Plan Recognition and Generalization in Command Languages with Application to Telerobotics

Wael I. Yared and Thomas B. Sheridan, *Fellow, IEEE*

***Abstract*—A method for pragmatic inference as a necessary accompaniment to command languages is proposed. The approach taken focuses on the modeling and recognition of the human operator's intent, which relates sequences of domain actions ("plans") to changes in some model of the task environment. The salient feature of this module is that it captures some of the physical and linguistic contextual aspects of an instruction. This provides a basis for generalization and reinterpretation of the instruction in different task environments. The theoretical development is founded on previous work in computational linguistics and some recent models in the theory of action and intention. To illustrate these ideas, an experimental command language to a telerobot is implemented; the program consists of three different components: a robot graphic simulation, the command language itself, and the domain-independent pragmatic inference module. Examples of task instruction processes are provided to demonstrate the benefits of this approach.**

I. Modeling Pragmatics in Command Languages

A. The Problem

A KEY ISSUE in command language design is the modeling and communication of the human user's intent. Intent relates sequences of domain actions ("plans") to changes in some model of the task environment. Its representation clearly constitutes, therefore, an issue in pragmatics. (For an introductory text on pragmatics, or the contextual use of language, see Levinson [15]. Some of the original papers in philosophy of language that have pioneered the field of linguistic pragmatics can be found in Martinich [17]—in particular, articles by GRICE, Austin, and Searle.) Command languages for interactive systems such as telerobots have been traditionally restricted to the syntactic and semantic aspects of the interaction, thus giving rise to two main limitations. First, commands are supported as single, isolated utterances rather than as interrelated steps in a cohesive discourse. Second, little or no contextual information is captured from the human operator's instruction, such as the relevance of a particular step in the instruction to the overall task. Without a common underlying representation of action interrelationships and intentionality, no amount of computation can perform the reasoning expected in a truly cooperative process, such as generalizing a plan of action or exploiting contingencies in the task environment. A human–computer interaction that lacks these notions is artificial—not in the sense of requiring the human to learn a formal syntax, but in the more aggravating one of lacking the purposive dimension of any human language.

To illustrate, consider the following scenario, chosen from the machine learning literature (Andreae [3]). A robot is to pick a block from the first row of a warehouse and return with it to its starting position, somewhere outside the warehouse (Fig. 1). The operator types (in some robot command language, e.g., VAL or RAPT (Lozano-Perez [16]) a sequence of commands that achieves this particular operation, which the robot then executes.

Suppose now the first row becomes depleted of blocks. A relevant question would be whether the robot can benefit from any part of the previous instruction to face this new situation, where it has to try the second row. There is more to this problem than a simple substitution of parameters: the general block-retrieval procedure may turn out to have a complex structure (with branching forks, iterative loops, etc.) not present in the initial formulation of the task by the operator. Recovery of the general procedure from a linear sequence of commands provided by the human operator partly involves identifying the operator's intent behind each element of the sequence.

The motivation for this research stems from the difficulty of interfacing with semiautonomous telerobots operating in unstructured environments such as outer space. These systems are typically called on to perform nonrepetitive tasks, and methods must be developed to achieve on-line task instruction and plan generalization.

B. The Approach

We approach the procedure-acquisition problem from a plan recognition perspective. A plan recognition system reasons about a set of described or observed actions, which it attempts to explain by constructing a plan that contains them (Kautz and Allen [13]). Classical plan recognition essentially revolves around a search and match method applied to a library of possible plans. We are departing from the classical plan recognition formalism in two ways. First, we are extending it from the domain of question-answering, in which it had been developed (Allen and Perrault [1]) and is traditionally applied (e.g., Perrault and Allen [19], Allen [2], Sidner [23]) to that of acquiring procedures from traces. Second, we are applying it to recognize intentions and generalize from plans. In other words, the human operator provides the entire plan as a sequence of actions applicable in one particular instance, and it is the recognition system's burden to infer the operator's intentions. It is important to point out what this entails: first, since the entire plan is provided by the operator, the usual search control problems are avoided and the closed-world assumption necessary to current plan recognition systems can be lifted. This enables the system to deal with novel situations and plans, a vital capability for systems that learn. Intention recognition, as used here, is not a search and match process but one of nondeductive inference. Second, a generalization technique is added to the recognition algorithms. This generalization (or "plan reinterpretation") algorithm makes use of the information inferred in the recognition stages, and evaluates the relevance of the intended acts of the plan in the new context. The intention recognition and plan reinterpretation algorithms are

Manuscript received March 6, 1990; revised August 5, 1990. This work was supported in part by the National Aeronautics and Space Administration, and in part by the Jet Propulsion Laboratory, California Institute of Technology.

The authors are with the Man-Machine Systems Laboratory, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number 9040003.

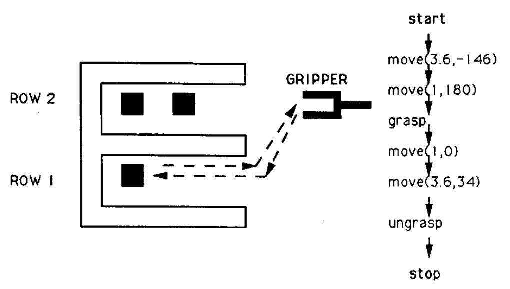


Fig. 1. Sample task instruction.

based on a representation of action interrelationships and a model of intentional action that are explained in the next section. The central feature of this representation is the emphasis it places on the relevance of an action in a plan, or the role each act plays in the plan. We draw a parallel here between utterances in a natural language discourse and physical actions as steps in a plan: just as the context of a discourse provides constraints on the interpretation of each of its utterances, so does the overall plan form the role of each of its actions. We focus on just two features of context here, the physical and the "linguistic." The physical context of an action in a plan is the particular configuration of the task environment at the onset of that action, including the location and physical features of each object, the agent, and the complete state of the agent. The "linguistic" context of an action in an instruction is meant here to include the other steps of the instruction (the remaining actions).[1]

To illustrate these ideas, we present a simple command language for an instructible two degree of freedom cartesian manipulator as a case study.

II. Action Interrelationships in Plans

A. Background

The AI literature defines a plan to be a temporal order on a finite number of steps (Chapman [7]), which almost invariably represent actions.[2] Following the early STRIPS model (Fikes and Nilsson [9]), actions are usually represented by operators; each operator instance is characterized by finite sets of prerequisites, positive effects and negative effects (Genesereth and Nilsson [10]). States of the world are represented by situations, or sets of propositions. A planning problem then consists of an initial situation, a goal situation, and a database of operators that describes the available actions. An adequate plan must demonstrably achieve the goal situation when executed in the initial state.

Standard plan recognition systems explain a set of described or observed actions by constructing a plan that contains them (Kautz and Allen [13]). Typically, they need a plan library to do so. The plan library is assumed to be correct and complete: thus, not only must the operator library (the repertoire of primitive actions) contain representations of all valid actions in the domain, it must also contain representations of all propositions and actions that can be related to any encoded action. In other words, all valid action combinations that occur in a particular domain must be encoded. A plan recognizer is then a search and matching process (Goodman and Litman [12]) that takes the input—e.g., a described action or set of actions—and finds all the plans in the plan library that are consistent with the input. See Fig. 2.

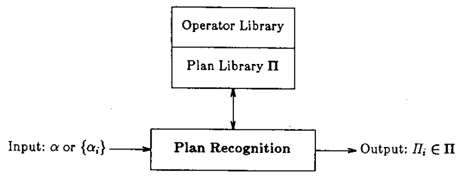


Fig. 2. Standard plan recognition.

As stated in Pollack [20], three different relations (and special cases thereof) can exist between two actions α and β of a plan:

1) α causes β: there is an operator with header α in the operator library that has β on its effect list.
2) α is-a-precondition-of β: there is an operator with header β in the operator library that has α on its precondition list.
3) α is-a-way-to β: there is an operator with header β in the operator library that has α (as part of) its body.

The first two relate actions that are at the same hierarchical level in a plan—in other words, siblings. The third relation differs from the previous two in that it depicts α as part of a step decomposition of β and not as a sibling of β. Consider the following typical operators:

HEADER: pickup(x)
PRECONDITIONS: ontable(x) $\wedge$ handempty $\wedge$ clear(x)
EFFECTS: $\neg$ontable(x) $\wedge$ $\neg$handempty $\wedge$ holding(x)
HEADER: ungrasp(x)
PRECONDITIONS: holding(x)
EFFECTS: ontable(x) $\wedge$ handempty.

The symbols $\wedge$ and $\neg$ denote the logical connectives AND and NOT. In a pickup(block) $\rightarrow$ ungrasp(block) sequence the link between the two operators is an example of the precondition relationship because pickup (block) asserts (among other things) holding(block), the precondition for the ungrasp operator.

The basic shortcoming of the STRIPS/NOAH (Sacerdoti [21]) representation of action interrelationships is that it underspecifies the role each act plays in the plan. Stated differently, the precondition/body/effect representation is not conducive to reasoning about actions in their context. For example in the definition of a plan comprising a pickup $\rightarrow$ ungrasp sequence, it is not explicitly specified just how the actions are related: for one thing, any realistic action operator is likely to have several side effects, besides the relevant one(s) for the purposes of the current sequence. In trying to reconstruct an agent's plan from one or more observed actions, an observer using this representation will encounter the problem of deciding how to "thread" the actions together. The view of plans as sequences of operators having lists of preconditions and effects is intuitive from a programmer's perspective, and pervades most of the early computer science literature on planning. Following Bratman [5] and Pollack [20], we refer to this as the "data structure" view of plans.

We alter the data-structure view of plans in order to achieve

[1]The parallel between utterances and physical acts is generally credited to Austin (see [4] in particular] in the philosophy of language literature. Its earliest occurrence in the computational literature is in Bruce [6], expanded upon in Cohen and Perrault [8]. While Austin's essential point was to see utterances as being "performative," or similar to physical acts, we adopt here the reverse view and model the interpretation of physical acts after that of natural language utterances.

[2]But see Lansky [14] for an event-based representation. The classical ("state-based") definition given previously runs into several problems, for example, in representing concurrent or continuous actions.

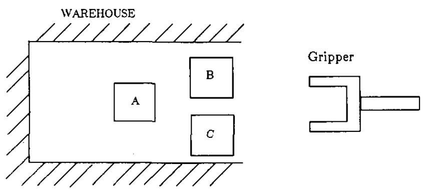


Fig. 3. Initial state.

```
Remove block_x:
        rem [x,y]        ;remember current position
        move_to block_x
        grasp block_x
        move_to [10000.0,y]
        rotate(180)
        ungrasp
        move_to [x,y]
        rotate(180)
```

Fig. 4. Remove procedure.

what Bratman and others have called the mental phenomenon view of plans (Bratman [5, pp. 28–35], or Pollack [20, chs. 2 and 3]). The latter view is chronologically posterior to the data structure view. It was recognized even as researchers were representing plans as sequences of NOAH operators (see for example Allen [2]). What the mental phenomenon view of plans emphasizes is the configurations of beliefs and intentions that must underlie any plan of rational action. We earmark plans by their underlying *intentions,* and use that information for later reasoning about and re-interpretation of plans.

In summary, the present work attempts to go beyond the traditional data structure view of plans in two ways. First, it relates the plan of action to the concrete situation surrounding it. This requires an understanding of the relevance of each action to its corresponding situation. Second, whereas most previous work in planning essentially models plan languages more or less after formal programming languages, Pollack's example is followed here in carefully building the action representation on foundations laid (informally) by philosophers of action. By combining these two steps, we obtain a system able to interpret and execute plans not just mechanically, by following a program counter, but in relation to their surroundings.

B. Enablement

In a sequence of actions forming a plan Π, some of these actions may be related to each other in a (yet to be defined) causal sense, while others may not. Consider, for example, the situation depicted in Fig. 3. The goal here is to have block-A outside the warehouse. The robot is constrained in its workspace by the walls of the warehouse and, in this example, in its motion by the two obstacles (blocks B and C). A plausible plan of action consists in removing block B, removing block C, and then getting block A out of the warehouse. Let us assume that a nonprimitive "Remove" procedure has already been defined as shown in Fig. 4: it assumes its argument is a movable object in the environment to which the robot has clear access from its current position; the robot grasps that object and slides it along the row into oblivion, and then returns to the row entry point. The plausible plan to get A out of the warehouse is given by the operator as

$$\Pi/S_0 = \{\alpha_1 \rightarrow \alpha_6\} \tag{1}$$

where

$$\begin{aligned}
\alpha_1 &\equiv (\text{Remove block-}B)\\
\alpha_2 &\equiv (\text{Remove block-}C)\\
\alpha_3 &\equiv (\text{move\_to block-}A)\\
\alpha_4 &\equiv (\text{grasp block-}A)\\
\alpha_5 &\equiv (\text{move\_to } [RE\_x, RE\_y])\\
\alpha_6 &\equiv (\text{move\_to} <\text{target\_pt}>).
\end{aligned} \tag{2}$$

This should be read as "the plan Π defined in initial situation S_0 and consisting of the temporal sequence α_1 to α_6." Focus for the moment on the first 3 acts: α_1, α_2, and α_3. It is evident that α_1 and α_2 belong to the category of acts in a plan that are not causally related to each other. α_1 and α_2 are independent, and it is reasonable to assume that the operator is not committed to any particular ordering of α_1 and α_2. Furthermore, the performance of one leaves the other totally unaffected, save for occupying the robot's resources temporarily. As a matter of fact, if we assume a robot with infinite resources, then α_1 and α_2 might as well be carried out simultaneously without disrupting the rest of the plan. Contrast the relation of α_1 to α_2 with that of α_2 to α_3: α_2 and α_3 are also in simple sequence and they do not overlap, but they are certainly not exchangeable. Even more important, the performance of α_2 is vital to that of α_3: a fact that does not hold for the first pair. The relationship of α_1 to α_2 is one of simple concatenation. This relationship is rather trivial, and is of little interest for our purposes. We shall return to it later, however, when we come to examine acts that are enabled by several of their predecessors.

In contrast to concatenation, the relation of enablement has the following necessary properties.

1) The enabling act, when and if it occurs, must be prior to the enabled act.
2) The enabled act is not executable by the agent prior to the occurrence of the enabling act.
3) The enabled act becomes executable by the agent after the occurrence of the enabling act.

To refer back to the example of Fig. 3, it is clear that prior to the occurrence of α_2, α_3 is not executable: even though α_3 is a move_to act, and moves are basic in the present robot domain, the robot does not have clear access to its destination point and cannot therefore execute the move. This situation is altered after the performance of α_2 since the only obstacle between the current robot position and block A is removed. What about the relation of α_1 to α_3? Clearly, it should also be an enabling relationship since α_1 and α_2 are similar and exchangeable. Yet, it fails the definition because the robot is still not in standard conditions with respect to α_3 after the occurrence of α_1. More generally, as given above, the definition fails to capture the enablement of an act by two or more enabling acts, and fails to capture instances of remote enablement links. The properties of the definition do not uniquely characterize enablement, i.e., they are not sufficient conditions for enablement. The second and third stages of recognition presented in the next section provide these conditions in an algorithmic fashion.

III. The Intention Recognition Algorithms

A. The Three Stages of Intention Recognition

The three-stage recognition algorithm identifies the relationships between actions of a given plan Π, and based on this information infers that acts in Π are enabling acts (in the sense introduced in the preceding section), and which acts are intended acts.

1) Enablement Necessary Conditions: The first stage of recognition accepts a plan definition given by the human operator as a linear sequence of commands, and attempts to recognize the enabling acts in the plan. The algorithm for this first stage corresponds to the necessary property of enablement: namely, that the enabling act places the agent in standard conditions with respect to the enabled act. To establish whether or not the enablement relationship holds between two successive acts α_i and α_{i+1} in the sequence $\{\alpha_j\}\ j = 1, \cdots, n$ that constitutes the plan Π, three mechanisms are needed.

1) A mechanism that captures the context V_i of an act α_i in Π: in other words, the state V_i of the world model—including the robot's perception of it—prior to the occurrence of α_i.
2) A mechanism that updates V_i to V_{i+1}: in other words, a means of simulating the occurrence of α_i without disturbing the actual model of the world.
3) A pointer from each domain act to its list of standard conditions that must be satisfied for the act to be executable. The pointer is passed to the recognition algorithm, and the latter evaluates whether or not an act is executable based on this information.

A description of how these mechanisms are actually implemented is postponed until Section IV. It should already be hinted, however, that these mechanisms afford considerable generality in that they are totally domain- and application-independent. Specifically, by capturing and updating the V_i's nothing more needs to be encoded to determine whether or not an agent is in standard conditions with respect to an act α_i. Illustrative examples will be provided in the next section.

With these three mechanisms in hand, the first stage of the recognition process is straightforward and corresponds directly to the second and third clauses in the definition of enablement as follows.

1) If α_i were an enabling step for α_{i+1}, then the robot (the "agent") would not be in standard conditions with respect to α_{i+1} prior to α_i:

$$\neg SC(\alpha_{i+1}, G, V_i)$$

The predicate SC denotes the "standard conditions" that the agent must be in to be able to perform a certain act: thus, the expression $SC(\alpha, G, V_i)$ indicates that the agent G is in standard conditions with respect to α in state V_i. See Yared [25] for a detailed explanation of the SC predicate.
2) if the above condition holds and the agent becomes in standard conditions with respect to α_{i+1} after the occurrence of α_i (or $SC(\alpha_{i+1}, G, V_{i+1})$) then we can assert enables(α_i, α_{i+1}).[3]
3) if, on the other hand, the agent is in standard conditions with respect to α_{i+1} even prior to α_i—or $SC(\alpha_{i+1}, G, V_i)$—then the enabling relation doesn't hold and α_i corresponds to a shift in the user's intention. α_i is identified as an intended act.
4) the last act in the plan, α_n is always assumed to be an intended act.

Note that under this interpretation any act α_i in Π is either an intended act or an enabling act, but cannot be both. All three recognition algorithms depend on this simplicity condition,

[3]If the agent is still not in standard conditions with respect to α_{i+1} after the occurrence of α_i then the plan is unexecutable and a warning is issued to the user.

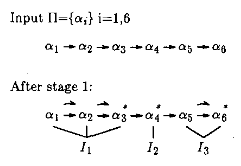
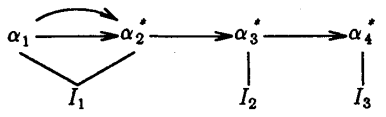


Fig. 5. Sample output from stage 1 of recognition.

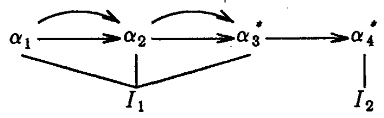
that each act plays only one role in the plan. Fig. 5 shows an unprocessed linear sequence of commands and its processed version after stage 1 of recognition. The intended acts are labeled with an asterisk and signal different intentional groups. The directed arcs denote enabling pairs. The intentional groups are indicated under their corresponding acts. It is possible for a single act to constitute an intentional group of its own.

2) Dealing with Pragmatic Ambiguity: As given, the first stage of recognition suffers from two main limitations. The first limitation concerns cases of pragmatic ambiguity and is treated presently.

The simplicity condition allows for only one role for each act in a plan. We will call plans where at least one act α_i plays more than one role instances of pragmatic ambiguity. In such plans, the system will only attribute one role to α_i, possibly one that is not its most important role in the plan. An example of a pragmatically ambiguous act can readily be seen in the situation of Fig. 6. The plan $\Pi = \{\alpha_1 \rightarrow \alpha_4\}$ depicted in Fig. 6 would normally carry the following interpretation:



In other words, moving to the target enables the robot to grasp it; grasping signals a shift in intention; grasping the block, moving to a conveyor belt, and ungrasping it are intended acts in their own right. However, the output from the first stage of recognition yields the following interpretation:



The reason for this misinterpretation is that the pair (α_2, α_3), when taken out of the context of the overall plan, does indeed have the central properties of an enablement pair: once the robot is stationed at the block (the situation depicted in Fig. 6), if told to move upward towards the conveyor belt without having first grasped the block it will simply collide with the block. Once block A is grasped, however, it is no longer a free-floating obstacle in the robot's trajectory, and the robot is then in standard conditions with respect to moving to the conveyor belt. Thus, there seems to be some justification for inferring that the grasp operation enables the subsequent move_to operation. Put in concrete terms, grasping a potential obstacle can be narrowly interpreted as a means of freeing the path of the robot. In this

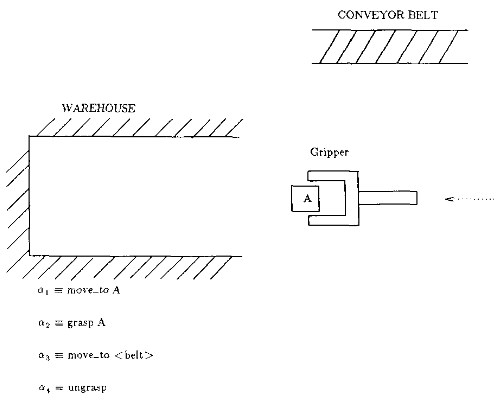

CONVEYOR BELT
WAREHOUSE
Gripper
A
$\alpha_1 \equiv$ move_to A
$\alpha_2 \equiv$ grasp A
$\alpha_3 \equiv$ move_to <belt>
$\alpha_4 \equiv$ ungrasp

Fig. 6. Case of pragmatic ambiguity (state of the world after occurrence of α_1).

case, of course, when one looks beyond the pair (α_2, α_3) it becomes apparent that α_2 plays quite a different role in the plan. Its purpose is not to enable the robot to move toward the conveyor belt, because that could have been achieved prior to moving to the block: $SC(\alpha_3, G, V_1)$ is true. There must be then some other reason for grasping the block, and we would rather obtain:

$$\begin{tikzcd}
\alpha_1 \arrow[rr] \arrow[rr, bend left] \arrow[dr, dash] & & \alpha_2^* \arrow[r] \arrow[dl, dash] & \alpha_3^* \arrow[r] \arrow[d, dash] & \alpha_4^* \arrow[d, dash] \\
& I_1 & & I_2 & I_3
\end{tikzcd}$$

The latter result is obtained by performing a more thorough check on the executability of intended acts. The pragmatic ambiguity is resolved by taking into account contexts V_j $(1 \leq j \leq n)$ in addition to the ones adjacent to the intended act α_i. The first recognition algorithm only took into account adjacent pairs of acts (α_i, α_{i+1}) and their surrounding and intervening contexts V_i and V_{i+1} respectively. To rectify that situation a second recognition stage is applied to the output of the first.

The second recognition algorithm performs essentially the same work as the first, with two main differences as follows.

1) It only analyzes acts the first stage identified as being intended
2) It performs a more exhaustive analysis on each α_i^* by examining all the contexts V_l that belong to the same intentional group as α_i^*.

The algorithm enters a loop and forward searches for the next intended act (α_j^*), provided there is one left. Once it has identified an intended act and its corresponding intentional group (I_{α_j}), it proceeds to test for the standard conditions of the intended act in the context of each enabling act in the intentional group in sequence $\left(SC\left(\alpha_j^*, G, \vec{V}_i\right)\right)$. If none of the tests is positive it then proceeds to the next intended act and intentional group; if one enabling act passes the test then the algorithm breaks the enabling link between that act and its successor, and establishes that act as an intended act.

Note that in the plan of Fig. 6, the pragmatic ambiguity is resolved by taking into account more contextual information. But there are plans where the ambiguity is more deeply entrenched and cannot be resolved by the second algorithm. Consider, for example, a variant of the plan in Fig. 6 where the robot is told to 1) move to A, 2) grasp A, 3) move to some point inside the warehouse. As before, the second algorithm outputs the following structure:

$$\begin{tikzcd}
\alpha_1 \arrow[r] \arrow[r, bend left] \arrow[dr, dash] & \alpha_2 \arrow[r] \arrow[r, bend left] \arrow[d, dash] & \alpha_3^* \arrow[dl, dash] \\
& I_1 &
\end{tikzcd}$$

But then the second algorithm leaves that structure intact, since it confirms that

$$\neg SC(\alpha_3, G, V_1) \Lambda \neg SC(\alpha_3, G, V_2) \Lambda SC(\alpha_3, G, V_3)$$

and, therefore, confirms enables(α_2, α_3). Such plans are truly ambiguous—as opposed to the previous one, in which the ambiguity is only apparent. They simply do not contain sufficient information to determine which role of the critical α_i is dominant, and for all practical purposes α_i can be taken to play several equally important roles in Π. It is not clear that a human observer could resolve the ambiguity based on a single instance of Π.

3) Multiple and Nonadjacent Enabling Acts: The second limitation of the definition of enablement is its inability to capture cases of multiple enabling acts, and the related cases of remote (non-adjacent) enabling acts. Fig. 3 exemplifies such situations. The plan for Fig. 3 consists of the following acts:

$$\Pi/S_0 = \{\alpha_1 \rightarrow \alpha_6\}$$
$$\alpha_1 \equiv \text{remove } B$$
$$\alpha_2 \equiv \text{remove } C$$
$$\alpha_3 \equiv \text{locate } A$$
$$\alpha_4 \equiv \text{move\_to } A$$
$$\alpha_5 \equiv \text{grasp } A$$
$$\alpha_6 \equiv \text{move\_to<start>}$$

where locate, move_to and grasp are appropriately defined domain-basic acts and remove some previously defined procedure. The principal feature of this plan is that α_1 and α_2 taken together co-enable α_4. The algorithm of Stage 1 is faced with the dual problem here of misunderstanding the relationship of α_1 to α_2—which is one of temporal concatenation only–and of missing completely that of the first two acts to the non-adjacent α_4. The output of the first recognition stage when applied to this plan yields

$$\begin{tikzcd}
\alpha_1^* \arrow[r] \arrow[d, dash] & \alpha_2^* \arrow[r] \arrow[d, dash] & \alpha_3 \arrow[r] \arrow[r, bend left] \arrow[dr, dash] & \alpha_4 \arrow[r] \arrow[r, bend left] \arrow[d, dash] & \alpha_5^* \arrow[r] \arrow[dl, dash] & \alpha_6^* \arrow[d, dash] \\
I_1 & I_2 & & I_3 & & I_4
\end{tikzcd}$$

What is desired is the following interpretation

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.