983

N90-22301

Constraint-Based Evaluation of Sequential Procedures

Matthew R. Barry

Rockwell Space Operations Company
NASA/Johnson Space Center DF63
Houston, TX 77058
mbarry@nasamail.nasa.gov

19 January 1990

Abstract

Constraining the operation of an agent requires knowledge of the restrictions to physical and temporal capabilities of that agent, as well as an inherent understanding of the desires being processed by that agent. Usually a set of constraints are available that must be adhered to in order to foster safe operations. In the worst case, violation of a constraint may be cause to terminate operation. If the agent is carrying out a plan, then a method for predicting the agent's desires, and therefore possible constraint violations, is required. The conceptualization of *constraint-based reasoning* used herein assumes that a system knows how to select a constraint for application as well as how to apply that constraint once it is selected.

The purpose of this paper is to discuss the application of constraint-based reasoning for evaluating certain kinds of plans known as *sequential procedures*. By decomposing these plans, it is possible to apply context- dependent constraints in production system fashion without incorporating knowledge of the original planning process.

As an illustration of these ideas, this paper presents a system used in the Space Shuttle Mission Control Center to evaluate propulsive consumables management plans.

1 Introduction

We being with the assumption that a planning system has defined a plan to achieve some (desirable) goal. Normally, the planning agent passes the plan to an executor agent in order to achieve the goal. If the planner and the executor are independent, however, there may be situations in which misinterpretations or invalid instructions occur. These situations may occur if the processes assume different operation states, run asynchronously, or are modified frequently.

Rather than discovering a problem when an improper action is attempted, the output of the planning agent can be evaluated by an intermediate agent using a set of constraints governing the operation of the executor agent. If the evaluation process uncovers conflicts between the actions specified in a plan and the actions executable by the intended agent, then execution of that plan is inhibited until a mediator resolves the conflict. Though these constraints normally would be considered in an automated planner's line of reasoning, it might be the case that a plan is generated manually. In this case the evaluation process acts as an assistant to the human plan developer.

2 Plans

A plan specifies a means for accomplishing a goal[1]. The collection of actions defining the plan might be unique for each world in which the plan is applied. Furthermore, the results of applying the plan are dependent upon

[1]This paper does not investigate the various techniques for constructing plans (see instead [Steel 1987,Wilkins 1988,Stefik 1981] or the volume edited by Georgeff and Lansky [Georgeff and Lansky 1987]).

the initial state before any of the constituent actions are undertaken. We can sometimes conceptualize the overall plan as consisting of a set of shorter plans, with each element of the set possibly operating in a unique context.

We select a universe of discourse and a set of constraints that specify how the actions occurring in the changing world are to be applied in the current state. We define a *state* as a representation of the current situation in the world. An *action* changes the state of the world. An *action block* consists of a finite sequence of actions. A *conditional action* consists of a satisfaction condition and two different actions. If the condition evaluates *true* in the current state, then one of the two actions is selected. Otherwise, the second action is selected.

A *sequential procedure* maps positive integers into the action that is to be performed at the corresponding step of an infinite sequence [Genesereth 1987]. For example, a sequential procedure P enumerates the order of application for some actions $ActionI$ and $ActionJ$:

$$P(1) = ActionI(x)$$
$$P(2) = ActionJ(x)$$
$$P(3) = ActionJ(y)$$

where $ActionI(x)$ denotes the application of the object constant $ActionI$ to the object x, and so on. The state of the world at the end of the sequential procedure is the result of applying each action in turn beginning with some initial state for which the plan was generated.

2.1 Assumptions

In order to suitably restrict the kind of plans we can reason about, we make the following assumptions:

1. The agent assigned to carry out the actions will assume that the plan is executable and satisfies the goal.[2]

[2] In some cases there may be multiple agents available to operate in parallel, each on a different part of the plan [Lansky 1987]. Plans for these situations may require

2. The overall plan is decomposable into a finite sequence of smaller plans.

3. There are no conditional actions.

4. None of the action blocks overlap (the plan is linear).

3 Constraints

Since we assume that another agent created the plan, we must validate that agent's work. To do this we check that the plan satisfies the operational constraints of the executor agent. The constraints considered herein evaluate both the structure and content of a plan.

3.1 Identification

Three sorts of constraints are defined for suitably-restricted plans: *internal*, *local*, and *global*. *Internal* constraints apply to the semantic content of action or action block objects. These constraints validate the object itself, rather than its existential purpose.

Local constraints apply to the event currently under consideration as well as the events occurring just before it. These constraints are independent of the plan context. They represent physical system limitations, temporal requirements, and operational management techniques. Usually we can reason about *local* constraints as *action blocks*.

Global constraints apply to all of the actions in the plan, and are dependent upon the evaluation context.

---

an *interagent* constraint evaluation among differing contexts. Pednault [Pednault 1987] describes a technique for reducing some plans intended for multiple agents into a plan for a single agent. Such a plan may introduce a contextual evaluation based on *boundary conditions*.

3.2 Application

All of these constraints manifest themselves as *production rules* in the plan evaluation system. Each rule represents one constraint. Certain groupings of rules permit preprocessing and postprocessing activities, which might be context dependent. All of the contexts encountered during the evaluation are maintained in the *context memory*, which essentially is a database of running sequences, accomplished events, unaccomplished events, etc.

The implementation described below uses standard *production rules* to represent the constraints. This is convenient due to the nature of most constraints. Typically they read "Only do step B after step A is complete," or "Shutdown if value V of component X exceeds threshold Y." These statements might be captured with production rules like

```
if not Complete(Step(A))
then Pause(Step(b)).
```

or

```
if X.V > Y
then Shutdown.
```

4 Example

4.1 Background

One of the duties assigned to the Propulsion team in the Mission Control Center (MCC) is to maintain a propellant budgetting plan for all scheduled activities through the end of the mission. These plans allot propellant to future maneuvers and attitude maintenance activities. Furthermore, the team must ensure that certain minimum propellant quantities, or "redlines", are available at various points in the sequence. These plans are constructed initially before launch, but are updated frequently during the

course of the mission. The plan must always be an accurate representation of the activities to be carried out by the astronauts. Various constraints dictate proper implementation of maneuver sequences, redline construction, mandatory activities, etc. Violation of the redlines is cause to terminate a mission abruptly.

A propellant budget usually consists of a few hundred records itemizing each of the maneuvers and attitude maintenance periods. This implies the special case of *sequential procedures* within the previously defined plan restrictions. Each record in the plan represents an action to be performed, and the *immediate* context applies only to that record. Certain of the rules apply to the *immediate* context constraints. Some of these rules are especially important for verifying the validity of propellant usage references. That is, they verify that the propellant cost for a particular action is (1) non-zero, and (2) the proper budget item for the *global* context.

Most of the rules apply to *local* constraints. The *local* context consists of the current record and the few records before and after it in the plan, or an *action block*. *Local* constraints limit action durations (e.g. a maneuver should not last more than 20 minutes), adjacent actions (consecutive OMS burns are not realistic), and action modes (no primary FRCS thrusters firing during crew sleep periods).

The evaluation process levies constraints against the entire plan as well as to each action comprising the plan. Within the every plan there must appear certain actions, and these actions must appear in a certain order, regardless of how many actions separate them. The *global* context can be derived from the name of the plan file or from the first actions appearing in the plan (the evaluator assumes these represent the persistent context). The *global* context determines which data files are to be accessed, which constraint limits are to be applied, etc.

4.2 Implementation

The example system was coded in **awk** running on a UNIX workstation. The *pattern / action* constructs processed by **awk** represent the production

rules. Some machinery was built around these constructs to manage the context memory, to control iteration, and to manage data files. Though **awk** runs as an interpreter, the full application evaluates a plan consisting of several hundred actions in only a few seconds. This level of performance is quite acceptable considering the utility of the output and the potential time saved in manually debugging a plan.

The evaluator only displays problem conditions: it does not fix the problem itself.[3] A typical problem list may look like the following:

> Vernier timeline evaluation:
> (1) ERROR: Differing attitudes without maneuver (line 130).
> (2) ERROR: Invalid event time (line 151).
> (3) WARNING: RCS Hotfire occurs before FCS Checkout.
> Processed 260 lines.

Here the integer reference to the *sequential procedure* step number sometimes appears in the problem context description. The first **ERROR** message above might have been due to the sequence

$$P(129) = AttHold(180, 0, 270)$$
$$P(130) = AttHold(270, 0, 270)$$

whereas a correct implementation of the (virtual) action block might be

$$P(129) = AttHold(180, 0, 270)$$
$$P(130) = Maneuver(90, 0, 0)$$
$$P(131) = AttHold(270, 0, 270)$$

The example application uses the declarative programming paradigm to distinct advantage. The constraints involved in plan evaluation typically are ill-ordered, being applicable whenever the constrained situation arises, not as a sequential application of other constraints. By applying constraints

[3] Though it certainly could do so for errors occurring in an unambiguous context.

through a production system, the application is able to accommodate additional constraints without regard to the computational sequence. Procedural techniques which accomplish the same sort of reasoning are certainly possible, however the declarative techniques are easier to implement.[4] An equivalent system might also be coded in `CLIPS`, `LISP` or some other readily available substitute.

5 Conclusions

A simple technique for evaluating sequential procedures by applying operational constraints has been presented. This technique is useful for determining the feasibility of carrying out a plan that was created without rigorous knowledge of the constraints imposed by the executor agent.

The evaluation process incorporated into the Space Shuttle consumables planning programs strives to eliminate the mistakes commonly made when developing propellant budgets. This process provides real-time quality assurance for these critical products. It uncovers subtle problems that might go unnoticed until further downstream in the development effort by applying constraints to various aspects of the plan. It provides context sensitive reasoning capabilities that the (human) plan developers might overlook. Most importantly, it is flexible to enhancement, easily accommodating constraint modifications.

References

[Genesereth 1987] Genesereth and Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Georgeff and Lansky 1987] Georgeff and Lansky (eds.), *Reasoning About*

[4] Moreover, they better represent the actual evaluation process carried out by experts.

Actions and Plans: Proceedings of the 1986 Workshop, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Lansky 1987] Lansky, "A Representation of Parallel Activity Based on Events, Structure and Causality," in *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, Georgeff and Lansky (eds.), Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Pednault 1987] Pednault, "Formulating Multiagent, Dynamic-World Problems in the Classical Planning Framework," in *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, Georgeff and Lansky (eds.), Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Steel 1987] Steel, "Topics in Planning," in *Advanced Topics in Artificial Intelligence*, Nossum (ed.), Springer-Verlag, Berlin, 1987.

[Stefik 1981] Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence 16*, 1981.

[Wilkins 1988] Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.