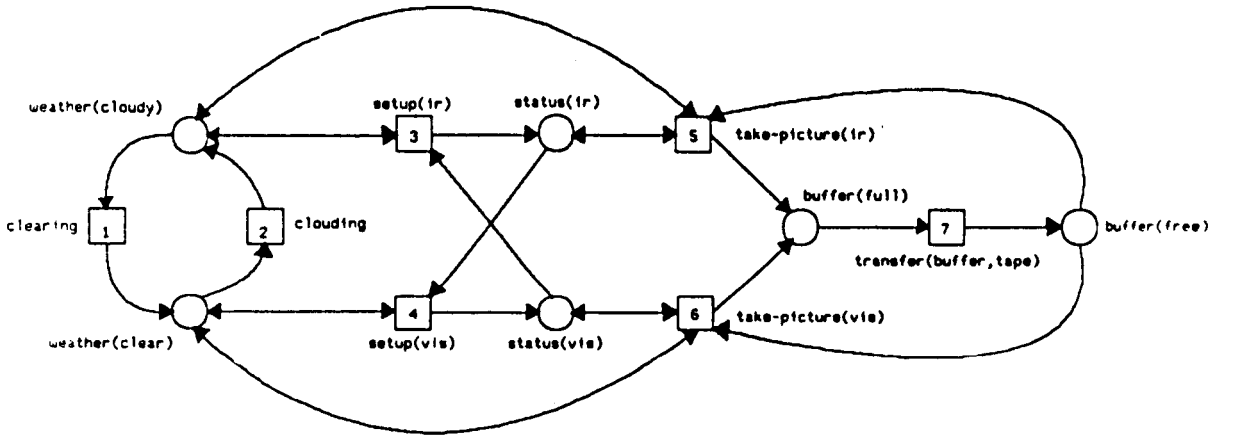

Figure 1: A contingent plan for taking visible light or infrared pictures.

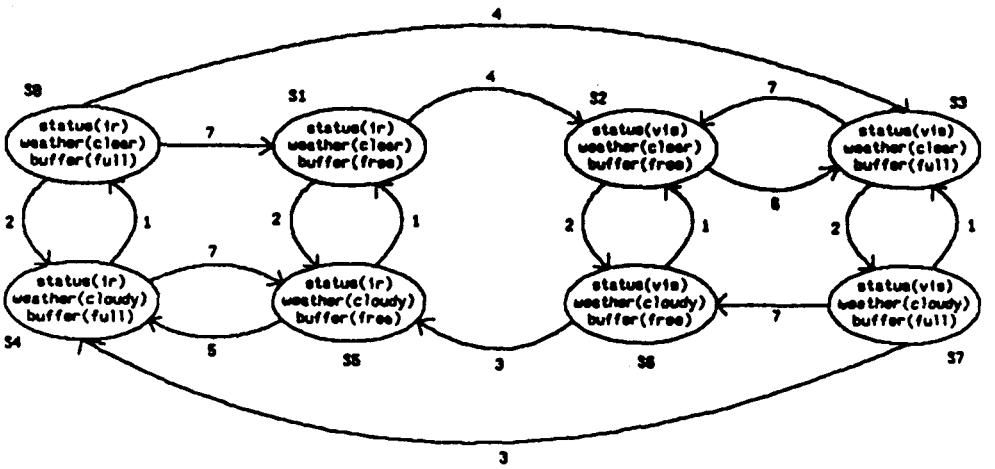

Figure 2: The projection of the picture-taking plan.

reasoning about it only when necessitated by plan execution failures. This paper goes some way towards the construction of such a planner by defining a flexible and expressive plan representation which has the ability to represent disjunctive plans. It does this without losing information such as Goal Structure, used by systems like NonLin [6], O-Plan [1] and SIPE [10].

It is important to realize that what we have defined is a representation able to describe contingent actions which is useful from an AI perspective. It is not hard to write contingent computer programs. But the eventual goal is to automate spacecraft command generation. It is likely that AI techniques will be used to perform this task. A start at this has been made with the Deviser planner for the Voyager spacecraft [12]. What this means is that AI representations must be used, and where inadequate, must be improved. Since disjunctive situations will often arise, any planner automatically generating spacecraft commands must be able to reason about disjunction.

We are now working on adapting O-Plan to generate C-plans. Simple disjunctive plans can already be generated; more interesting examples will require more complex generation algorithms. We are currently working on an algorithm to achieve a specified marking in a Petri Net to help produce a robust and efficient C-plan generation algorithm.

8. Acknowledgements.

This work was supported by the U.K. Alvey programme and the Science and Engineering Research Council on Grants GR/D/58987 (An Architecture for Knowledge Based Planning and Control) and GR/E/05421 (T-SAT: AI Applied to a Spacecraft). The support of System Designers plc for the work of the AIAI Knowledge-Based Planning Group is gratefully acknowledged.

9. References.

[1] Currie, K., & Tate, A. 1985. O-Plan: Control in the open planning architecture. In *The Proceedings of the BCS Expert Systems '85 Conference*. Warwick (December), Cambridge University Press.

[2] Stefik, M. 1981. Planning with constraints (Molgen: Part I), *Artificial Intelligence*. Vol. 16, pp. 111-140.

[3] Sacerdoti, E.D. 1975. The non-linear nature of plans. In *The Proceedings of IJCAI-75*. Tbilisi U.S.S.R.

[4] Chapman, D. 1985. Nonlinear planning: a rigorous reconstruction. In *The Proceedings of IJCAI-85*. pp. 1022-1024.

[5] Drummond, M.E. 1986. A representation of action and belief for automatic planning systems. In the proceedings of the *CSLI/AAAI workshop on Planning & Action*. Oregon, U.S.A. Morgan Kauffman. (Also Artificial Intelligence Applications Institute technical report AIAI-TR-16.)

[6] Tate, A. 1977. Generating Project Networks. In *The Proceedings of IJCAI-5*. Cambridge, Mass., U.S.A. pp. 888-893.

[7] Tate, A. 1984. Goal structure -- capturing the intent of plans. In *The Proceedings of ECAI-84*. Pisa, Italy. (September).

[8] Reisig, W. 1985. *Petri nets: an introduction*. Springer-Verlag, EATCS Monographs on theoretical computer science, vol. 4.

[9] Drummond, M.E. 1986. Plan Nets: a formal representation of action and belief for automatic planning systems. Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh.

[10] Wilkins, D.E. 1984. Domain independent planning: representation and plan generation. *Artificial Intelligence*, No. 22.

[11] M.E. Drummond. (August) 1985. Refining and extending the procedural net. In *The Proceedings of IJCAI-85*. Los Angeles, Calif. pp. 1010-1012.

[12] S.A. Vere. (May) 1983. Planning in Time: Windows and Durations for Activities and Goals. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. PAMI-5, No. 3, pp. 246-267.