Detecting and Reacting to Unplanned-for World States

Ella M. Atkins **Edmund H. Durfee** **Kang G. Shin**

University of Michigan AI Lab
1101 Beal Ave.
Ann Arbor, MI 48109
{marbles, durfee, kgshin}@umich.edu

Abstract

The degree to which a planner succeeds and meets response deadlines depends on the correctness and completeness of its models which describe events and actions that change the world state. It is often unrealistic to expect perfect models, so a planner must be able to detect and respond to states it had not planned to handle. In this paper, we characterize different classes of these "unhandled" states and describe planning algorithms to build tests for, and later respond to them. We have implemented these unhandled state detection and response algorithms in the Cooperative Intelligent Real-time Control Architecture (CIRCA), which combines an AI planning system with a separate real-time system so that plans are built, scheduled, and then executed with real-time guarantees. Test results from the flight simulation domain show the new algorithm enables a fully-automated aircraft to react appropriately to certain classes of unhandled states, averting failure and giving the aircraft a new chance to achieve its goals. We are currently working to further improve CIRCA's planning system, and to extend our detection and response mechanisms to other classes of unhandled states.

1 Introduction

Autonomous control systems for real-world applications require extensive domain knowledge and efficient information processing in order to build and execute situationally-relevant plans of action. To enable absolute guarantees about safe system operation, domain knowledge must be complete and correct, plans must contain actions accounting for all possible world states, and response times to critical states must have real-time guarantees. Practically speaking, these conditions cannot be met in complex domains, where it is infeasible to preplan for all configurations of the world, if indeed they could even be enumerated. Realistic autonomous systems use heuristics to bound the expanded world state set, coupled with reactive mechanisms to compensate when unexpected situations occur.

In this paper, we focus on the question of how an autonomous system can know when it is no longer prepared for the world in which it finds itself, and how it can respond. We assume limited sensory and computational capabilities, and that a system will devote available resources to the accomplishment of its tasks. As a consequence, such a system will not notice unexpected occurrences in the world unless it explicitly has a task of looking for them. In other words, the system must satisfy absolute guarantees about safe operation in expected states, and must also be ready to recognize and respond to the unexpected.

To ground our discussion and empirically validate our solutions, we will consider issues in dealing with unexpected occurrences within the context of the Cooperative Intelligent Real-time Control Architecture (CIRCA) applied to the aircraft domain. CIRCA combines a traditional AI planner, scheduler, and real-time plan execution module to provide guaranteed performance for the control of complex real-world systems (Musliner, Durfee, and Shin). With sufficient resources and accurate domain knowledge, CIRCA can build and schedule control plans that, when executed, are assured of responding quickly enough to any events so that CIRCA remains safe (its primary task) and whenever possible reaches its goals.

When faced with imprecise knowledge and resource constraints, a CIRCA plan may not be prepared to handle all possible states. CIRCA may have planned actions to assure safety in some state but may not be able to progress toward a goal state. Or, it may have anticipated a state but, due to resource limitations, chose not to schedule actions to keep it safe in that state. Or, it may not even have anticipated that state because of knowledge base imperfections. The contribution of this paper is to articulate, more precisely, such different classes of unhandled states (Section 3), to describe methods to detect when a system reaches one of these states (Section 4), and to respond appropriately (Section 5). We argue that these capabilities are crucial for developing reliable intelligent real-time control systems, and we highlight how they improve CIRCA's performance for simulated aircraft control (Section 6).

2 Background

The CIRCA system (Musliner, Durfee, and Shin) was designed to provide guarantees about system performance even with limited sensing, actuating, and processing power. When controlling a complex system in a dynamic environment, a real-time system may not have sufficient resources to be able to react in all situations. Based on user-specified domain knowledge, CIRCA builds a plan to keep a system "safe" (i.e., avoid catastrophic failure) while working to achieve its goals. CIRCA then uses its knowledge about system resources to build a schedule that guarantees meeting critical deadlines. This schedule is then executed on a separate real-time processor.

Figure 1 shows the general architecture of the CIRCA system. The *AI subsystem (AIS)* contains the planner and scheduler. The "shell" around all AIS operations consists of meta-rules controlling a set of knowledge areas, similar to the PRS architecture (Ingrand and Georgeff). Working memory contains tasks to be executed, including planning, scheduling, and downloading plans from the AIS to the *real-time subsystem (RTS)*, which executes the scheduled plan.

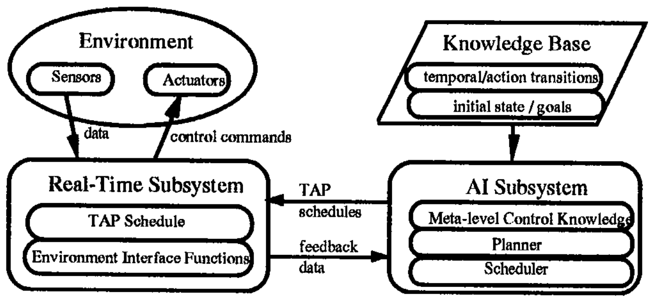


Figure 1. CIRCA Architecture.

The CIRCA *domain knowledge base* includes a set of *transitions* which model how the world can change over time, specification of the *initial* (or startup) *state*, and a list of all *subgoals* which, when achieved in order, will enable the system to reach its final goal -- the last subgoal in this list. To minimize domain knowledge complexity, the *world model* (i.e., reachable set of states) is created incrementally based on initial state and available transitions. The planner uses traditional methods of selecting actions based on estimated utility (i.e., cost vs. benefit) and backtracks if the action does not ultimately help achieve a subgoal or avoid catastrophic failure (e.g., aircraft crash). CIRCA minimizes memory and time usage by expanding only states produced by transitions from initial states or their descendants. State expansion terminates whenever all specified goals have been reached while avoiding all failure states.

A CIRCA knowledge base contains two transition types: action and temporal. All CIRCA transitions have a set of *preconditions*, feature-value pairs that must be matched before that transition can occur, and a set of *postconditions*, feature-value pairs that will be true after that transition occurs. *Action transitions* correspond with commands that CIRCA may explicitly execute (e.g., put aircraft landing gear down), while *temporal transitions* correspond to state changes not initiated by CIRCA (e.g., collision-course air traffic appears). Each temporal transition has an associated probability function that models that transition's likelihood as a function of time, starting at the moment all that transition's preconditions become true. Time remaining until a transition will occur is particularly important when a *temporal transition to failure (TTF)* is involved. In this case, CIRCA must schedule an action that will be guaranteed to execute before that temporal transition has a chance of occurring, effectively preempting the TTF (thus avoiding the catastrophic situation). CIRCA "plays it safe" by assuming the action must be guaranteed to occur before the TTF has more than some small probability ε. The use of probabilistic models in CIRCA is described in (Atkins, Durfee, and Shin).

Once CIRCA has finished expanding the set of reachable states, a list of planned actions and states in which to execute each of these actions is compiled. This list is called a *control plan* and is represented as a set of *test-action pairs (TAPs)*. Typical tests to detemine system state involve reading sensors and comparing the sensed values with certain preset thresholds, while actions involve sending actuator commands or transferring data between CIRCA modules. When the AIS planner creates a TAP, it stores an associated execution deadline, which is used by a *deadline-driven scheduler* (Liu and Layland) to create a periodic TAP schedule that guarantees system safety when TTFs are present. If the scheduler is unable to create a schedule to support all deadlines, the AIS backtracks to the planner. For the next planner iteration, the lowest probability temporal transitions are removed to reduce the number of actions planned. If the scheduler is still not able to build a schedule when only relatively high probability transitions are considered, the CIRCA planner fails, leaving the RTS executing its last plan which will ideally keep the system "safe" but never reach the goal.

Presuming the planner and scheduler are successful, the AIS downloads the TAP plan to the RTS. During normal operation, the RTS sends only handshaking messages to the AIS. This paper describes the introduction of RTS state feature feedback to prompt AIS replanning when an unhandled state is detected.

3 Unhandled State Classes

Figure 2 characterizes the relationships between subclasses of all possible world states for any domain. At the top level, states are either "modeled" or "unmodeled". Modeled states are those whose distinguishing features and values are represented in the planner's knowledge base. Because the planner cannot consider unmodeled states without the addition of a feature discovery algorithm, unmodeled states are beyond the scope of this paper.

Within the modeled set, the "planned-for" states include those the planner has expanded. This set is divided into two parts: "handled" states which avoid failure and can reach the goal, and "deadend" states which avoid failure but cannot reach the goal with the current plan.

Aside from the "planned-for" states, a variety of other states are modelable by the planner. Such states include those identified as reachable, but which have been "removed" because attending to them along with the "planned-for" states exceeds the system's capabilities. Other modeled states include those that indicate "imminent failure;" if the system enters these states, it is likely to fail shortly thereafter. Note that some states might be both "removed" and "imminent-failure", as illustrated in Figure 2. Finally, some modeled states might not fall into any of these categories, such as the states the planner considered unreachable from the initial states but that are not necessarily dangerous. We are working to find other important classes or else show no other modelable state classes are critical to detect.

The boldly outlined region in Figure 2 illustrates a state set reached during plan execution. To assure reaching the desired goal, the set should be empty except for where it overlaps the "handled" region. To assure safety, the set should only have elements that are in the "planned-for" region. When the set has elements outside these regions, safety and performance depend on classifying the new state and responding appropriately. For this reason, we next provide more detailed definitions of the most important classes.

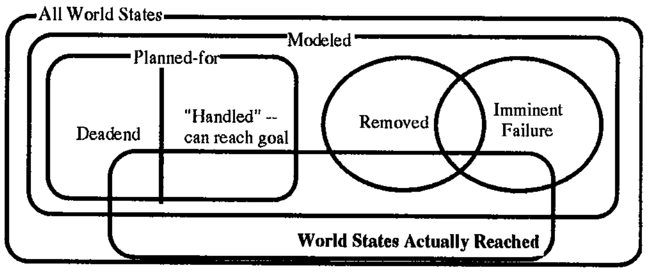


Figure 2. World State Classification Diagram.

A "deadend" state results when a transition path leads from an initial state to a state which cannot reach the goal, as shown in Figure 3. The deadend state is "safe" because there is no transition to failure. However, the planner has not selected an action that leads from this state via any path to the goal. As illustrated by a flight simulation example (Section 6), deadend states produced because a planner could find no action leading to a goal are called "by-necessity", while those produced because the planner simply did not choose an action leading to the goal from this state are called "by-choice" deadend states.

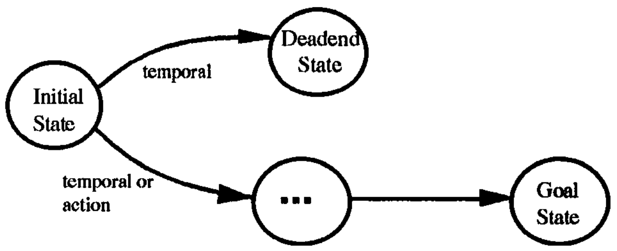


Figure 3. "Deadend state" illustration.

A planner that generates real-time control plans needs to backtrack whenever a plan cannot be scheduled. Backtracking prompts the planner to select different actions while maintaining those required to avoid failure. However, even after exhaustive backtracking, a planner may fail to find actions that meet all objectives while still being schedulable. One option is ignoring some reachable states, thus not planning actions for them. A control plan so constructed cannot claim to be foolproof. However, for real-time control applications, it is often judged more important to make timing guarantees under assumptions that exceptional cases will not occur than to make no guarantees about a more inclusive set of cases.

One heuristic for deciding which states to prune would be to overlook states that are least likely to occur. A "removed" state set, therefore, is created when the planner has purposefully removed the set of lowest probability states during backtracking, as illustrated in Figure 4. In the first planner iteration, all temporal transitions are kept during state expansion. The "Before Pruning" diagram in Figure 4 shows a state diagram in which both low and high probability states are kept. As depicted in the figure, a low probability transition leads to a state which transitions to failure. This failure transition is preempted by a guaranteed action. Suppose the scheduler fails. The planner will then backtrack and build a new plan without low-probability states. The resulting state diagram -- "After Pruning" -- is shown in Figure 4. Due to the one low probability transition, all states downstream of this transition are removed from consideration. The preemptive action no longer needs to be planned, giving the scheduler a better chance of success. A flight simulation domain example with removed states is shown in Section 6.

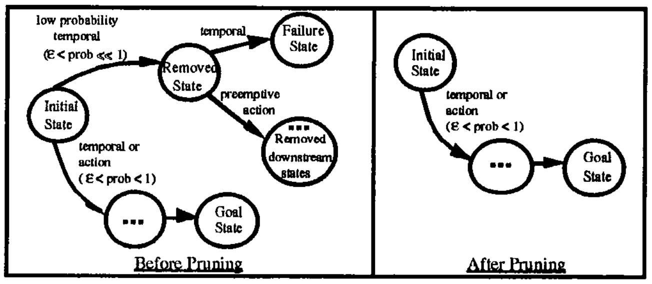

low probability temporal (ε < prob << 1)
temporal
Failure State
Removed State
preemptive action
Initial State
... Removed downstream states
temporal or action (ε < prob < 1)
...
Goal State
Before Pruning
Initial State
temporal or action (ε < prob < 1)
...
Goal State
After Pruning

Figure 4. "Removed state" illustration.

During plan development, all temporal transitions to failure from reachable states are preempted by guaranteed actions. If preemption is not possible, the planner fails. However, the planner does not worry about transitions to failure from any states it considers unreachable from the initial state set. The set of all modelable states considered unreachable that also lead via one modeled temporal transition to failure are labeled "imminent-failure".[1] Actually reaching one of the recognizable imminent-failure states indicates either that the planner's knowledge base is incomplete or incorrect (i.e., it failed to model a sequence of states that in fact was possible), or that the planner chose to ignore the possibility of this state occurring in order to make other guarantees.

Figure 5 shows a diagram of a reachable state set along with an isolated state (labeled "Imminent-failure") leading via one temporal transition to failure. This isolated state has no incoming transitions from the reachable state set, thus the planner will not consider it during state expansion. However, if this state is actually reached, the system will fail if the outgoing temporal transition occurs. The imminent-failure unhandled state set is important to detect because avoiding system failure is considered a primary goal of CIRCA. Examples of imminent-failure states present in flight simulation tests are described in Section 6.

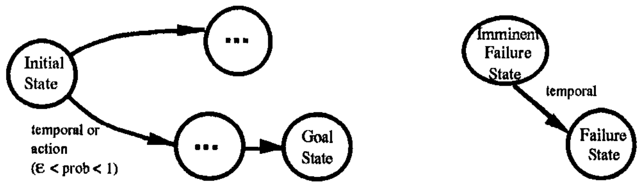


Figure 5. "Imminent-failure state" illustration.

4 Building Tests to Detect Unhandled States

A critical premise in our work is that a planner, or system in which it is embedded, cannot be expected to somehow just "know" when it has deviated from plans---it must explicitly plan actions and allocate resources to detect such deviations. For example, to make real-time guarantees, CIRCA's AI planner must specify all TAPs the RTS will execute, including any to detect and react to unhandled states. In our implementation, after the planner builds its normal plan, it builds special TAPs to detect each of the deadend, removed, and imminent-failure unhandled state classes. First, the planner builds separate lists of all states that fall into each unhandled state class. The algorithms to build the lists for deadend, removed, and imminent-failure states are described below. The TAP tests for an unhandled state class *could* include an explicit test for every set of state features in that unhandled state class list, but these tests would be repeated frequently during plan execution and may be unnecessarily time-consuming with long state lists. Thus, once each list is completed, the planner calls the ID3 test minimization algorithm (Quinlan) with that unhandled state list as the set of positive examples and a subset of the reachable states (depending on the unhandled state type) as the set of negative examples. ID3 returns what it considers a minimal test set, which is then used as the special TAP test to detect that unhandled state class.

When any of the three unhandled state detection TAP tests are active, the RTS action is to feed back current state feature information to the AIS along with a message stating the type of unhandled state detected. The AIS uses this feedback to build a new TAP plan which will be able to handle this state. The algorithm by which the AIS replans is described in Section 5.

Deadend states are the subset of the reachable state set from which no path to the goal state is possible with the planned actions. To identify the deadend state set, CIRCA follows transition links from each state in search of a goal state. If no goal path is found, that state is labeled "deadend" while he reachable states along a goal path are labeled "non-deadend". The deadend states are used as positive ID3 examples while non-deadend reachable states are used as negative examples when building a minimal feature test set for the deadend state detection TAP.

Each temporal transition has an associated function that describes its probability of occurring as a function of the time, starting when that transition's preconditions were satisfied. As discussed in Section 3, whenever backtracking due to scheduling difficulties or the planner's inability to find any working plan, transitions with probabilities below a non-zero threshold (ε) may be eliminated, pruning the overall state and action set and giving the planner and scheduler a better chance of success. To build the "removed" state list, the planner executes its state expansion routine using the entire reachable state set as "initial states". Both low and high probability temporal transitions are considered, although no high probability transitions will result in a new removed state except downstream from the application of a low probability transition. Only the planned action transitions are used to build new states, using the associated minimized TAP test conditions to determine if that action will occur in each state. The result of this state expansion procedure is a list of states containing both the original reachable states and the new low-probability or "removed" states that were not considered in the original plan. To build the removed state detection tests, ID3 is called with the list returned from state expansion (minus the reachable states) used as positive examples and all reachable states from the original plan used as negative examples.

[1] Note that it is also possible that unmodelable states could lead directly to failure with a known transition, or that modelable states could lead directly to failure with transitions that are not known to the planner, or that unmodelable states could lead directly to failure with an unknown transition. We exclude these cases from the "imminent-failure" set because the planner is incapable of classifying them in this way.

While the planner should look for deadend and removed states because they are more likely to occur than other unhandled states, likelihood is not the only criterion for allocating resources to detection. No matter how unlikely, imminent-failure states are important to detect because of the potentially catastrophic consequences of being in such states. When building imminent-failure state sets, we assume the modeled set of temporal transitions leading immediately to failure is complete and correct, even though having gotten to an imminent-failure state may imply that some temporal or action transition is not accurately modeled. To build the imminent-failure state list, the AIS begins with a list of all precondition feature sets from transitions leading directly to failure (i.e., transitions whose postconditions violate some safety condition). Next, this list is expanded to fully enumerate all possible states that would match these preconditions. Any reachable states present in the list are removed. The minimized imminent-failure detection TAP tests are then built by calling ID3 with this "imminent-failure" state list as positive examples and the list of reachable states as negative examples. Note that a complete list of fully instantiated states can be very large when many general preconditions lead to failure, but this list must be fully-instantiated before the basic ID3 algorithm can succeed. We are still looking for ways to efficiently build a minimal feature test set, perhaps using an "anytime" (Dean, Kaelbling, Kirman, and Nicholson) version of ID3 or another classification system.

5 System Reaction when an Unhandled State is Detected

Regardless of whether it is a "deadend", "removed", or "imminent-failure" state, any unhandled state has the potential to prevent the RTS from ever reaching its subgoal with the executing TAP plan. Additionally, removed or imminent-failure states will likely result in system failure if no appropriate action is taken. To increase CIRCA's goal achievement and failure avoidance, the AIS replans to account for any unhandled states that have been detected.

When one of the special TAPs described in Section 4 detects any class of unhandled state, the RTS feeds back all state feature information to the AIS.[1] Upon receipt of this message, the AIS generates the equivalent of an interrupt that is serviced as soon as possible, in the following sequence. First, the AIS reads the type of feedback message (e.g., "deadend"). This message type label will allow future versions of CIRCA to run different procedures for the different unhandled state classes, although all three are treated the same now. Next, the AIS reads the uplinked state feature values and selects a subgoal (from the knowledge base list) that is closest to the final goal but with preconditions matching the uplinked state features. Next, the AIS runs the state expansion part of the planner, using the state feature feedback as the initial state, all temporal transitions, and the executing plan's test-action (TAP) transitions. The state list returned from this state expansion routine contains all possible states the RTS could reach *while* the AIS is replanning,[2] thus each is a possible initial state when the new plan begins executing. The AIS then replans using this potentially large initial state set and the selected subgoal. This new plan is downloaded to the RTS which will then have the ability to react to the previously unhandled state and its descendants (i.e., the constructed initial state set).

By detecting all unhandled states which may reach failure,[3] the system will always be able to initiate a reaction to avoid impending doom. However, this is predicated on the planner being able to return a control plan to avert disaster faster than disaster could strike. For the purposes of the examples within this paper, we assume that this is the case: in the aircraft domain, we assume that the plane is at sufficient altitude and distance from the runway so that a new plan is constructed before the plane crashes "gear up" during touchdown. However, in other situations the system might have less opportunity to postpone disaster. We suggest methods to build a more robust imminent-failure state handling system by adding failure-avoidance actions to a control plan whenever possible and by limiting replanning time (Section 7).

6 Testing in the Flight Simulation Domain

We chose to test our unhandled state classification, detection, and reaction ideas in an aircraft flight simulator. In this section, we describe motivations for selecting the flight domain and discuss how the aircraft is controlled by CIRCA. We then present examples detecting and removing unplanned-for states, and describe how our algorithms improve CIRCA performance.

Perhaps the main attraction to the aircraft domain is that continuous real-time operation is essential -- an aircraft can never "stop and remain safe indefinitely" once it has left the ground. Fully-automated flight is a complex problem which has only been solved when an aircraft is restricted to certain regions of state-space, meaning only a special set of anomalies beyond normal flight can be handled autonomously. The "solved" part of flight can be modeled in a CIRCA domain knowledge base, then CIRCA's AIS can create plans that will issue commands to a low-level control system. With the addition of the unhandled state feedback described in this paper, CIRCA begins to reach beyond traditional aircraft autopilots, allowing the system to select a new plan (i.e., trajectory) which may not reach the previous goal but that allows the system to avoid failure and divert to a new location if necessary.

[1] Even though the RTS senses feature values in sequence, we assume none will change before the state is uniquely determined.

[2] We assume no further modeling errors during this state expansion, thus no possible initial states are ignored.

[3] Presuming we have modeled all actual transitions to failure.

We interfaced the ACM F-16 flight simulator (Rainey) to a low-level Proportional-Derivative (P-D) control system (Rowland) which calculated the proper actuator commands to achieve a commanded altitude and heading. CIRCA's RTS issued commands to the low-level controller, not directly to the simulator. We believe assisting CIRCA with standard techniques from the control field is wise because this allows a simpler knowledge base with high-level discrete-valued features. Modeled state features include altitude, heading, location (or "FIX"), gear and traffic status, and navigation sensor data. CIRCA successfully controlled the aircraft during normal "flight around a pattern" (illustrated in Figure 6) from initial takeoff through a full-stop landing on the runway.

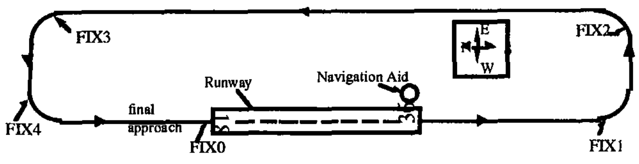
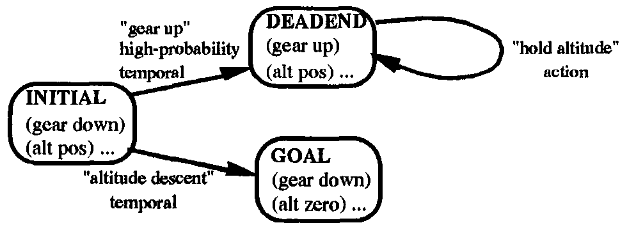


Figure 6. Flight pattern flown during simulation.

We have tested our new algorithms using two emergencies: "gear fails on final approach", and "collision-course traffic on final approach". In either situation, failure to notice and react to the problem will result in a crash. To detect these problems, we included feature "gear" with values "up" and "down", and feature "traffic" with values "yes" and "no". Note that more complex feature-value sets would work, but these simple examples illustrate the utility of our algorithms. Due to space limitations, we will discuss only "gear failure" here; similar results from the "collision-course traffic" emergency were obtained and are discussed in (Atkins, Durfee, and Shin).

During normal flight, the gear was left in the "down" position. A temporal transition to failure occurred with gear "up" when landing to simulate the impending crash. Proper reaction to a "gear up" emergency is to execute a go-around (i.e., continue around the pattern a second time to avoid an immediate crash), performing any available actions such as cycling the gear control to help the gear extend. A deadend "gear up" state was created using a high-probability temporal transition with precondition "gear down" and postcondition "gear up". Figure 7 illustrates the resulting deadend state with planned action "hold positive altitude" selected to avoid failure. For brevity, only two of seven state features are listed.

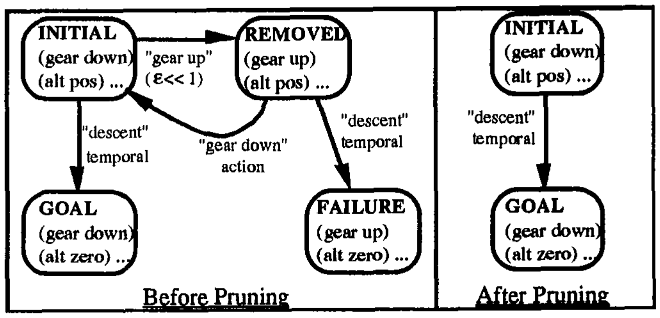


"gear up" high-probability temporal
DEADEND (gear up) (alt pos) ...
"hold altitude" action
INITIAL (gear down) (alt pos) ...
"altitude descent" temporal
GOAL (gear down) (alt zero) ...

Figure 7. Flight Simulation with Deadend State.

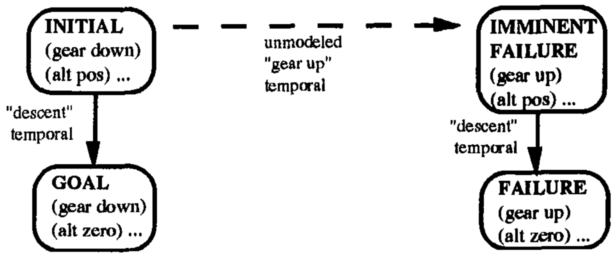
A removed "gear up" state set was created by including a low-probability "gear up" temporal and a time-consuming action to put the gear back down. The initial AIS plan included a "gear up" temporal along with a "gear down" action. This action needed to be guaranteed for the final approach region, and the scheduler failed because the action when combined with other critical approach-to-landing operations was too time-consuming. Thus the planner pruned all low-probability states, and they became "removed" states. Figure 8 shows a partial state diagram illustrating simplifications due to low-probability state removal.

An imminent-failure "gear up" state set was created by having an initial state with gear down and omitting any temporal transitions from "gear down" to "gear up". A "gear down" action was included in the knowledge base. Without any temporal transition leading to "gear up", the planner had no reason to expand any states with the "gear up" feature value, so no "gear up" action was planned. However, since attribute "gear up" led directly to failure, an imminent-failure-state detection TAP was built so that the RTS could detect the "gear up" state when it occurred. Figure 9 shows the reachable state set along with the "gear up" imminent-failure state.



INITIAL (gear down) (alt pos) ...
"gear up" (ϵ<< 1)
REMOVED (gear up) (alt pos) ...
"descent" temporal
"gear down" action
"descent" temporal
GOAL (gear down) (alt zero) ...
FAILURE (gear up) (alt zero) ...
Before Pruning
INITIAL (gear down) (alt pos) ...
"descent" temporal
GOAL (gear down) (alt zero) ...
After Pruning

Figure 8. Flight Simulation with Removed States.



INITIAL (gear down) (alt pos) ...
unmodeled "gear up" temporal
IMMINENT FAILURE (gear up) (alt pos) ...
"descent" temporal
"descent" temporal
GOAL (gear down) (alt zero) ...
FAILURE (gear up) (alt zero) ...

Figure 9. Flight Simulation with Imminent-failure States.

Test results for each of the unhandled state subclasses using the "gear up" emergency are summarized in Table 1. For each test, the aircraft flew around the pattern, and the gear unexpectedly went up just after starting final approach past FIX4. The table describes the events occurring after the gear goes up. Rows corresponds to test runs, while columns describe the test features. The "Gear down action" and "Gear up temporal" columns depict whether those transitions were present in the knowledge base for that test run. Column 4 indicates whether the unhandled state algorithms were used, followed by the "Detected state type" column indicating which unhandled state class (if any) was detected during that test. Output columns show a brief summary of the result (column 5), and finally, an indication of whether the plane crashed or landed safely (column 6).

In summary, whenever the "gear up" emergency occurred but went undetected (tests 1 and 2), the aircraft crashed, regardless of the existence of a "gear down" action. If the "gear up" emergency was deadend "by necessity" (tests 3 and 4), the aircraft lands safely after a go-around only if the gear serendipitously extends during a go-around (which has happened on occasion). In all other cases (tests 5 through 7), detecting and reacting to the gear problem enables CIRCA to execute a go-around and subsequent "gear down" action, resulting in a successful landing.

As shown in Table 1, detecting the "gear up" problem only increases the aircraft's chance to successfully land, particularly when the "gear down" action is functional, thus this simple example clearly illustrates how CIRCA's performance can improve with the detection of unhandled states. However, the "gear up" and "collision-course traffic" anomalies are relatively simple to model. We are working to show that CIRCA can function properly when more dynamically complex or interrelated features produce unhandled states. This requires modification of the low-level P-D control law because extreme flight attitudes are not controllable with such a simple linear system. We also want to show that CIRCA can react properly when more than one unhandled state is detected during execution of a single plan. We plan to show this by adding more features to the knowledge base and creating scenarios in which transitions produce multiple unhandled states during final approach.

Despite numerous improvements that must be made before we declare our fully-automated aircraft "ready-to-fly" (except in simulation), our experiments demonstrate the advantages of detecting and responding to unhandled states. We expect flight simulation to provide many challenges during future testing.

Table 1. Simulation Test Results with "Gear Up" Emergency.

Input					Output:	
Test #	"Gear-down" action	"Gear up" temporal	Detection algorithm used	Detected state type	Result	Crash?
1	N/A	No	No	None	Lands "gear up" on runway	Yes
2	N/A	Yes	No	None	Flies straight ahead, avoiding "gear-up" landing until out of fuel	Yes
3	No	Yes	Yes	Deadend by necessity	Gear never extends; plane executes go-arounds until out of fuel	Yes
4	No	Yes	Yes	Deadend by necessity	Gear locks itself during go-around(s)	No
5	Yes	Yes	Yes	Deadend by choice	Plane executes "gear down" during go-around	No
6	Yes -- slow execution	Yes -- low probability	Yes	Removed	Plane executes "gear down" during go-around	No
7	Yes	No	Yes	Imminent-failure	Plane executes "gear down" during go-around	No

7 Summary and Future Work

Plans built for complex domains may not handle all possible states due to either imperfections in domain knowledge or approximations made to enhance planner efficiency. We have identified important classes of such unexpected (or unhandled) situations, and for each of these classes have described a means by which detection tests can be generated. When these tests detect an unhandled state, the features of the state are fed back to the planner, triggering replanning. To implement these techniques, we modified CIRCA's planner to create a group of special test-action pairs (TAPs) which would detect three classes of unhandled states: "deadend" states from which no action could reach the subgoal, "removed" states which were low probability states not included in planning primarily due to scheduling constraints, and "imminent-failure" states that led directly to failure but were not considered during planning because no transition sequence led to them from the initial state set. When the RTS detects one of these unhandled states, it uplinks all state features to the planner, which then identifies a new subgoal and initial state set and replans.

These algorithms were tested using the ACM F-16 flight simulator. A series of test runs provided examples of each possible unhandled state type: deadend, removed, and imminent-failure. A comparison of tests with and without the unhandled state detection/reaction routines demonstrated that the aircraft had a better chance to land safely when the unhandled states were detected than when they were ignored.

This paper has identified capabilities that need further development before CIRCA can consistently succeed when reaching any unhandled state. First, we have developed algorithms that identify and detect three specific unhandled state subclasses. Other classes may also be important. A simple solution to this problem is building a TAP to generically detect any state that is not in the reachable set. However, classification algorithms such as ID3 cannot be used to minimize test conditions without enumerating both positive and negative examples, and building the set of all modelable states could be prohibitively time-intensive.

CIRCA's reaction to unhandled states is coincidentally real-time, but this may not be acceptable when unhandled states quickly lead to failure. Timely reactions may be achieved either by bounding replanning execution time or by building reactions in advance. As planning technology progresses, more architectures employ methods for bounding planner execution times ((Dean, Kaelbling, Kirman, and Nicholson), (Hendler and Agrawala), (Ingrand and Georgeff), (Zilberstein)). Implementing time versus quality tradeoffs in the CIRCA planner is considered in (Musliner, Durfee, and Shin), but no tradeoff algorithms have been implemented to-date.

8 Acknowledgements

This work was supported under NSF grant IRI-9209031.

9 References

E. M. Atkins, E. H. Durfee, and K. G. Shin, "Plan Development using Local Probabilistic Models," to appear in *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, August 1996.

T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, "Planning with Deadlines in Stochastic Domains," *Proceedings of AAAI*, pp. 574-579, July 1993.

J. Hendler and A. Agrawala, "Mission Critical Planning: AI on the MARUTI Real-Time Operating System," in *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, pp. 77-84, November 1990.

F. F. Ingrand and M. P. Georgeff, "Managing Deliberation and Reasoning in Real-Time AI Systems," in *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, pp. 284-291, November 1990.

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.

D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans", *Artificial Intelligence*, vol. 74, pp. 83-127, 1995.

J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81-106, 1986.

R. Rainey, *ACM: The Aerial Combat Simulation for X11*. February 1994.

J. R. Rowland, *Linear Control Systems: Modeling, Analysis, and Design*, Wiley, 1986.

S. Zilberstein, "Real-Time Robot Deliberation by Compilation and Monitoring of Anytime Algorithms," *Proceedings of AAAI*, pp. 799-809, 1994.