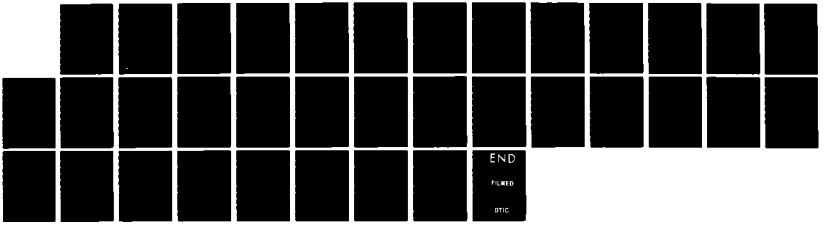
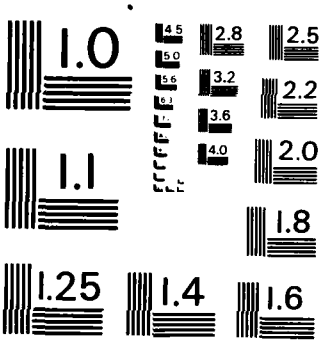
AD-A161 014 MODELING AND PLANNING ROBOTIC MANUFACTURING(U) SRI INTERNATIONAL MENLO PARK CA P C CHEESEMAN ET AL. MAR 85 N00014-83-C-0649 1/1

UNCLASSIFIED F/G 13/8 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS - 1963 - A

SRI International

(12)

MODELING AND PLANNING ROBOTIC MANUFACTURING

Final Report

March 1985

By: Peter C. Cheeseman, Senior Computer Scientist
William T. Park, Senior Research Engineer
Robotics Laboratory

Prepared for:

Office of Naval Research
Department of the Navy
800 N. Quincy Street
Arlington, Virginia 22217

Attention: Alan Meyrowitz

Contract N00014-83-C-0649 NR-049-573

SRI Project 6190

Approved:

David Nitzan, Director
Robotics Laboratory

W.F. Greenman, Vice President
Advanced Technology Division

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
415 326-6200 • TWX 910-373-2046 • Telex 334-486

CONTENTS

LIST OF ILLUSTRATIONS iv

LIST OF TABLES v

I INTRODUCTION 1

II PLANNER CAPABILITIES 2

A. Defined Conditions 2

B. Variable Value Selection 2

C. Truth Maintenance 3

D. Planner Extensions 3

III WORLD MODEL DESCRIPTIONS 6

A. Modeling Requirements 7

1. Kinds of Reasoning to be Supported 8

2. Kinds of Time to be Represented 8

B. Manufacturing Equipment 11

C. Workpieces 14

D. Processes 16

IV SUMMARY 22

V REFERENCES 23

APPENDIX

CONFERENCE PAPERS 25

ILLUSTRATIONS

1	Categorization of Manipulators	12
2	Inheritance of Characteristics	14
3	Subassembly Example	15
4	EXPLAN Operator Representing the "Grasp From Above" Process	18
5	Time Intervals in the "Grasp" Process Operator	20

I INTRODUCTION

The project has been primarily directed toward design and implemention of a basic planning system called EXPLAN. EXPLAN is a general-purpose system that uses domain-specific knowledge to guide its choices at every step. EXPLAN was debugged using an extended "blocks world" domain, which is a simplification of typical robot pick-and-place operations. EXPLAN is based on a new knowledge representation that associates a symbolic time interval and a truth value of "true" or "false" with every proposition about the "world." As it builds a plan, EXPLAN uses these truth values and time intervals to discover all harmful interactions that could occur between parallel actions and to order these actions to remove these potential conflicts.

EXPLAN uses this explicit representation of truth values and time intervals to describe preconditions for actions, as well as the actions themselves. This allows EXPLAN to reason efficiently about temporal relationships that occur during planning. This is a different approach from that used with state-change operator representations, which are the basis of many other artificial intelligence (AI) planning systems [1, 2]. These earlier planners must use "protection mechanisms" and fine-grained operators to represent information about actions and conditions taking place over time. These indirect mechanisms make the most frequently posed questions about temporal relationships difficult to answer efficiently. Two papers describing our extended time representation were presented in 1984: one at the IEEE Robotics Conference in Atlanta, Georgia [3], and the other at the ASME conference in Las Vegas [4].

II PLANNER CAPABILITIES

The development of EXPLAN and its application to the robotic domain has required extensions to the state of the AI-based automatic planning art, as described below.

A. Defined Conditions

EXPLAN extended the complexity of the domains to which it can be applied by developing a method for allowing complex conditions to be associated with possible actions. For example, a condition for using a robot arm to pick up an object is that the object to be picked up be less than a given maximum weight. Because the weight of an object depends also on the weight of any other objects attached to it, the evaluation of the total weight entails a calculation. If the current weight is less than the given threshold, the condition is satisfied. However, as the plan continues to grow (more actions are added to it), further objects may be attached to the object to be picked up and these additional objects may push the total weight over the threshold. The problem is to detect efficiently when previously true conditions are affected (or potentially affected) by plan changes.

B. Variable Value Selection

Previous planners (e.g., SIPE [2]) have used variables during plan construction to represent quantities that are not explicitly specified at that stage. These variables have associated constraints that any possible values must satisfy, and these constraints accumulate as the planning process proceeds. However, an analysis of this use of variables shows that they are essentially existential variables, as used in a first-order predicate calculus. The use of EXPLAN in the simple domains in which it was tested showed that full first-order predicate calculus is necessary to represent common conditions associated with operators. This use of a full representation introduces both existential variables and universally quantified variables, as well as dependencies of these on each other.

The use of full logic to represent conditions introduces new problems in evaluating conditions and maintaining their truth during plan creation. In particular, the introduction of both existential and universally quantified variables requires proving that there is at least one value that satisfies the constraints (for existential variables) or that the constraints are satisfied by all values (for universally quantified variables). A further difficulty (which has required

considerable program modification to accommodate) is that when a proof of a condition involving a universally quantified variable fails, it is necessary to find all instances of failure and set them up as new goals. For example, the goal that a part be clear (i.e., nothing on it) may fail because there are particular objects on it. A specific goal to remove each such object must be created and added to the goal list.

The method for variable value selection that has been implemented defers the choice of value either until information becomes available for selecting a good value or until an arbitrary value selection is forced. This use of the "least-commitment" approach avoids premature selection of values and so avoids wasting time in backing up in appropriate arbitrary choices. Value selection is forced when subsequent decisions depend on which value is chosen. Rather than trying to maintain separate worlds for each possible choice, EXPLAN makes a selection and allows backup to the choice point if necessary.

C. Truth Maintenance

When a condition is evaluated and found to be satisfied, the planner must ensure that further elaboration of the plan does not invalidate such conditions. This is simple if the condition is a simple atomic form that can be checked by pattern matching. However, if the condition has a complex definition, maintaining the truth of the condition can be computationally expensive. A method that has been investigated to reduce this cost is to store the premises of the condition evaluation ("proof") in the data base and to check if any addition to the data base (i.e., to the world model) interacts with these premises. When such interactions are detected, the system must check the corresponding condition to determine whether it is still satisfied. A simple version of this truth maintenance mechanism has been implemented and it has worked satisfactorily on the examples investigated; however, further effort is necessary to make this more general.

D. Planner Extensions

EXPLAN could be extended in many ways to improve its ability to deal with real-world problems, such as occur in a manufacturing environment. These include:

- Preference Goals. This involves prioritizing the goals so that some are marked as essential while others are given a preference weighting by the user. Such goal preferences allow the planner to satisfy a weighted subset of the goals if a solution that satisfies all of them cannot be found.

- Temporal Ordering Constraints. These include such information as the fact that particular processes associated with a given work piece must be performed before others (in general, a partial ordering), and basic temporal resource constraints, such as that only one object may be processed at a time on a particular machine.

- Durational Constraints. This includes such information as the average time necessary to perform particular operations and constraints on the allowable duration of particular processes (e.g., the time necessary for an object to cool before the next process). In addition to the average time of particular intervals, a measure of the variance of these intervals will be included, so that the planner can reason about variations in the expected execution time for a plan (or partial plan--e.g., critical path analysis).

- Set Representation In addition to representing individual objects, it is necessary to represent sets of equivalent objects so that the planner can reason about batches. This representation will allow individuals within a batch to be identified if necessary (e.g., if execution failure leaves some objects incompletely processed).

- Sensory Planning. This requires the planner to create subplans for using sensor(s) to provide information that is missing at plan time, but that will be available at execution time. The use of sensor plans will be integrated with the action plan development by the planner.

- Conditional Planning. This is the creation of alternative branches in the plan dependent on the results of sensor test(s). Such branches will be created when the corresponding information is missing at plan time and the required actions depend on the correct value.

- Execution Monitoring. This is the insertion of sensor steps in the plan (at plan time) to verify that the world is in the state expected by the planner. If any of these tests fail at run time, then the system is in a particular error state.

- Replanning. If the execution system detects an error, then the planner will be used to replan in the new situation. This replanning will be designed to use as much of the old plan as is still valid.

- Durational Logic. Currently EXPLAN is only concerned with relative time orderings of actions to ensure consistency of parallel actions. We propose to extend the planner to reason about the duration of actions, so that EXPLAN can reason about critical paths, durational constraints, and so on.

- Coordination with Externally Scheduled Events. We propose to extend EXPLAN to reason about events that are not under the control of the system but that can still be incorporated into the plan. For example, the scheduled delivery of components by an external agent will be included in the plan by fixing the time moments in the plan to particular world times.

- Iterated Plans. We propose to extend the planner to produce plans with loops so that repeated operations (e.g., identical items in a batch) can be included. This will require extending the temporal reasoning representation and will require a more complex interaction detection and removal procedure. Simulation of Plan Execution. This requires developing a separate program that simulates an execution controller and the environment (including failures) to test plans generated by the planner.

III WORLD MODEL DESCRIPTIONS

As noted in the Introduction, this project emphasized the development of the planner program, rather than the world model. Early in the project, we developed a domain model for a trivial but illustrative box-stacking task to exercise the planner during its development. We also defined some more advanced modeling concepts to represent manufacturing equipment, workpieces, and manufacturing processes. In the time available, we could not develop these models sufficiently to construct a complete test case for a second experiment. However, defining them served to uncover the types of information that the planner would need from the world models in a realistic application.

Much of the information the planner needs is deducible from a relatively small set of fundamental facts, such as properties of individual objects, their taxonomic classification, and their aggregation into compound objects such as assemblies. We chose the semantic network formalism of Deliyanni and Kowalski [5] to represent such information in PROLOG.

PROLOG offered certain advantages over the other major artificial *intelligence language*, LISP, *for knowledge representation* and retrieval. PROLOG has built-in facilities for automatic deduction, unification, and backtracking. Although the C-PROLOG implementation we used [6] was more than adequate for the scale of these exploratory investigations, we would question its suitability for a realistic industrial application.

In a full-scale implementation, it may prove useful to implement the *modeling system and the planner in quite different ways.* For example, they might be two completely separate, but communicating programs (e.g., a general-purpose planner and an "intelligent" data base), implemented in different languages. In this project, we used the same PROLOG interpreter for both purposes.

In Section III-A, we describe the questions that we think the planner will need to ask of the world model. In Section III-B, we give some examples of equipment models, and in Section III-C, some examples of workpiece models. In Section III-D, we discuss ways of representing processes by a combination of semantic nets and the action descriptions used directly by the planner in backward chaining.

A. Modeling Requirements

In planning a manufacturing task, the planner will need answers to questions such as the following:

EQUIPMENT

- *Which manipulators can lift a 4-pound workpiece?*
- *Which manipulators can reach a particular part feeder?*
- *What camera's field of view covers the location of the workpiece?*
- *What sensors can inspect a workpiece for a crack?*
- *What machine tools can drill a half-inch hole?*
- *Can a particular gripper operate a particular hand tool?*

WORKPIECES

- *What processes are to be performed on a workpiece?*
- *In what order (if any)?*
- *Where is the workpiece?*
- *How much does it weigh?*
- *Is it ferromagnetic?*
- *What kind of tooling reference points does the workpiece have?*

PROCESSES

- *Where is a process performed?*
- *What equipment can perform the process?*
- *On what kind of workpieces can the process be performed?*
- *How long does the process take, on the average?*
- *Are any preparatory (or subsequent) processes necessary?*

1. *Kinds of Reasoning to be Supported*

The above questions can be categorized according to the type of reasoning necessary to answer them, as follows:

None
E.g., "How much does the workpiece weigh?" Ideally, these questions require no special kind of reasoning—only a single access to the appropriate item of information in the data base.

Taxonomic Classification
E.g., "Which machine tools can drill holes," or "Can a magnet hold the housing screws?" These questions may require specification (enumerating the items within a certain class, such as hole-drilling tools), or generalization (determining whether a given item or class of items is a member of a more abstract class, such as objects made of ferromagnetic materials). This would require following chains of relationships that describe membership of objects in classes, and of classes in other classes.

Aggregation Analysis
E.g., "Which parts support a given part in an assembly," or, "What components make up a given subassembly?" These questions require following chains of relationships such as adjacency or attachment between pairs of workpieces. Often, a list of objects found to be so interrelated must then be analyzed in complex ways to answer the query. For example, to determine support relationships would involve determining which parts were immediately under other parts.

2. *Kinds of Time to be Represented*

Another important characteristic of information in the world model is its variation with time. In fact, *two* different kinds of time variation are involved: one is variation with "planning," or "wall-clock" time (the time during which the planner is generating the plan). The other is variation with "plan" time (the time from the start of the plan). Furthermore, planning and plan times are not necessarily correlated: The planner could equally well construct a plan in reverse order –from end to beginning – or even piecemeal in no particular order.

Information in the world model changes during *planning* time whenever the planner makes a decision. For example, the world model might initially give a single location for a bolt – e.g., a bolt feeder. After the planner inserts an action in the plan to move the bolt from the feeder to a hole in a workpiece, the world model should give two different locations for the bolt: one corresponding to times in the plan before the move, and the other for times after. The information in the

model about the instantaneous state of the world therefore varies with *plan* time: The location that the world model gives for the bolt depends on the time in the plan.

The world model should record, during generation, when each fact was entered into the world model during the planning process. This allows the planner to retract all consequences of an incorrect decision when it has to backtrack. It should also record when, in plan time, each fact holds. This allows it to give the planner correct information about what the state of the world will be at any instant during execution of the plan.

One way to record when a fact was entered during planning generation is as follows: (1) Represent all consequences of a planning decision by only *adding* information to the world model, never erasing any old information. (2) Number the planner's decisions consecutively, in the order they are made in plan*ning* time. 3 Label each added fact with the number of the latest planning decision. Whenever backtracking undoes a decision, simply erase from the world model all the facts labeled with its number or a higher one.

One way to record when a fact holds during plan execution is to label it with the beginning and ending instants of the interval over which it holds, and to assert some additional facts about the temporal order of those instants with respect to other instants in the plan. Before adding a new fact to the world model, it is necessary to check that it does not conflict with other facts. If it does, this indicates that the planner should backtrack to the last decision it made and try to redo it. Thus, successive facts added to the world model may only tighten constraints on when time instants in the plan occur. A fact that holds at any time during planning continues to hold until backtracking retracts it, but the interval over which it holds may become more tightly constrained.

To implement this recommendation, we intend to represent all relationships as facts in the PROLOG data base of the general form

relationship(<*relationship-name*>**,**
<ordered list of entities so related>],
<*the time when the relationship starts*>,
<*the time when the relationship ends*>,
<*number of latest planning decision*>**)**.

The starting and ending times for the relationship are *plan* times, while the number of the latest planning decision represents the *planning* time. Thus, the fact is labeled with both kinds of time in which it occurs, as recommended.

In this notation, the initial state of the world (e.g., *"The location of the bolt is the bolt feeder."*) would be represented by the fact

```
relationship(location,[bolt,boltfeeder],t0,t1,0).
```

The meaning we attach to this **location** relationship is that the **bolt** is at the location of the **boltfeeder**. Furthermore, this relationship holds from time **t0** (the beginning of the plan) to some later time **t1**. The last item (**0**) in the body of this fact is the number of decisions that the planner has made so far (none, since planning has not yet started). Now, assume that the seventeenth decision the planner makes is to move the bolt to the hole in the workpiece. This would result only in the *addition* of the following new facts to the PROLOG data base:

```
relationship(location,[bolt,bolthole],t2,t3,17).
relationship(before,17,[],t1,t2,17).
```

Both facts are labeled with the number of the planning decision (shown here as "17" for purposes of illustration) that added them to the data base. The first of these two facts describes the new location of the bolt, the **bolthole**. By implication, **t2 < t3**. The second fact represents an explicit temporal relationship—that instant **t1** in the plan occurs **before t2**. Since a temporal relation only involves time instants, the list "**[]**" of related objects is empty. The effect of the temporal relationship is to further *constrain* the plan (by the additional condition that the the time interval **t0**—**t1** must end *before* **t2**. Note that it has not been necessary to erase the original fact

```
relationship(location,[bolt,boltfeeder],t0,t1,0).
```

from the data base, so no information has been lost as a consequence of this planning decision.

Now, assume that the planner later decides to take back its decision to move the bolt. It executes the following PROLOG statements:

```
retractall(relationship(_,_,_,_,17)).
fail.
```

The **retractall** function erases from the data base all the facts with "17" as the last item in their body—i.e., it retracts all consequences of decision number 17, which is the one being undone. The **fail** function then causes the PROLOG system to backtrack and attempt to re-satisfy (in a different, and hopefully more successful way) the goals that resulted in the incorrect decision.

B. Manufacturing Equipment

The representation we have developed for manufacturing equipment is primarily taxonomic. For example, a given factory might contain the equipment listed in Table 1.

Table 1: Hypothetical Equipment Set

Robot manipulators
- arm-1: Strong, fast, inaccurate, short
- arm-2: Weak, slow, accurate, short
- arm-3: Strong, slow, inaccurate, long

End effectors
- Dexterous, three-fingered gripper
- Simple, parallel-jaw clamp
- Torque wrench

Hand tools
- Power drill
- Power screwdriver
- Cutting torch

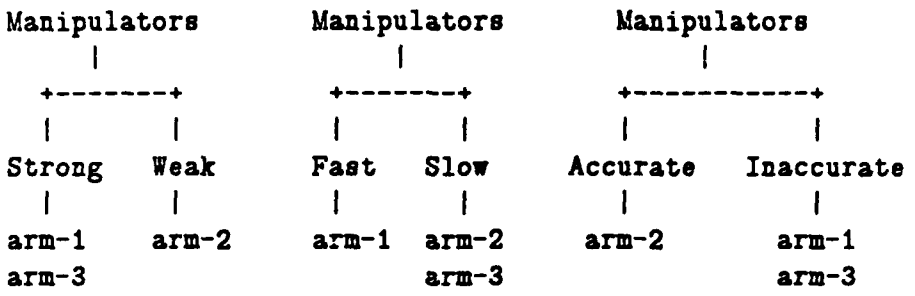
Sensors
- Fixed television camera
- "Free" television camera
- Tactile fingertip on dexterous gripper

Machine tools
- Drill press
- 5-axis Mill

Conveyors
- From robot to mill
- From robot to drill press
- From stock room, past robot, to shipping

Containers
- Tote boxes (movable)
- Tool rack (immovable)

Three possible taxonomies, for the manipulators in Table 3 alone, are the following:



Each of these classification schemes assigns each of the manipulators to one of two disjunctive, or mutually-exclusive, categories. Each divides the set of three arms in a different way. The instances of a class need not be mutually exclusive, however—for example, we could use all six categories from the above example at once, as in Figure 1 below.

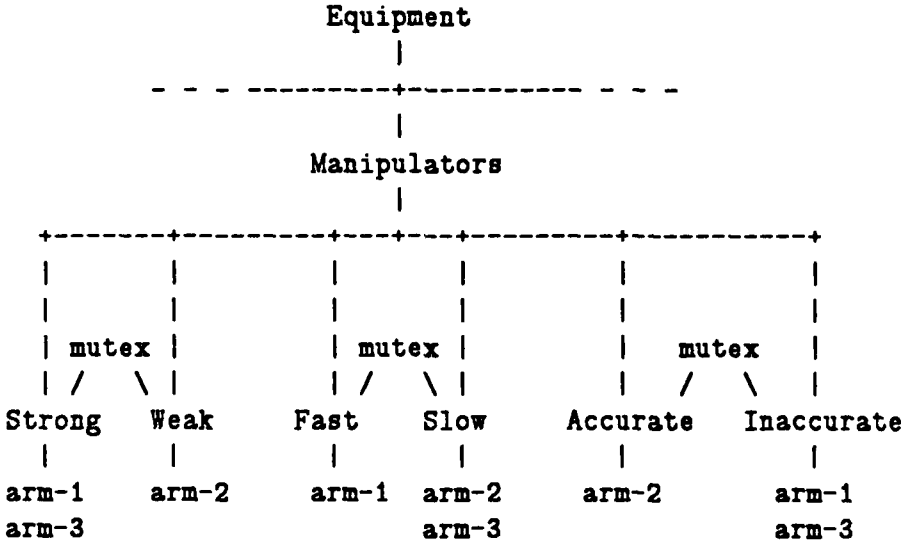


Figure 1: Categorization of Manipulators

The "mutex" relationships in this diagram indicate which sets of categories are mutually exclusive—e.g., no strong arm is also a weak arm). We would represent this categorization as follows in the PROLOG world model:

```
relationship(isa,[arm-1,strongarm],t0,tfinal,0)
relationship(isa,[arm-1,fastarm],t0,tfinal,0)
relationship(isa,[arm-1,inaccuratea_m],t0,tfinal,0)
```

... and so on for arm-2 and arm-3 ...

```
relationship(isa,[strongarm,manipulator],t0,tfinal,0)
relationship(isa,[weakarm,manipulator],t0,tfinal,0)
relationship(mutex,[strongarm,weakarm],t0,tfinal,0)


relationship(isa,[fastarm,manipulator],t0,tfinal,0)
relationship(isa,[slowarm,manipulator],t0,tfinal,0)
relationship(mutex,[fastarm,slowarm],t0,tfinal,0)


relationship(isa,[inaccuratearm,manipulator],t0,tfinal,0)
relationship(isa,[accuratearm,manipulator],t0,tfinal,0)
relationship(mutex,[inaccuratearm,accuratearm],t0,tfinal,0)


relationship(isa,[manipulator,equipment],t0,tfinal,0)
```

The *isa* ("is a") relationship means that the first item in the bracketed list is an instance of the class of objects denoted by the second item in the list. The mutex relationship says that the classes named in the list (e.g., strong arm and weak arm) are mutually exclusive categories. Each relationship holds over the entire plan interval (**t0** through **tfinal**).

The utility of the semantic net representation is that we only need to store the information about common characteristics of objects in a given class once, instead of once for each object. For example, consider the fragment of a net in Figure 2: where "-->" indicates information about an object or class of objects. In PROLOG syntax, the information in this diagram would appear as follows:

```
relationship(isa,[arm-1,strongarm],t0,tfinal,0)
relationship(can-reach,[arm-1,drillpress],t0,tfinal,0)


relationship(isa,[arm-3,strongarm],t0,tfinal,0)
relationship(can-reach,[arm-1,mill],t0,tfinal,0)


relationship(isa,[strongarm,manipulator],t0,tfinal,0)
relationship(can-lift,[strongarm,'10 pounds'],t0,tfinal,0)
```

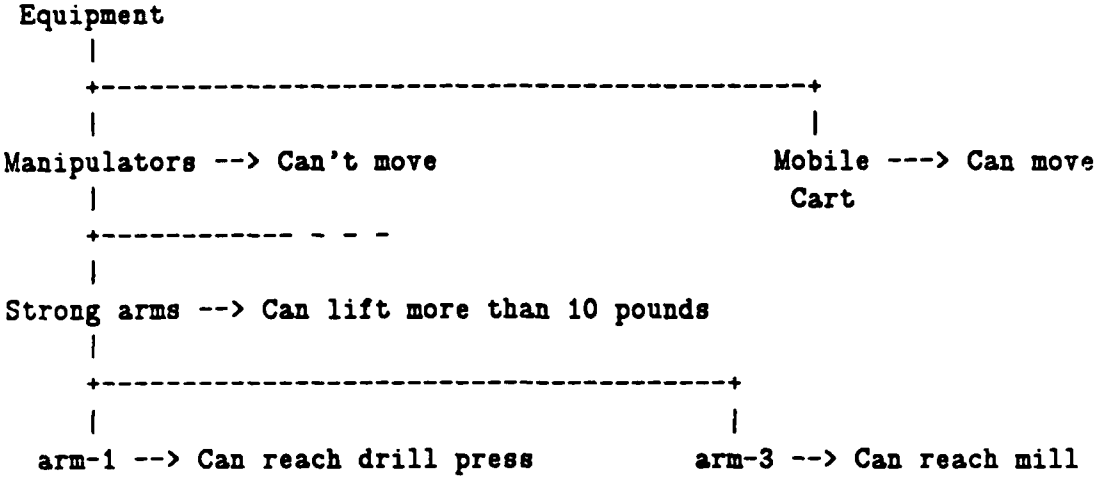


Figure 2: Inheritance of Characteristics

```
relationship(isa,[manipulator,equipment],t0,tfinal,0)
relationship(can-move,[manipulator,false],t0,tfinal,0)

relationship(isa,[mobile-cart,equipment],t0,tfinal,0)
relationship(can-move,[mobile-cart,true],t0,tfinal,0)
```

To find out if an object has a particular characteristic, the world model system simply checks for that characteristic in each of the classes to which the object belongs. For example, the planner might need to know how much weight **arm-1** can carry. This information would be found in the net by tracing up the chain of "isa" relationships from **arm-1** to the "**manipulator**" node. Another use of hierarchical inheritance of information is to answer a question such as, "What non-mobile equipment can lift more than ten pounds?" Starting at the equipment node and tracing downwards along the "isa" relationships discovers all items of equipment with that property (arm-1 and arm-3).

C. Workpieces

Two important kinds of workpiece information for which semantic nets are a convenient representation are workpiece "features" and assembly descriptions. Some examples of features are holes, polished surfaces, and threads. Assembly descriptions are tree diagrams that indicate how components are to be combined into subassemblies, and how the subassemblies are to be combined into the final

product. Several levels of subassembly integration may be involved in manufacturing a complex product like a computer (e.g., discrete components into boards, boards into racks, and racks into cabinets). We might represent this example graphically as in Figure 3 below.

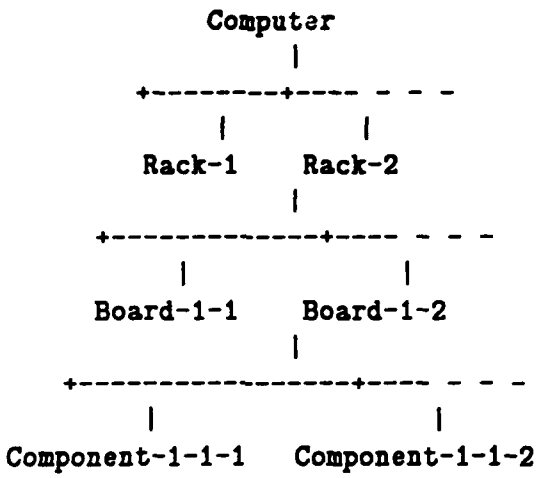


Figure 3: Subassembly Example

In PROLOG syntax the information in this diagram would appear as follows, using a relationship called "has-part":

```
relationship(has-part,[computer,rack-1],t0,tfinal,0)
relationship(has-part,[computer,rack-2],t0,tfinal,0)
    .
    .
    .
relationship(has-part,[rack-1,board-1-1],t0,tfinal,0)
relationship(has-part,[rack-1,board-1-2],t0,tfinal,0)
    .
    .
    .
relationship(has-part,[board-1-1,component-1-1-1],t0,tfinal,0)
relationship(has-part,[board-1-1,component-1-1-2],t0,tfinal,0)
    .
    .
    .
```

This sort of representation is particularly convenient when production engineering analysis has specified an assembly plan. Other representations for assemblies are

more appropriate for less hierarchically-organized products than computers. For example, a small gasoline engine contains a number of obvious subassemblies—carburetor, fuel pump, magneto, starter cord reel, etc. For such devices, *Buckingham chaining* can provide a feasible assembly plan. This consists of planning the disassembly of the complete product, and then running the plan in reverse. At each step of the reverse planning process, the planner would ask the world model for a list of immediately-removable parts. Selecting one part from the list to be the next part removed is a backtrackable decision. Forced backtracking, a built-in PROLOG feature, may be used to generate all feasible disassembly plans (and therefore all feasible assembly plans). These may then be compared on the basis of some performance measure, such as minimal number of reorientations required, or degree of parallelism in the plan (high parallelism provides an opportunity for reduced cycle time through overlapped assembly operations). Deducing which parts are removable from a given subassembly requires geometric reasoning, but at least one automatic planning program (NASA-Goddard's Assembly Sequence Planner) is already able to do this for blocks and cylinders with cylindrical holes [7]. Coincidentally, it is also written in PROLOG.

D. Processes

Processes are represented as EXPLAN operators, so that EXPLAN can reason about their interactions. This permits us to describe many important characteristics of processes, such as the following:

- The primary effect of the process
- Side effects of the process
- Prerequisites (or "preconditions") for performing the process (i.e., conditions that must pertain before the action can be performed)
- "Postrequisites" for performing the process (i.e., the action can only be performed if these conditions will pertain after the action is completed.).
- Events that occur during the process
- The order (total or partial) in which those events occur
- Duration of the process and of events within the process.

Figure 3 shows an example of an EXPLAN operator that represents the process of grasping an object from above. The equipment that does the grasping, as well as

the object that is grasped, are variables, or parameters, of this operators. So, this operator could be used in planning the assembly of a tape recorder to represent a robot hand descending onto a hex nut and then closing its fingers. It could equally well be used in planning the loading of a ship to represent an overhead crane lowering an electromagnet onto a two-ton roll of steel, and then turning on the magnet.

A "**%**" sets off a comment in an operator. The operator representation proper consists of four lists that, respectively, declare the time instants in the process, indicate their temporal ordering, describe the preconditions for performing the process, and describe the effects of performing the process.

The *time instant list* merely declares which PROLOG variables in the rest of the operator represent time instants. In this example, there are eight distinct instants, **T1** through **T8**. The order of the variables in this list does not imply any time ordering of the corresponding time instants.

The *temporal ordering list* declares all the important ordering relationships between the time instants in the process, in terms of pairwise orderings. For example, an entry such as **bfr(T3,T1)** indicates that the time instant represented by the variable **T1** occurs before the instant represented by **T2**. Usually, the set of pairwise orderings in this list will describe a single graph of partially-ordered instants, as in the present example. However, if it should be appropriate for a particular process, the relative order of some time instants with respect to others may be left unspecified. In that case, the ordering relationships would describe two or more *disjoint* graphs of partially-ordered time instants.

The *precondition list* is a list of "fact" predicates. They indicate conditions that must be true or false before, during, and/or after the process, in order to be able to perform the process. This set of conditions should be thought of as *necessary* but not *sufficient* for the process to be performed successfully: In general, there will usually be additional unstated conditions that are too difficult to describe, impossible to determine, or simply unknown. This merely reflects the fact that it is impossible to capture every aspect of the real world in a computer model.

The general form of a process precondition description is

fact(<predicate>, <tr | f>, <initial time>, <final time>)

which means that the fact represented by the predicate is either true (**''tr''**) or false (**''f''**), over the indicated time interval. For example,

fact(holding(U1,Equipment),f,T4,T3)

```
op( [ T1,T2,T3,T4,T5,T6,T7,T8 ],     %   TIME INSTANTS IN PROCESS

    [ bfr(T3,T1), bfr(T4,T3),        %   ORDERING OF TIME INSTANTS
      bfr(T5,T3), bfr(T1,T6),        % "bfr(Tx,Ty)" = "Tx before Ty"
      bfr(T7,T3), bfr(T1,T2),
      bfr(T2,T8)              ],

    %   TIME-INVARIANT PRECONDITIONS
    [ % The equipment that performs the process must be able
      % to lift the workpiece.
        can-lift(Equipment,Workpiece),

      % There should be an object, "Support," that can
      % support the workpiece.
        can-support(Support,Workpiece),
    ]
     % TIME-DEPENDENT PRECONDITIONS (See Figure 4)
    [ % The equipment is not holding anything, initially.
        fact(holding(U1,Equipment),f,T4,T3),

      % The workpiece is on its support, initially.
        fact(on(Workpiece,Support),tr,T5,T6),

      % Nothing else may rest on top of the workpiece before,
      % during, or after the grasping operation.
        fact(on(U2,Workpiece),f,T7,T8)
    ]
    % EFFECTS OF THE PROCESS (See Figure 4)
    [ % The equipment starts to close on the workpiece at T3
      % and finishes closing at T1, at which time the grasping
      % action has been accomplished.
        action(grasp(Workpiece,Equipment),T3,T1),

      % The equipment begins holding the workpiece at T1 (as soon
      % as it finished closing on it), and continues to hold it
      % until T2.
        fact(holding(Workpiece,Equipment),tr,T1,T2)
    ]).
```

Figure 4: EXPLAN Operator Representing the "Grasp From Above" Process

says that the equipment is *not* ("**f**") holding anything (represented by universally quantified variable "**U1**") during the time interval from instant **T4** to instant **T3**.

The *effects list* describes the changes that should be introduced into the world model if the planner decides to include this process in the overall plan. The first item in the list is of the form

action(<predicate>, <initial time>, <final time>)

which means that the action represented by the predicate should be inserted into the plan. It also indicates that the physical events that constitute the action should be considered to take place at execution time during the indicated time interval. In the workpiece-grasping example,

action(grasp(Workpiece,Equipment),T3,T1)

says that in the action which this operator represents, the equipment grasps the workpiece during the time interval from **T3** to **T1**.

Any items in the effects list after the first, or "action," entry, represent new facts that must be added to the PROLOG data base if the planner should decide to add the process to the plan. They have the general form

fact(<predicate>, <tr | f>, <initial time>, <final time>).

which means that the predicate is true ("**tr**") or *false* ("**f**") *over the time* interval. For example,

fact(holding(Workpiece,Equipment),tr,T1,T2)

indicates that if the "grasp" process is performed, the result will be that the equipment will hold the workpiece from time **T1** onward. This is, of course, the primary effect of the process, and the usual reason why the process would be included in a plan. More strictly, this operator says that the equipment will hold the workpiece only until the time instant **T2**. **T2** serves here as a dummy time instant that represents the time when the equipment stops holding the workpiece if it ever does. For example, the planner would usually insert an "ungrasp" process (not shown), into the plan at some point after the grasp process. Since the main effect of the ungrasp process is to make the equipment release whatever it is holding at the time, that would determine when **T2** occurs. Alternatively, the final version of the plan could leave the equipment holding the workpiece, if that was acceptable in the particular manufacturing task being planned. In that case, time **T2** would simply become the final time in the overall plan.

Figure 4 shows the temporal relationships, events, subprocesses, and state changes in the grasping process, as described by the operator in Figure 3. The

```
                    TIME-ORDERING RELATIONS

T7 -> T3
T5 -> T3
T4 -> T3
      T3 -> T1
            T1 -> T6
            T1 -> T2
                  T2 -> T8


                    PROCESS DESCRIPTION

T5 ************** T6        The workpiece is supported by something
T7 ************** T8        No other objects are on the workpiece
T4 ** T3                     The equipment is not holding anything
      T3 ** T1               The equipment grasps the workpiece
            T1 ** T2         The equipment is holding the workpiece
```

Figure 5: Time Intervals in the "Grasp" Process Operator

upper part of the figure shows the partial ordering of the eight important time instants in the process. For example, the first line

```
T7 ---> T3
```

means that time instant **T7** precedes time instant **T3**. The next line shows that **T4** and **T5** also precede **T3**. However, there is no information in the operator description (or the figure) about the relative order of **T7**, **T5**, and **T4**. (Although they appear at the same horizontal position in the diagram, that does not mean they occur at the same time.) Similarly, it does not say whether **T6** or **T8** occurs first, or if they are simultaneous.

The lower half of Figure 4 illustrates the conditions that hold at various times during the grasp process, and the actions that occur. This portion of the diagram is derived partly from the precondition list and partly from the effects list. For example, the line

```
T7 ********************* T8  No other objects are on the workpiece
```

tells us that, between time instants **T7** and **T8**, there ought to be no other objects on the workpiece to be grasped. This is a precondition for performing the action,

since, if there were another object on top of the part, the hand of the robot arm might not be able to reach it or grasp it securely from above. The time interval for this precondition, from **T7** through **T8**, encompasses essentially the entire grasping process, as is clear from the temporal-ordering chart in the upper half of Figure 4. This precondition also happens to involve a universal quantifier ("no other objects"), as discussed on page 2.

The second line of the process description,

```
T5 ********************* T6  The workpiece is supported by something
```

says that the part should be supported by something for the entire duration of the grasping process (since **T5** and **T6** encompass the interval **T3**—**T1** in which the grasping action takes place). This is because, in the type of grasp being described here, the equipment only closes on the workpiece: It does not remove the workpiece from its support. (Another operator, such as "pickup" would probably appear later in the plan to accomplish that). Furthermore, the workpiece will not be grasped firmly until the equipment finishes closing on it, and the operator represents the time during which the closing action takes place as the time interval from **T3** until **T1**. The support for the workpiece can be removed at any time after **T1** with no ill effects, and the operator represents the time at which that might occur as **T6**.

The next three lines of the process description

```
T4 **** T3                    The equipment is not holding anything
        T3 **** T1            The equipment grasps the workpiece
                T1 **** T2    The equipment is holding the workpiece
```

represent the grasping process itself, which takes place between time instants **T3** and **T1**. Before **T3**, the equipment is holding nothing, and after **T1** it is holding the workpiece. According to this description of the process, what the equipment is holding is undefined between **T3** and **T1**. This is often a useful way to describe a transitional process. In this case, it allows us to represent the fact that the action of closing on the workpiece is a kind of autonomous process that should not be interrupted. We might wish to represent other processes as taking place instantaneously—e.g., turning on a floodlight for a camera. For these processes, we would simply omit the middle interval from **T3** to **T1**, and equate **T3** to **T1** to close the gap.

IV SUMMARY

The AI-based automatic planner (called EXPLAN), developed under ONR Contract N00014-83-C-0649, extended the state of the automatic-planning art in a number of ways. The most important of these includes the use of a temporal interval representation to facilitate reasoning about parallel activities and the introduction of defined conditions in operators to allow more realistic operator descriptions. These extensions necessitated the ability to perform truth maintenance automatically on defined conditions to ensure that they remained true as the plan evolved and the use of variables whose value assignment was deferred as long as possible. The temporal reasoning component was presented at an IEEE robotics conference [3] and an ASME conference [4] (see appendix).

V REFERENCES

1. S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Trans. Pattern Analysis and Machine Intelligence* (May 1983).

2. D. E. Wilkins, "Recovering from Execution Errors in SIPE," (A.I. Center Technical Note 346), Menlo Park, California, SRI International, January 1985.

3. P. C. Cheeseman, "A Representation of Time for Automatic Planning," *Proc. IEEE Conf. on Robotics*, MIT Press, Cambridge, Massachusetts, Atlanta, Georgia (March 1984).

4. P. C. Cheeseman, "A Representation of Time for Automatic Planning in Complex Domains," *Computers in Engineering*, pp. 352, Amer. Soc. of Mechanical Engineers, New York, New York (August 1984).

5. A. Deliyanni and R. A. Kowalski, "Logic and Semantic Networks," *Communications of the ACM*, March 1979, *22* (3), pp. 184-192. .

6. W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* New York, New York, Springer-Verlag, 1981.

7. T. Premack et al., "Design and Implementation of a Compliant Robot with Force Feedback and Strategy Planning Software," (Tech. Memo. 86111), College Park, Maryland, NASA Goddard Space Flight Center, May 1984.

Appendix

CONFERENCE PAPERS

Proc. of 1984 Int. Computers In Engineering Conference and Exhibit, Las Vegas, Nevada, August 12-15, 1984

A REPRESENTATION OF TIME FOR AUTOMATIC PLANNING IN COMPLEX DOMAINS

P. Cheeseman
SRI International
Menlo Park, California

In previous planners, time is represented implicitly in the sequence of states and the operators (actions) that make up a plan. This method of representing time makes it impossible to describe what is happening during an operator application, and makes it difficult to reason about possible interactions between parallel actions. A new time representation is described here that avoids these problems by representing time explicitly so that all the time relationships needed in planning can be deduced efficiently. This representation associates an interval with every proposition about the world so that the current set of such propositions forms a model of the world (over the time period of the plan). During planning the planner tries to insert the effects of proposed actions into the current plan and to evaluate if the conditions of the action are currently satisfied. Both these deductions require finding all the propositions in the current plan that could interact with the additions, then using a fast interval logic to decide if there is an interaction and what to do in that case.

When planning is extended to deal with complex domains, it becomes unreasonable to expect the operators to assert all the effects (direct and indirect) that result from the application of the new operator in the current plan context. For example, a condition associated with a robot PICKUP operator is that the object to be picked up should be less than a given maximum weight. If this condition is found to be satisfied in the current context, it is unreasonable to expect a subsequently applied operator (such as a PUTDOWN operator) to detect that this maximum weight condition is possibly violated. This is an example of the well known problem in AI on "Truth Maintenance," where it is necessary to detect when a new piece of information invalidates previous inferences, thus making all conclusions based of these false premises suspect. The problem is to find efficient methods for detecting such interactions and methods for correcting all the affected consequences.

The solution adopted here is for every domain to divide the possible predicates into primitive or other predicates defined in terms of the primitives. The operators only assert primitive effects, leaving the planner to find all the relevant consequences of such assertions. For example, if the PUTDOWN operator puts a new object on another, it is up to the planner to check if this effect invalidates any current conditions (such as the maximum weight condition of a previous PICKUP operator). Similarly, when the conditions of an operator are evaluated in a particular plan context, the planner must check that these conditions are true given current world model.

The time representation discussed here is similar to that discussed by Allen,* but here the intervals are represented by their begin and end moments, making the interval relationship deductions needed in planning simpler. Also, this representation only deduces the temporal relationship between intervals associated with interacting predicates rather than all currently known intervals. Further, it has been found to be more efficient to fully order any interacting intervals in the current plan even when the current constraints allow more than one possibility, rather than keeping constraints on the possible orderings. The advantage of forced orderings is to reduce the complexity of deciding specific temporal relationships (simple chaining is sufficient) while allowing for the possibility of alternative orderings through backup. In practice, there is usually good heuristic reasons for preferring one ordering over another when an ordering has to be enforced.

In addition to the above interval logic, each proposition can have a duration associated with it. This duration information can be in the form of a maximum and minimum duration. Such duration information allows the deduction of critical paths in the current plan and plan optimization. Constraints on the duration of actions can interact with the deduction of possible interval orderings because a particular ordering may be allowed by the current interval constraints but violate duration constraints.

*"Planning Using a Temporal World Model," James Allen and Johannes Koomen, Proc. Eighth Conference on Artificial Intelligence, Karlsruhe, W. Germany (August 1983).

Proc. of IEEE Int. Conf. on Robotics, Atlanta, Georgia March 1984.

A Representation of Time for Automatic Planning

Peter Cheeseman

Robotics Department,
SRI International,
333 Ravenswood Rd., Menlo Park, California

Abstract

This paper describes a representation of time and its associated inference rules that is being applied to process planning. The representation associates a time interval and a duration with every proposition in a world model, so that particular time relationships can be reasoned about explicitly. These time intervals are defined by their beginning and end instants, so interval relationships, such as "before" and "overlaps", are defined indirectly by the corresponding relations between the end instants. This new representation allows simpler rules for deducing the usual planning interval relationships. Similarly, interval durations are associated with each proposition by asserting the maximum and minimum durations, thus allowing the discovery of critical paths and allowing partial plans to be optimized. A planner has been designed to use the above time representation to produce plans for work stations with multiple robots, for example. A method is given for the detection and correction of interactions in the current partial plan that avoids any conflicts and deadlocks that could arise.

1 Introduction

This paper describes the representation of time and the rules for its manipulation as used in an automatic planner. Such a planner is being developed and is being applied to the problem of process planning for robotic equipment. Since the main bottleneck in setting up a new robotic process is the time required to program a working process plan containing all the necessary details, the development of an automatic planning system would be an economic gain.

Artificial intelligence (AI) has discovered a number of techniques, such as hierarchical and parallel planning, that bring the goal of automatic planning systems nearer. However, when these techniques are applied to real world problems, such as robotic process planning, some major shortcomings appear. In particular, the problems of representing and reasoning with uncertainty and planning in incompletely specified environments need to be solved. Also, the fundamental problem of representing and reasoning about processes (possibly in parallel) taking place over time requires an efficient solution. Fortunately, some AI workers have considered the general problem of the representation of time ([8],[11]) and have developed temporal logics that claim to solve the problem. Interval based logics have been proposed ([2],[3]) as suitable for planning, but for efficiency temporal reasoning must be integrated with the planning procedure–not as a separate temporal reasoning module, as proposed by Allen.

This paper splits temporal reasoning into reasoning about relative time orderings and reasoning about durations of intervals (and sequences of intervals). The rules axioms and definitions (i.e. logic) required to answer the temporal questions generated during planning are described in the following sections.

The work reported herein was supported by the Office of Naval Research under Contract No. N00014-83-C-0649 and by the National Science Foundation under Grant ECS-8200615.

2 Previous Methods of Time Representation in Planning

A representative early planner is the STRIPS system [5], in which time is represented implicitly. In this system, the planner keeps a "snapshot" of the state of the world after the application of an operator (an event that changes the world). A sequence of snapshots provides an implicit representation of a particular time sequence investigated by the planner. For efficiency, these snapshots record only the changes caused by the preceding operator and inherit everything else that was true before it was applied. For example, in a world containing just blocks (A,B,..), after the application of a PUT-ON(A, B) operator, the world is in a new state in which everything is the same except that ON(A, B) is true, and CLEAR(B) is false. These changes are dictated by the operator's add/delete list. In the previous notation, ON(A, B), for example, means that A is on B, where A and B are particular constants (note that all capitalized arguments in the following are constants).

The inherent contradiction in the fact that the proposition ON(A, B) is true and false at different times can be removed explicitly by using a "situational calculus" McCarthy and Hayes [7]. In this representation, every proposition has an extra constant added to it to indicate the situation (state, snapshot) in which it is being asserted. For example, there is no contradiction in asserting both NOT(ON(A, B, s21)) and ON(A, B, s32), as the proposi-

tions are being asserted in different states. These situational markers can be explicitly time-ordered [e.g. (s32 $\succ$ s21)], thus ordering the corresponding states. If these time orderings are combined with the persistence assumption (i.e., that the truth value of a proposition remains unaltered unless explicitly negated at some latter time), then the truth value of a proposition at any time can be deduced. The inheritance implied by the persistence assumption is an efficient way of representing the world at any time provided that the operators change only a small portion of the world description in a single application. Other planners based on the persistence assumption have also been written (Vere [10], Sacerdoti [9]).

The situational calculus approach allows any ordering information to be asserted between states, including partial time ordering as well as a strict sequential ordering. This freedom is utilized in parallel (non linear) planners in which a partial time order is imposed only to remove any inconsistencies in the world at any time. This method has the advantage of producing plans that are as parallel as possible (allowing parallel execution), and does not make unnecessary premature commitments to a particular orderings (which may have to be undone as the plan unfolds), resulting in faster plans that are produced more quickly.

This avoidance of premature orderings of operators (or corresponding states) can be taken a step further. For example, if it is discovered that situation s1 is incompatible with s2, then if either (s1 $\succ$ s2) or (s2 $\succ$ s1) the incompatibility will be resolved. However, there may be insufficient information at this stage to choose between the possible orderings. Rather than make an arbitrary choice, Eder[4] and Allen[2] suggested imposing the constraint ORDERED(s1 s2), which means that s1 comes before s2 or vice versa. Unfortunately, the presence of such disjunctional constraints usually creates an extremely large number of possible worlds—one for each combination of orderings. The approach described here is to arbitrarily chose a particular ordering when there isn't a good reason for preferring one over the other, and to allow the system to reverse this choice on backtrack, if necessary. This method can potentially explore all possible orderings, but only keeps one set of orderings active at a time.

3 Planning with an Interval Based Representation

The major difference between the temporal logic of this paper and STRIPS [5] is that it uses a set of world descriptions (worlds instead of states) where a world is described over all time, and the operators map between these worlds. A world description is a set of predicates asserted to hold or not hold over particular intervals, with an optional duration. Since a world describes the state of all relevant things over all time, it can easily describe the state of the world at any particular time when needed. A world description is a generalization of STRIPS states, and can be thought of as the set of all possible states over the period covered by the plan, allowing a global view of the current plan. An operator in the new representation generates a new world description which is the same as the previous, except where modified by the operator. Because a world description is over all time, it is possible to represent what is happening during an operator application—not just at the beginning and end of an operator application, as in the case where actions are represented as state change operators (e.g., DEVISER, Vere [10]).

This representation follows the example of other authors (e.g., Allen [1]) in making intervals the principal time relation. However, in this logic, intervals are represented implicitly by their start and finish times rather than a single symbol (e.g., Allen [1]), allowing more convenient deduction of interval relationships. More formally, a world description consists of a set of propositions such as:

A1: Holds(On(x, A), T1, T2)
i.e., $\exists x\ \forall t\ [T1 \preceq t \preceq T2 \rightarrow On(x, A)]$
or "There exists an x such that for all times t between T1 and T2, x is on A".

A2: NotHolds(On(x, A), T1, T2)
i.e., $\forall(x,t)\ [T1 \preceq t \preceq T2 \rightarrow \neg On(x, A)]$

or "For all times t between T1 and T2 there is no x on A".

A3: Holds($\neg$On(x, A), T1, T2)
i.e., $\exists x\ \forall t\ [T1 \preceq t \preceq T2 \rightarrow \neg On(x, A)]$

A4: NotHolds($\neg$On(x, A), T1, T2)
i.e., $\forall(x,t)\ [T1 \preceq t \preceq T2 \rightarrow On(x, A)]$

The intended interpretation of such "Holds" and "NotHolds" assertions is to relate the first argument (a predicate) to an interval defined by its begin and end times (the last two arguments). Note that if the predicate argument only contains constants (no variables), then "Holds" assertions are all that is needed. For example, in the above assertions, if the variable x is replaced by a constant (e.g., x = B), then A1 and A4 assert that B is on A over the interval and both A2 and A3 assert that B is not on A over the same interval. Thus, if the goal language is restricted to ground predicates only, the interval logic can be reduced to the logic of the "Holds" predicate. We could use an equivalent (object language) representation in which the interval end-markers appear explicitly in every world proposition, such as:

On(A, B, T1, T2), and At(Robot, X, Y, T3, T4).

The meta-language Holds/NotHolds representation is preferred here because it provides a cleaner separation between the interval and its associated predicate.

Operators map a world description into a new one. These operators consist of a set of conditions that must be true in the old world to make the operator applicable, and a set of changes (additions and deletions) that create a new world from the old. An evolving plan (a partial time ordering of operator applications) is stored by the planner and is complete when the corresponding final world description satisfies the given goal criteria. The main task of the planner is to ensure that the effects and conditions of every proposed action (operator) that is to be inserted into the current plan is consistent with the corresponding current world description. This consistency check is explained in detail below, and essentially is just the requirement that no proposition (or logically equivalent proposition) is both true and false at the same time. In addition to consistency checking, the time representation conveniently allows the evaluation of the conditions of a proposed operator to see if they are already achieved as a result of an existing operator or as part of the initial world description.

A plan is built by repeating the following cycle. The simplified version given here, however, ignores such problems as the bookkeeping necessary for backup, what to do if a goal evaluates to "unknown", methods of assessing progress on different OR branches, etc.

Step 1—Evaluate all new goals in the context of the current world description, adding any that are not already achieved to the list of unsolved goals. A goal is achieved if either it can be unified with part of the initial world description (note not the initial state as in STRIPS), or if it can be made a consequence of an existing action (operator). If the evaluation of a goal requires the imposition of an arbitrary time ordering or variable binding, then these decisions are stored as possible backup points.

Step 2—Select a goal from the list of unsolved goals as the next target goal. If this list is empty, the problem has been solved. The planner can use heuristic selection rules to guide its next goal choice.

Step 3—Select an operator to achieve this goal (with other possible side effects). Alternative choices are stored as possible backup points. If there are no remaining operators for this goal, then if it is one of the given goals, the problem is unsolvable (since further backup is impossible), otherwise, undo the operator that generated the current goal and go to Step 2. Operator selection can also be guided by heuristic selection rules.

Step 4—Insert the effects of the chosen operator into the previous world description and check to see if this generates any inconsistencies. Attempt to remove any such inconsistencies by imposing a minimal time ordering (and possibly changing previous arbitrarily imposed orderings). If any inconsistencies cannot be removed, undo this operator application and go to Step 2.

Step 5—Add any conditions of the operator that are not achieved in the current context to the new goal list, and also remove any previous unachieved goals that are now achieved as an result of the new operator from the goal list. Go to Step 1.

In Step 1 and Step 5 of the above procedure, the planner must find the relationship in time between a particular Holds/NotHolds assertion and all other relevant assertions that constitute the current world description. If there is no definite relationship between two intervals (i.e., the end moments of both assertions are not all in a definite before/after order), then the planner should impose an ordering on the end moments to create a definite relationship, as described in the next section.

"Holds" and "NotHolds" assertions relevant to determining the status of a goal are those whose predicate argument matches (unifies) with the goal, where qualifiers of the predicate term (such as "not") are ignored in matching (this is achieved in practice by making the truth value of the predicate an extra argument of "Holds/NotHolds"). For example, the goal Holds(On(B, A), T3, T4) will unify with assertions A1 to A4 above, including the special cases in a particular world description where x in A1 and A3 is replaced by a particular constant. Note that the interval arguments are ignored in the matching process, since the aim is to find all intervals that might be relevant to the current goal. Similarly, the goal NotHolds(On(y, A), T5, T6)—(i.e., Clear(A)) will unify with A1 to A4. The last example illustrates a problem that arises with logical equivalence. If the goal Holds(Clear(A), T1, T2) is defined to be equivalent to NotHolds(On(x, A), T1, T2) in a particular domain, the planner will not detect this equivalence during matching because the forms are different. The solution adopted by the author is for each domain to distinguish between primitive and defined predicates, and to always translate defined predicates into their equivalent primitive form. For example, "Clear" goals are always translated into equivalent "On" goals in the blocks world.

In the above examples, some of the predicates of the matching relevant intervals are logically compatible with the goal while others are inconsistent. Inconsistencies can be removed by ensuring the corresponding intervals do not overlap, and compatible intervals can be merged, as described below. A further complication arises in dealing with compound goals. Usually, a conjunction of goals (i.e., goal1 and goal2 and...etc.) in the first argument position of a Holds or NotHolds assertion can be broken into a set of Holds/NotHolds assertions, each with a different predicate goal as its first argument. However, if the conjunction of goals shares some of the same variables, then it is more convenient to keep such interdependent goals together as a compound goal so that the full effects of a variable binding can be easily checked. For example, the goal NotHolds([On(x, A) and Piston(x)], T1, T2) should not be decomposed into separate goals.

4 The Logic of Relative Interval Relationships

The previous section shows that, during planning, it is frequently necessary to find the temporal relationship between two intervals whose associated predicates may be compatible (unifiable) or inconsistent. These relationship investigations arise firstly when a condition of an operator is evaluated within the current world description and the planner must discover if this condition is consistent with the current world and if it is, whether it is already achieved in the current world. Also, the planner must check to see if the effects of a proposed operator are consistent with the current world, and if they are whether they change the status of any currently unachieved goals. The rest of this section presents the possible interval relationships and the corresponding planner responses.

In the following, interval I1 (a relevant interval in the current world description) begins at moment T1 and ends at moment T2, while interval I2 (an interval associated with the current goal) begins at T3 and ends at T4. These time moments can be ordered relative to each other by asserting such relations as "before(T1, T2)", "after(T4, T3)" and "before-or-equal(T2, T3)", and so indirectly order the corresponding intervals. Since time is transitive (i.e., there are rules such as "if T1 is before T2 and T2 is before T3 then T1 is before T3"), a set of atomic ordering relations between the moments allows the deduction of the relationship between any pair of moments (i.e., before, after, before-or-equal, after-or-equal or undefined). In trying to establish the relationship between intervals I1 and I2, we already know that before-or-equal(T1, T2) and before-or-equal(T3, T4) is true. It then remains to establish the relationship between (T1,T3), (T2,T3), (T1,T4) and (T2,T4). Each of these pairs can potentially have one of five possible relations (above), giving 5^4=625 potential interval relationships. Fortunately, many of these possibilities are inconsistent (e.g., before(T4,T1), before(T2,T4)) or redundant (e.g., before(T3,T1) implies before(T3,T2)) or unnecessary (as the planner can reach a decision on what action to take when the interval relationship is only partially known).

The following set of rules is used by a planner to decide what action to take.

(1) If there is no overlap between I1 and I2, then there is no interaction between these intervals and the planner does not consider this possibility further. There is no overlap if either before(T4,T1) or before(T2,T3) is already asserted.

(2) If before(T1,T3) then
  If before-or-equal(T3,T2) then
    (i.e., T3 in interval I1)
    If predicates inconsistent then
      abort attempt to insert the goal (I2).
  Otherwise achieve the goal by making I2 a sub-interval of I1 (i.e., assert before-or-equal(T4,T2)).
  If predicates inconsistent then try make I1 before I2 (i.e., assert before(T2,T3)), thus removing the overlap between them.
  Otherwise, can achieve the goal by making I2 a sub-interval of I1.

(3) If before-or-equal(T3,T1) then
  If before-or-equal(T1,T4) then
    (i.e., T1 in interval I2)
    If predicates inconsistent then
      abort attempt to insert the goal, I2.
    Otherwise, I1 is irrelevant to determining the status of the goal, because I2 starts before I1.
  If the predicates are inconsistent then try to make I2 before I1 (i.e., assert before(T4,T1)), leaving the goal unachieved.
  Otherwise, I1 is irrelevant to determining the status of the goal, because I2 starts before I1.

(4) If before(T4,T2) then
  If before-or-equal(T1,T4) then
    (i.e., T4 is in interval I1)
    If predicates inconsistent then
      abort attempt to insert the goal, I2.
    Otherwise, make the goal achieved by asserting before-or-equal(T1,T3).
  If predicates inconsistent, then make I1 irrelevant to I2 by asserting before-or-equal(T4,T1). (i.e., separate the intervals).

(5) If before-or-equal(T2,T4) then
  If before-or-equal(T3,T2) then
    (i.e., T2 is in interval I2)
    If predicates inconsistent then
      abort attempt to insert the goal, I2.
    Otherwise, I1 is irrelevant to the goal.
  If predicates inconsistent then make I1 irrelevant to I2 by asserting before-or-equal(T2,T3). (i.e., separate the intervals)

(6) If there is no relationship between I1 and I2 and the predicates are compatible, then achieve the goal by make I2 a sub-interval of I1 by asserting before-or-equal(T1,T3) and before-or-equal(T4,T2).

(7) If none of the above conditions are met, then get the next relevant interval and repeat.

An example illustration of the reasoning behind rule (2) is given below.

```
Case1 before(T1,T3), before-or-equal(T3,T2), predi-
cates inconsistent.

  I1 (other interval) T1.................T2
  I2 (Goal interval)     T3.......??????????T4

Case2 before(T1,T3), before-or-equal(T3,T2), predi-
cates compatible.

  I1 (other interval) T1.....................T2
  I2 (goal interval)     T3.........T4

Case3 before(T1,T3), predicates compatible.

  I1 (other interval) T1.....................T2
  I2 (goal interval)     T3.........T4

Case4 before(T1,T3), predicates inconsistent.

  I1 (other interval) T1.......T2
  I2 (goal interval)          T3........T4
```

The above rules are an English statement of the rules used in a planning system written in PROLOG [12]. This planner is general purpose, but its intended first application is planning assemblies for a robot work-station. Since such work stations can include multiple manipulators, there is a strong need to develop a temporal logic that allows reasoning about parallel activities, as given in the above set of rules.

5 The Logic of Durations

When building a plan, it is important not only to arrive at a partial ordering of the activities that constitute the plan, but it is necessary to know how long each activity takes in order to find the minimum time (or minimum cost) plan. If there is no information on the duration of some of the proposed actions, either because it is unknown or because it is inherently too variable, then the planner is unable to estimate the execution time of the resulting plan. If such plans include parallel branches, then it is necessary for the planner to include coordination commands for the run-time execution system. For example, if the plan requires that two different manipulators do an insertion in the same small area and the times for these insertions are unknown, then the plan must include instructions to the run-time controller to prevent such actions taking place simultaneously.

The coordination of asynchronous parallel processes is a well known problem on operating system design, where the major difficulty is to ensure that deadlocks do not occur. A deadlock is where process 1 is waiting for process 2 to finish and visa versa, so neither get started. The planning procedure and its associated interval logic given above ensure that resource conflicts and deadlocks do not occur. This comes about because the conditions associated with each operator specify what resources (e.g., space) are required for that operator (cation) to successfully apply. The interval reasoning component then ensures that possible competing conditions are detected, and appropriate time orderings are imposed to remove possible conflicts. An alternative procedure for coordinating sub-plans for parallel activities without deadlocks is described by Geogeff [6].

When the duration of component activities is known, or at least upper and lower bounds on the duration are known, then it is possible to deduce important information about a plan execution, such as critical paths and maximum/minimum time of completion. The analysis of critical paths and other plan properties is a well studied branch of operations research, and it is intended to adapt such techniques to the planner being built by the author. The information about durations can be represented in the same way as interval information, by appending the duration to each predicate along with the interval moments. If it is assumed that all duration information can be expressed as upper and lower bounds (in some standardized units such as seconds) then the form of a component of the world description could be:

Holds(Predicate, T1, T2, Min, Max)—and similarly for "NotHolds"

This representation allows every predicate in the world description to be associated with an interval and a duration.

6 Summary

The problem of automatically planning an industrial task, such as assembly, is a complex one where most of the complexity is in representing and reasoning about geometry, uncertainty and temporal relationships over the objects in the domain. This paper addresses the problem of temporal representation and reasoning. It outlines a procedure for planning and the temporal decision logic to go with it. A representation of interval information is given that associates an interval (defined by its beginning and end moments) with each predicate that constitutes a world description. The use of world descriptions that give the evolution of the world over all time, rather than

a sequence of state descriptions (as in STRIPS [5]) allows greater complexity of time relationships to be represented and reasoned about. This explicit representation allows a comparatively simple logic for deciding the interaction, if any, between any pair of potentially interacting conditions and operator effects. The given procedure for evaluating proposed insertions of conditions and action effects ensures that there are no conflicts or deadlocks. In addition, this representation can be easily extended to include information on the duration of an interval, allowing the deduction of critical path and total execution time estimates.

References

[1] Allen, J. F., "An Interval-Based Representation of Temporal Knowledge", Proc. International Joint Conf. on Art. Intell., Vancouver pp221-226 1981.

[2] Allen, J. F. and Koomen J. A., "Planning Using a Temporal World Model", Proc. Eight International Joint Conf. on Art. Intelligence, Aug. 1983, Karlsruhe, pp 741-747.

[3] Cheeseman, P. C., "A Representation of Time for Planning", Tech. Note 278, Artificial Intelligence Center, SRI International, Feb. 1983.

[4] Eder, G., "A System for Cautious Planning", Research Report No. 27, Dept. Art. Intell., Univ. of Edinburgh. 1976.

[5] Fikes, R. E., Hart, P. E., and Nilsson, N. J., "Learning and Executing Generalized Robot Plans", Artificial Intelligence, 3, 4, pp.251-288 Winter 1972

[6] Georgeff, M. P., "Communication and Interaction in Multi-agent Planning", Proc. Am. Assoc. for Art. Intelligence, Washington, Aug. 1983. pp 125-129.

[7] McCarthy, J. and Hayes, P., "Some Philosophical Problems from the Standpoint of Artificial Intelligence", Machine Intelligence, Vol. 4, B. Meltzer and D. Michie, eds., American Elsevier, N.Y. pp. 463-502, 1969.

[8] McDermott, D., "A Temporal Logic for Reasoning About Processes and Plans", Research Report 196, Dept. of Computer Sci., Yale Univ. 1981

[9] Sacerdoti, E. D., "A Structure for Plans and Behavior", Elsevier Computer Science Library, 1977.

[10] Vere, S. A., "Planning in Time: Windows and Durations for Activities and Goals", Research Report Jet Propulsion Lab. Nov. 1981.

[11] Vilain, M. B., "A System for Reasoning About Time", proc. American Assoc. for Artificial Intelligence, Pittsburgh, Aug. 1982.

[12] Warren, D. H. D., Pereira, L. M. and Pereira, F. "Prolog—The Language and its Implementation Compared with Lisp." In SIGPLAN/SIGART Newsletter. ACM Symposium on A.I. and Programming Languages, August 1977.

END

FILMED

12-85

DTIC