Execution Monitoring and Recovery Planning with Time

David J. Musliner   Edmund H. Durfee   Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

Focusing on Allen and Koomen's temporal planning scheme [1], this paper characterizes the possible locations for failure during plan execution. By observing the interval collapse criterion during plan generation, the planner is able to integrate the planning of goal-directed actions and execution verification actions. This allows the planner to schedule sensor usage and reason about sensing and data processing delays.

We also present a simple and robust recovery planning scheme which inserts corrective steps into the original plan. We generate purely inserted recovery plans efficiently, without destroying the original plan and without unnecessary temporal inferencing.

1 Introduction

To work in the real world, an AI system must often interact with a dynamic, time-constrained environment. Several planning systems have recognized the need to deal explicitly with deadlines, scheduled events, and actions that take time [1, 9, 12]. However, since real-world domains cannot be modeled perfectly, a system that predicts the results of planned actions in a simulated world must be able to detect and recover from discrepancies between its expected and actual state during execution. Verifying the correct execution of a plan can be difficult, because a variety of sensor inputs and expected condition changes must be monitored almost continuously.

Our execution monitoring and replanning techniques are described in the context of the planning and execution system shown in Figure 1. We emphasize the temporal aspects of execution monitoring which are not addressed by previous integrated and reactive systems [2, 5, 11]. Examples will be drawn from a blocks-world domain in which a robot ($\mathcal{R}$) performs the fine-grain actions Pick-Up, Move-Over, and Put-Down on three blocks ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$) and one table ($\mathcal{T}$). Figure 2 summarizes a simple temporal plan to stack $\mathcal{A}$ on $\mathcal{B}$.

The work reported in this paper was supported in part by the National Science Foundation under Grant DMC-8721492 and a NSF Graduate Fellowship.

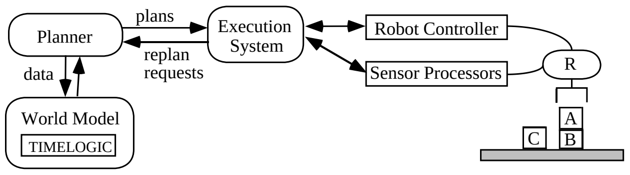

Figure 1: Overview of the system.

2 The World Model & Planner

The World Model (WM) portion of our system is responsible for maintaining a database of the system's believed and sensed knowledge about the world. The WM uses Koomen's TIMELOGIC system to implement Allen's interval temporal logic [1, 7]. In addition to the relational constraints which Allen defined to indicate relative order between intervals, TIMELOGIC supports durational constraints indicating relative magnitude between intervals. Our WM code associates world data with intervals and provides a data indexing scheme. The WM also includes several inference mechanisms which monitor the addition of data from the planner, enforcing domain constraints such as "no two blocks can be ON the same block at the same time." If a proposed addition to the WM violates either these domain constraints or the temporal constraints maintained by TIMELOGIC, the addition is rejected and the planner must chronologically backtrack to try a different search path.

Our planning system implements Allen and Koomen's algorithm [1], using constraint-based interval temporal logic to generate complex plans involv-

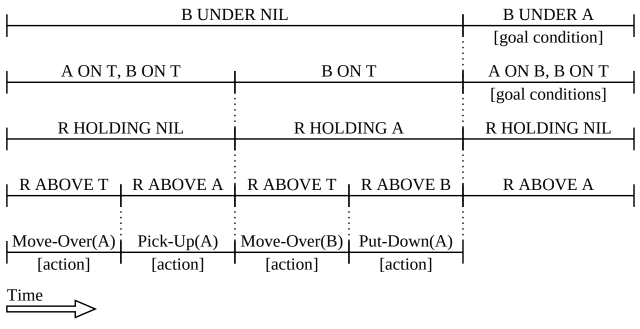

Figure 2: Temporal plan to stack $\mathcal{A}$ on $\mathcal{B}$. Dotted lines connect equal interval endpoints.

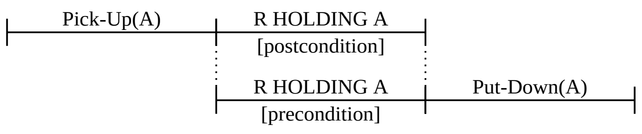

Figure 3: An interval collapse.

ing multiple agents, overlapping actions, and explicit time constraints like deadlines. Interval constraints also permit a "minimal commitment on order" strategy, so that temporal orderings are postponed as long as possible. The planning algorithm performs a depth-first search by backward-chaining from the goals to the initial world state.

Our implementation maintains a goal stack, and begins by examining (but not removing) the top goal and its interval. If an assertion in the world model is equivalent to the goal and its interval can be equal to the goal interval, we do not need to plan anything, the goal is already achieved— we simply express the temporal equality constraint and pop the goal from the stack (see Figure 3). This behavior is the only way to remove a goal from the stack, and corresponds to Allen and Koomen's "interval collapse" [1]. If no interval collapse is possible, we must plan a step to achieve the goal, adding its postconditions to the WM and pushing its preconditions onto the goal stack. Recursing on this algorithm via an iterative control scheme gives the requisite operator-subgoaling, depth-first search.

3 Execution Monitoring

Limited sensing and/or processing abilities may make it impossible to verify all of the expectations expressed by a plan [3, 9]. Thus, we must restrict execution monitoring as much as possible by addressing two important questions: what expectations must be verified, and how often must they be verified.

3.1 What To Verify?

When resource limitations are a consideration, we would like to restrict execution monitoring to only those assertions which are critical to the plan's success. Doyle, Atkinson, and Doshi noted that only postconditions which are used to satisfy preconditions are on the *critical path* representing the dependencies between actions [3]. Unfortunately, their state-based representation makes identifying these dependencies costly. Our temporal planning formulation makes the conditions on the critical path very easy to locate automatically: *only those conditions involved in interval collapses must be verified.*

Characterizing the critical conditions as interval collapses highlights two special cases which might otherwise be overlooked. Interval collapses occur when a plan's initial conditions are used to satisfy the preconditions of a plan step, and when the postconditions of a step achieve a final goal. Since significant planning time may elapse between the time the initial conditions are sensed and the time the first step of the plan is executed, the initial conditions may no longer be true. And since plans for conjunctive goals may attain one goal long before another, there is a chance that an achieved goal condition may be disrupted before the plan is completed. The interval collapse condition automatically requires verification for these special cases.

3.2 How Often To Verify?

Expected conditions might be verified only once, when they are achieved. However, conditions which have significant duration should be confirmed again before they are used as preconditions of later actions, since they may have been violated by an intervening unexpected event [3]. Even this double verification approach is undesirable if significant time elapses between the two verification actions, because unexpected conditions may not be detected quickly. For example, consider a robot which must grasp an object and move it thirty feet. If the robot only checks to make sure it is holding the object at each end of the trip, recovering an object dropped in transit may be quite expensive. On the other hand, the robot probably cannot afford to focus its sensors solely on checking that the object is still held. Using the temporal information in our plan representation, we must balance the frequency of monitoring various conditions against the sensor resource limitations.

3.3 Taking Verification Time into Account

One aspect of verification which other systems [2, 3] did not consider is the time cost of the monitoring actions themselves: sensing and processing sensed data can take significant time. Inserted verification actions can prolong the duration of a plan so that it no longer meets deadlines. Therefore, verification actions should be of concern to the planner itself, not relegated to a separate postprocessor as in the GRIPE and SAN systems [3, 4]. Just as regular plan steps use robots over significant time intervals, verification actions use sensors over time intervals.

We have modified the basic temporal planning system to plan monitoring actions and take verification time into account. The system sensors are explicitly represented in the WM. Each time an interval collapse is performed, we insert an appropriate verification action into the plan. Verification actions have preconditions and postconditions in the same way as regular plan steps. Figure 4 shows an example in which a verification action uses a vision system to confirm the condition (*B* UNDER NIL). The action has a postcondition indicating that it makes the camera BUSY when the action is in progress. WM domain constraint mechanisms ensure that only one action can make a resource (sensor or robot) BUSY at any time, so the

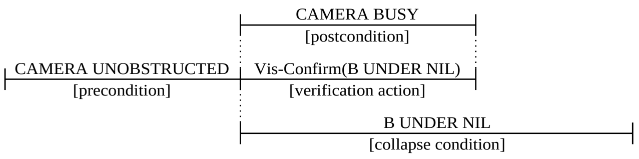

Figure 4: Example verification action.

planner only schedules one verification action to use a sensor at any one time. The example verification action also has a precondition which can be a candidate for subgoaling: if the camera's view is blocked, we create a subgoal to move the obstructing objects. Because the GRIPE and SAN systems plan verification in a postprocessor, they can only achieve this behavior by passing a subgoal back to the planner and requesting a plan modification. By fully integrating our verification planning and goal-directed planning, we avoid *ad hoc* methods and allow fine-grained interleaving of verification actions and regular actions.

Verification actions have durations determined by how rapidly the system can execute the corresponding sensor checks. When a verification action is planned, its interval is constrained to start with, and end before, the collapse interval. Thus, the collapse interval's duration must be at least as long as the verification action's (see Figure 4). After the planner creates an initial plan, each condition on the critical path has been assigned exactly one verification action.

When the plan is completed, the interval collapse durations have stabilized, and the planner can detect conditions whose durations are significantly longer than their single verification actions. Unlike the single and double verification strategies discussed in Section 3.2, our planner can exploit the temporal representation to schedule periodic verification actions for these persistent conditions. Sharing the limited sensor resources among the verification actions will require a sophisticated periodic task scheduling algorithm [8]. This algorithm must balance the frequency with which various concurrent conditions are monitored against the probability and recovery cost of an execution error.

In its current form, our approach to execution monitoring has one potential disadvantage: plans that cannot be verified because of sensor limitations are simply not produced. Every interval collapse must have a verification action successfully scheduled, or the planner backtracks to form a different plan.

4 Recovery Planning

When a verification action fails during plan execution, the execution system sends a replan request to the planner, listing the violated expectations and the unexpected conditions. Figure 5 shows an example replan request generated when an external agent unexpectedly drops $\mathcal{C}$ onto $\mathcal{B}$, violating the expectation that $\mathcal{B}$ is UNDER NIL before the action Move-Over($\mathcal{B}$) in Figure 2. The request includes the last step executed in the real world, so the planner can understand how far the execution system and the world have progressed in the plan.

4.1 Inserted Recovery Plans

We have focused on a recovery planning scheme in which new plan steps are inserted to make the world rejoin the original plan. Figure 6 shows an example of a *purely inserted recovery plan*, corresponding to the replan request in Figure 5. The recovery plan is inserted after the failure, moving $\mathcal{C}$ off of $\mathcal{B}$ and rejoining the plan where it was disrupted. Gini and Gini describe a similar recovery insertion technique [6], but their system only fires hand-coded recovery rules, and thus lacks the generality which our recovery method achieves by using the planner. We generate purely inserted recovery plans in three steps:

1. Push the WM conditions existing at the end of the last action onto the planner's goal stack, with the violated condition at the top. The recovery plan may alter existing conditions, but must restore them before rejoining the original plan.
2. Modify the WM state to match the real world state (i.e., alter it by the violated and unexpected conditions).
3. Run the planning algorithm, constraining the recovery plan to fit between the last executed step and the next step in the original plan.

4.2 Discussion of Inserted Recovery Plans

The inserted recovery plan technique has a number of advantages over other recovery planning methods. By retaining all of the future section of the original plan, we avoid destroying the work done in previous planning cycles. Thus, the insertion method minimizes the disruption of potentially large plans, making it more efficient than recovery techniques which retain little or none of the original plan [10].

Retaining the old plan also avoids the difficulty of removing sections of the plan which depend on violated conditions. Backward chaining planners create plans from the goal back to the initial state, so the first steps to be executed are the ones most recently planned. Thus chronological backtracking would begin by trying to remove plan steps which might already have

```
(request-replan
  :violated-condition ((B UNDER NIL) INTERVAL18)
  :unexpected-conditions ((C ON B) (B UNDER C))
  :last-action (Vis-Confirm (B UNDER NIL)))
```

Figure 5: Replan request for unexpected appearance of $\mathcal{C}$ on $\mathcal{B}$.

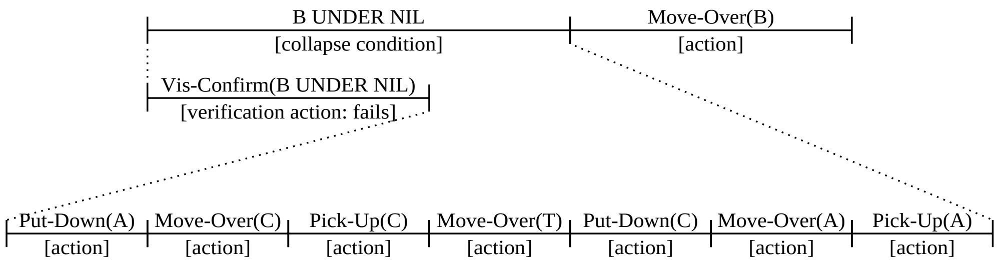

Figure 6: Purely inserted recovery plan for unexpected appearance of $\mathcal{C}$ on $\mathcal{B}$.

been executed [10]. Dependency-directed backtracking and case-based reasoning methods address some of these difficulties, but removing information that depends on a violated condition can be costly because it may force the system to re-derive previous plan information. Inserted recovery plans do not require delicate plan surgery or undoing actions, and they retain the understanding that an expectation violation has occurred.

Inserted recovery plans also avoid excessive temporal inferencing. Since the recovery plan is defined to be contained entirely within the violated condition's collapse interval, no temporal inferencing need be done between the recovery plan intervals and those outside of the collapse interval. Since temporal constraint propagation currently accounts for 91% of the planning time, avoiding temporal inferencing is a major consideration.

Inserting recovery plans is a highly robust recovery method: the technique can resolve any reversible execution error, if enough time is available. Irreversible errors like deadline violations cannot be fixed by inserted recovery plans, because a deadline violation requires a new plan which either makes up for the overrun or mitigates the resulting problems. Inserting a recovery plan also fails if the recovery sequence cannot be temporally constrained to fit within the collapse interval duration. In either case, more is required than simply rejoining the original plan. Note, however, that the planner recognizes when it cannot generate a purely inserted recovery plan. We may extend the system to attempt a more complex replanning technique when the simpler insertion method fails.

5 Current Status & Future Directions

We have implemented our planning and recovery planning methods in Common Lisp and CLOS. The system has successfully planned recovery from a variety of blocks-world scenarios involving unexpected events, including the examples presented in this paper. Unfortunately, the planner's memory requirements appear to scale exponentially with task complexity. We attribute this unacceptable performance to the temporal inferencing mechanism, rather than the methods we have introduced here. We will examine how these methods can be extended to other planning mechanisms which have superior scaling characteristics. Our current research is focused on developing and interfacing an execution system which carries out plans under hard real-time constraints.

References

[1] J. F. Allen and J. A. Koomen, "Planning Using a Temporal World Model," in *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 741–747, 1983.

[2] J. A. Ambros-Ingerson and S. Steel, "Integrating Planning, Execution and Monitoring," in *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 83–88, August 1988.

[3] R. J. Doyle, D. J. Atkinson, and R. S. Doshi, "Generating Perception Requests and Expectations to Verify the Execution of Plans," in *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 81–88, August 1986.

[4] E. Gat et al., "Path Planning and Execution Monitoring for a Planetary Rover," in *Proc. IEEE Conf. on Robotics and Automation*, 1990.

[5] M. P. Georgeff and A. L. Lansky, "Reactive Reasoning and Planning," in *Proc. Nat'l Conf. on Artificial Intelligence*, pp. 677–682, July 1987.

[6] M. Gini and G. Gini, "Towards Automatic Error Recovery in Robot Programs," in *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 821–823, 1983.

[7] J. A. Koomen, "The TIMELOGIC Temporal Reasoning System," in *University of Rochester Computer Science Department Technical Report 231*, 1989.

[8] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[9] D. P. Miller, *Planning by Search Through Simulations*, PhD thesis, Yale University, 1985.

[10] H. J. Porta, "Dynamic Replanning," in *Proc. Second Annual Workshop on Robotics and Expert Systems*, pp. 109–115, June 1986.

[11] M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 1039–1046, 1987.

[12] S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 3, pp. 246–267, 1983.