

## Using Model Checking to Plan Hard Real-Time Controllers

Robert P. Goldman and David J. Musliner

Honeywell Technology Center  
3660 Technology Drive  
Minneapolis, MN 55418  
{goldman,musliner}@htc.honeywell.com

Michael J. Pelican

Reliable Software Technologies  
21351 Ridgeway Circle, Suite 400  
Dulles VA 20166-6503

### Introduction

We are developing autonomous, flexible control systems for mission-critical applications such as Uninhabited Aerial Vehicles (UAVs) and deep space probes. These applications require hybrid real-time control systems, capable of effectively managing both discrete and continuous controllable parameters to maintain system safety and achieve system goals. Using the CIRCA architecture for adaptive real-time control systems (Musliner, Durfee, & Shin 1993; 1995; Musliner *et al.* 1999), these controllers are synthesized *automatically* and dynamically, on-line, *while the platform is operating*. Unlike many other AI planning systems, CIRCA's automatically-generated control plans have strong temporal semantics and provide safety guarantees, ensuring that the controlled system will avoid all forms of mission-critical failure.

This paper is intended to convey an intuitive understanding of the way CIRCA uses model-theoretic methods, particularly model-checking, in its planning. Our work differs from other work in this area in at least four important ways:

**Type of model** In order to provide temporal guarantees, we use as our models *timed automata* (Alur 1998), rather than the simpler finite automata. Note that these automata do *not* have a finite state-space, and that they are not Markovian in the same way that most planners' action representations are.

**Use of model-checking** We do not reformulate the problem of planning as a model-checking problem, and then use a model-checker to generate a plan. Instead, we use the model-checker as a test oracle in a generate-and-test style planning algorithm. However, we go beyond conventional uses of verification

to check programs, by using error traces to direct backtracking in the planning algorithm and by incrementally checking partially constructed controllers.

**Dynamic environment** We relax the conventional planning assumption of static environments.

**Type of plan** Instead of generating plans that are simply directed at goal achievement, we generate reactive controllers that attempt to move the agent to a goal, while maintaining the system in a satisfactory state. If the agent is already in a goal state, CIRCA will attempt to remain there. Accordingly, CIRCA controllers do not terminate, as conventional plans do.

In the next section, we provide a brief description of the planning process performed by CIRCA's State Space Planner (SSP). We will also describe the execution semantics of the reactive controllers CIRCA generates, and how we model it (and its dynamic environment) as a timed automaton. Then we discuss the problem of verifying the safety of a controller automaton, and present a brief overview of the model-checking approach. We show how the SSP automaton is converted into a verification problem for the model checking mechanisms, and discuss optimizations used to reduce the complexity of model checking.

### The CIRCA SSP

The CIRCA State Space Planner (SSP) automatically synthesizes timed discrete-event controllers (reactive plans) for hard real-time applications. The input to the SSP is a description of a control problem in the form of environment dynamics (including uncontrollable processes and threats to system safety), actions available to the controller, and goals to be realized. The SSP returns a controller that is guaranteed to maintain the safety of the controlled system, while it opportunistically tries to reach goal states.

---

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under contracts F30602-98-C-0212 and F30602-00-C-0017. Thanks to the anonymous referees for helpful suggestions.

SSP-synthesized controllers are compiled into sets of individual reactions (test-action pairs) that specify what action should be taken for each reachable system state, where system state is defined as a set of assignments of values to observable features. The controller provides safety guarantees by meeting the timing requirements of the control problem; these timing requirements are inferred from the model of the uncontrollable processes that threaten the system.

For example, Figure 1 contains the transition descriptions for a simple UAV control problem. CIRCA's transition descriptions are STRIPS-like, but permit nondeterminism and are augmented with timing information, which we will explain later. The transitions in Figure 1 describe a problem in which a UAV is attempting to follow a normal flight path (hence the `*goals*` statement). However, at any time during its flight, the UAV might be tracked by enemy radar. Some time after the initial tracking, a surface-to-air missile (SAM) may be launched. If no countermeasures are taken, that SAM may destroy the UAV after at least a certain minimum amount of time has passed (e.g., the minimum flight time of the missile). The UAV has available to it some evasive maneuvers that will cause the SAM to miss the UAV, if the UAV initiates its maneuvers quickly enough. Also, since the maneuvers divert the UAV from its nominal trajectory, the UAV should end its evasive behavior whenever possible.

Figure 2 shows the state space resulting from a simple timed controller design that will preserve the safety of the UAV. In the initial state, labeled "State 17" and shown as a shaded oval, the UAV is on its normal trajectory and has no indication of a radar-guided missile tracking it. This is a desirable state, so the controller will make no effort to leave it. However, at any time, a radar threat could occur, moving the system into state 16. The controller will react to this threat by taking evasive action, and maintaining the evasive maneuvers until the missile has been avoided (i.e., until the system has entered state 24). At this time the threat has been neutralized, and the system is free to return to its normal flight path. This controller was automatically generated by CIRCA, and the state diagram was generated from CIRCA data structures by the daVinci program (Fröhlich & Werner 1995).

There are several important aspects to note about this example state space model, or finite automaton. Note that the automaton contains loops: the UAV may be threatened by more than one missile, and will remain in (or re-start) evasive maneuvers as long as it is threatened.

Note also that time is not an explicit part of the state representation. Omitting time from the planner's state space is critical to the compact representation of loop-

```
(setf *goals* '((path normal)))

;; Radar-guided missile threats can occur
;; at any time.
(make-instance 'event
  :name "radar_threat"
  :preconds '((radar_missile_tracking F))
  :postconds '((radar_missile_tracking T)))

;; You die if don't defeat a threat by 1200
;; time units.
(make-instance 'temporal
  :name "radar_threat_kills_you"
  :preconds '((radar_missile_tracking T))
  :postconds '((failure T))
  :min-delay 1200)

;; It takes no more than 10 time units to start
;; evasives.
(make-instance 'action
  :name "begin_evasive"
  :preconds '((path normal))
  :postconds '((path evasive))
  :max-delay 10)

;; We defeat missile in between 250 and 400
;; time units.
(make-instance 'reliable-temporal
  :name "evade_radar_missile"
  :preconds '((radar_missile_tracking T)
              (path evasive))
  :postconds '((radar_missile_tracking F))
  :delay (make-range 250 400))

;; It takes no more than 10 time units to
;; end evasives.
(make-instance 'action
  :name "end_evasive"
  :preconds '((path evasive))
  :postconds '((path normal))
  :max-delay 10)
```

**Figure 1:** A simple domain description for a UAV threatened by radar-guided missiles.

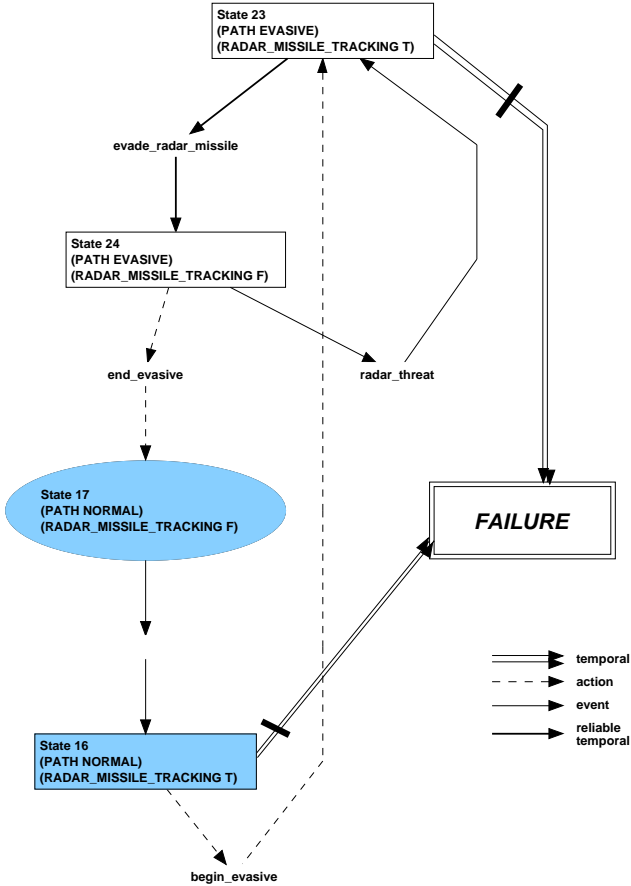


Figure 2: Simple UAV controller for evading radar-guided SAM threats.

ing plans; if we included time in the state representation, then persistent reactive control against an unpredictable or adversarial world would explode the state space. A second reason for omitting time has to do with the execution semantics of CIRCA plans. CIRCA controllers can be implemented as simple, memoryless reactive systems. Controllers that explicitly represented time and used it in their computations would be difficult, and might be impossible to implement in multi-processing hard real-time systems.

The SSP’s temporal model was carefully designed to support reasoning about system safety with only a minimal amount of temporal information, thus limiting the complexity of the automata model. We associate with each transition a set of bounds on the time ( $\Delta$ ) which the system must dwell in the transition’s source state before the transition could possibly occur. The model includes four different types of transitions:

**Temporal Transitions** — Drawn as double arrows, temporal transitions represent uncertain processes that may lead to change, but only after at least some minimum amount of time has passed ( $\Delta \geq \min\Delta$ ). The only temporal transitions in our simple UAV example lead to failure, and are not shown in Figure 2 because the safety-preserving controller design makes failure unreachable.

**Event Transitions** — Drawn as single arrows, event transitions represent instantaneous transitions that are out of our control, and may happen any time their preconditions are satisfied. They are essentially the same as temporal transitions with a  $\min\Delta$  of zero.

**Action Transitions** — Drawn as dashed arrows, action transitions represent processes that are guaranteed to occur before the system has dwelled a certain amount of time in the source state. That is, action transitions will definitely occur before  $\Delta$  reaches an upper bound  $\max\Delta$ .

**Reliable Temporal Transitions** — Drawn as bold single arrows, reliable temporal transitions represent processes that are guaranteed to occur, if given enough time. They have both lower and upper bounds on the dwell time the system must stay in the source state before the reliable temporal transition will occur ( $\min\Delta < \Delta < \max\Delta$ ).

Using this information, the SSP reasons about one key temporal relationship: *preemption*. A transition  $t$  is preempted iff some other transition  $u$  from the same state must definitely occur before  $t$  could possibly occur. In other words,  $t$  is preempted iff  $\max\Delta(u) < \min\Delta(t)$ . In our UAV example, the `radar_threat_kills_you` transition is preempted in state 16 by the action transition `begin_evasive`.

Preemption is the key temporal relationship in CIRCA models because it allows the SSP to build discrete event controllers that make certain parts of the potential system state space unreachable. By making all potential failure states unreachable, the SSP can build plans (controllers) that are guaranteed to keep the system safe, while also pursuing other less-critical goals. The goal of plan verification, discussed in the next section, is to prove that the preemptions CIRCA has planned will in fact hold true for all possible future world “trajectories” (i.e., paths through the reachable states).

Note that the `begin_evasive` action does not actually disable the `radar_threat_kills_you` transition: it simply begins the process of defeating the threat, which is represented by the reliable temporal transition `evade_radar_missile`. In Figure 2, we see that the `radar_threat_kills_you` TTF is actually preempted out of both state 16 and the subsequent state 23. This is called a *dependent temporal chain*, because the amount of time left to preempt the TTF in state 23 is not the original minimum dwell time (as it was in state 16), but the original  $\min\Delta$  minus however much time the system may have dwelled in state 16 before transitioning to state 23. Since CIRCA reasons about worst-case circumstances, we can assume that that dwell time, in the worst case, is equivalent to the upper bound dwell time ( $\max\Delta$ ) imposed by the planned action `begin_evasive`, so that the new  $\min\Delta$  in state 23 is actually  $1200 - 10 = 1190$ .

Thus the SSP’s model is actually non-Markovian: the temporal semantics of the TTF out of state 23 depend on the path the system takes to get there. Naturally, this complicates the process of reasoning about the temporal model, and motivates our use of model checking to verify the required TTF-preemption properties.

## Model Checking for Plan Verification

In order to verify that the CIRCA SSP’s plans are safe, we must project what will happen when they are executed. In particular, we must determine whether the actions we have planned do, in fact, preempt all possible transitions to failure. Recall that the SSP does not represent time in its state space; it reasons only about the feature space of the system. Instead, the SSP uses a separate module, employing techniques developed in the computer-aided verification research community to check the plans it generates. Specifically CIRCA uses techniques for verifying properties of *timed automata* (Alur 1998).

A naive algorithm for CIRCA plan verification is easy to propose: start at the initial state(s), find all the possible successor states, and repeat. If you ever

enter a failure state, the verification has failed.

The problem with this algorithm is hidden in the definition of system state. To determine the possible successor states, we must know how long transitions have been enabled. For example, to determine at state 23 whether `radar_threat_kills_you` happens before or after `evade_radar_missile`, we must know whether the former transition has been active<sup>1</sup> for 1200 time units before the latter has been active for 400 (see Figure 1).

Imagine that each transition has associated with it a timer, or “clock.” When the transition is enabled, that clock is reset to zero and started. When the transition is disabled, that clock is turned off. Whenever that clock goes over the lower bound on the corresponding transition, the transition may occur; the transition is guaranteed to occur before the upper bound on the transition (unless some other transition intervenes).

Thus we can characterize the full state of the controlled system by the full set of feature values and a vector of artificial clock values. For example:

```
flight_path = evasive
radar_missile_tracking = true
clock(evade_radar_missile) = 40
clock(radar_threat_kills_you) = 700
```

By comparing this state against the problem definition given in Figure 1, you may readily see that this state is safe. `radar_threat_kills_you` cannot take place for 500 more time units, by which time `evade_radar_missile` will have preempted it.

Unfortunately, the verification problem, as naively framed, is not practically solvable. Since the clocks are real-valued, the set of system states is uncountably large. However, the set of interesting values is less than infinite, since there are only a finite number of decisions that need be made. For example, all values of `clock(radar_threat_kills_you)` that are over 1200 are equivalent. However, the number of relevant states may still be very large.

**Timed Automata Representation** Fortunately, researchers in computer-aided verification have found ways to compactly represent states like this for a class of finite state machines called *timed automata* (Alur 1998). Timed automata differ in a few minor ways from SSP state machines, but SSP state machines can be translated into timed automata. Timed automata states comprise a *location* (corresponding to an SSP state, or feature vector) and a *clock-interpretation*, or *vector of clock values*. All of the clocks increment synchronously, but can be independently reset to zero by selected transitions. Transitions themselves are instan-

<sup>1</sup>i.e., its preconditions have been entailed by the state feature description.

taneous. Temporal constraints in timed automata take two forms: transition *guard expressions* that must be true to enable a transition, and *state invariant* expressions that must be true all the time the system remains in a particular state.

Mapping an SSP state space model into a timed automaton is a fairly simple matter of assigning different clocks to different CIRCA transitions and translating the CIRCA transition timing constraints into timed automaton clock constraints. Once this translation is complete, the timed automaton model can be passed to our model-checking code, the Real-Time Analysis (RTA) module, to determine whether failure is reachable and, if so, what path of transitions leads to failure (to guide CIRCA’s intelligent backjumping).<sup>2</sup>

Figure 3a illustrates the RTA timed automaton that corresponds to CIRCA’s solution to the UAV example of Figure 2. Briefly, the automatic translation process involves mapping each type of SSP state space transition, as follows:

**Temporal Transitions** — Temporal transitions require the system to dwell in a state for a certain amount of time before the transition may occur. This corresponds exactly to a transition guard expression in RTA. Thus temporal edges are each assigned a clock, and have guard expressions constraining the value of that clock to be greater than the temporal transition’s minimum delay. The clock is reset by all edges entering the source state of the temporal edge, if that edge does not come from a state in which the same temporal is enabled.

**Event Transitions** — Because event transitions can occur at any time, they have no associated clocks and are simply unrestricted edges in the RTA graph.

**Action Transitions** — Recall that action transitions place an upper bound on the time the system may dwell in the transition’s source state before it necessarily will move to the transition’s sink state. In our RTA model, this corresponds to an upper bound state invariant expression. Each instance of an action transition (action edge) is assigned a new clock. The clock is reset by all edges entering the source state of the action edge, if that edge does not come from a state in which the same action is enabled. The action edge itself has no guarding clock constraints; instead, the action edge’s upper bound is expressed as an invariant in the edge’s source state.

**Reliable Temporal Transitions** — Reliable temporals combine the lower-bound and upper-bound

<sup>2</sup>In the RTA module we have been experimenting with varying mixtures of our own custom code and timed automaton checkers developed elsewhere, notably the KRONOS system (Yovine 1998).

Let  $\Sigma$  be the set of reachable SSP states,  $\sigma_i$ . Let  $\mathcal{S}$  be a corresponding set of timed automaton locations,  $s_i$ . Let  $\mathcal{C}$  be a set of clocks,  $c_{t_i}$  one for each transition. Apply the following map  $f(\sigma) \rightarrow s$ :

- If  $\sigma$  has not yet been planned,  $s$  will be an accepting location of the automaton;
- Otherwise, construct a new location, and for every transition,  $t$  enabled in  $\sigma$ , add the following features to  $s$ :

**Invariants** If  $t$  has an upper bound (i.e., if  $t$  is a reliable temporal transition or an action), add an invariant to the state,  $c_t \leq \max \Delta t$ .

**Transitions out** Add a transition from  $s$  to the successor state,<sup>3</sup>  $t(s)$ . The *guard* of this transition is the condition  $c_t > \min \Delta t$ . On this transition, *reset* all clocks that correspond to uncontrollable transitions (temporal transitions and events) not enabled in  $t(s)$ . Also *reset* the clock corresponding to the action planned for  $s$ .

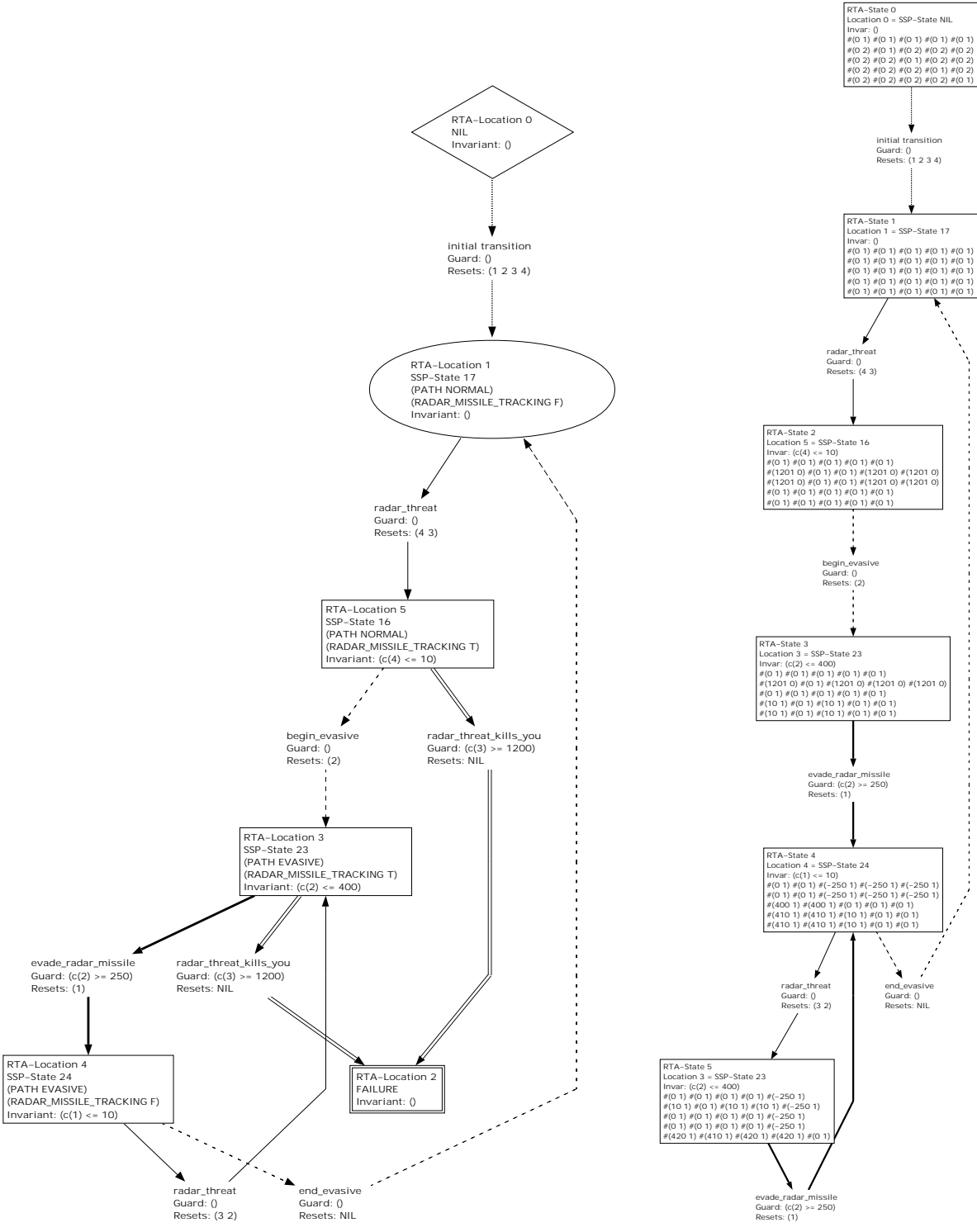
**Figure 4:** Summary of translation process.

timing constraints of temporals and actions, so their RTA mapping uses transition guards to represent the lower bounds and state invariants to represent the upper bounds.

We summarize the translation process in Figure 4.

**Efficient Model Checking** The critical concept for taming the complexity of timed automaton verification is an equivalence relation (“region equivalence”) between system states (Alur 1998). This equivalence relation makes use of the intuition that all values for a given clock are equivalent above a maximum value (the largest constant the clock is ever compared to). Furthermore, since we are only concerned with the reachability of various states, the actual values of different clocks in a state are not as important as their *relative* values. Because the clocks are all notionally incremented at the same rate, the relationships between the clock values upon entry to a state is sufficient to determine which outgoing transitions are possible: a clock that is behind another cannot catch up (within a state). Based on this equivalence relation, it can be shown that any timed automaton (SSP plan) has only a finite number of states.<sup>4</sup> Therefore, the problem of determining reachability (SSP plan verification) is decidable.

<sup>4</sup>More precisely, there are only a finite number of state equivalence classes, and state equivalence classes are sufficient to determine reachability.



(a) Input finite automata.

(b) Clock-zone expansion.

Figure 3: The input model and clock zone analysis for the example UAV domain.

A further optimization is possible, to make verification practical. The key intuition behind this optimization is that all reachability questions hinge on pairwise comparisons between clock values. In order to determine whether or not one transition can occur, we compare a single clock against a constant. To determine whether one transition occurs before another, we only need to determine which will reach its associated constant first. To answer this question, we only need to know the *difference* between pairs of clock values.

Therefore, we can compactly represent clock regions using a *difference-bound matrix* (Dill 1989) whose entries represent bounds on the difference between pairs of clocks and between single clocks and a dummy clock whose value is always zero. Difference-bound matrices have two advantages. First, they provide a compact representation for equivalence classes of clock-states in timed automata. Second, they also have a canonical form, derived using any standard all-pairs shortest-path algorithm. Putting the difference-bound matrices into canonical form makes it easy to determine when two automaton states are equivalent. Recognizing equivalent states is necessary in order for reachability search to terminate.

Figure 3b illustrates the reachability verification of the SSP plan given in Figure 1, optimized by the use of difference-bound matrices. Space limits preclude us from describing the difference bound notation in detail. However, a simple examination of Figure 3b shows one notable aspect of the RTA verification: there are two RTA states (3 and 5) that correspond to the SSP state 23. That is, the RTA algorithm has recognized the distinction between the two routes into SSP state 23 (see Figure 2) as being a temporally significant difference. The temporal transition to failure from state 23 will have different amounts of time left on its clock depending on whether we enter from state 16, where it was already enabled, or state 24, where it was not enabled (see Figure 3a, RTA-locations 5 and 4).

**Using Verifier Results to Control Search** If the verification system indicates that the controller is *not* correct, the SSP uses the verification system’s output to identify and repair the incorrectness. The verification system, when it detects an incorrectness does so in the form of a path (an execution trace) from an initial state of the system to a distinguished failure state. The SSP maps entries in the trace into entries in its search stack. Each search stack entry corresponds to the assignment of a control action a reachable state.<sup>5</sup> Thus for each location that appears in an execution trace, there is a corresponding decision in the SSP’s stack.

<sup>5</sup>This is a slight oversimplification, but sufficient to grasp the essentials.

This set of decisions, taken together, make up a *no-good*: a set of search decisions that, taken together, is inconsistent. When encountering a verification failure, the SSP uses a backjumping search method (Gaschnig 1979), to revise the most recent search decision that appears in the corresponding no-good. For this application, backjumping is much more efficient than chronological backtracking. Indeed, for all but the simplest examples, chronological backtracking is simply infeasible, taking days of compute time.

Note that the ability to employ the execution trace is a substantial advantage of our approach. The execution traces provided by verification systems are *not* minimal in any sense of the word. This means that designers using a verification tool to evaluate a design must locate errors by extensive inspection of a program trace leading to failure. CIRCA automates this process.

## Related Work

The CIRCA SSP is a reactive planner, and thus has much in common with work on reactive systems in AI and control theory. CIRCA is unusual in two ways: it automatically synthesizes, or plans, its reactions, and it provides performance guarantees through the methods of hard real time.

In independently-developed work, Asarin, Maler, Pnueli and Sifakis (Asarin, Maler, & Pnueli 1995; Maler, Pnueli, & Sifakis 1995) developed a game-theoretic method of synthesizing real-time controllers. They view the problem as trying to “force a win” against the environment, by guaranteeing that all traces of system execution will pass through (avoid) a set of desirable (undesirable) states. Their method is very similar to ours, but their work stopped at the development of the algorithm and derivation of complexity bounds; it was never implemented.

Kabanza, et. al. have developed work very similar to ours in intention. Their early work (fully presented in (Kabanza, Barbeau, & St.-Denis 1997)) is similar to the original CIRCA State Space Planner work, but does not take into account metric temporal information. Later work (Kabanza 1996), extends the original framework by incorporating metric time, but does so by effectively imposing a system-wide clock and progressing the controller one “tick” at a time. In control problems with widely varying time constants, this approach will lead to an explosion of states; we have adopted model-checking techniques that minimize this state explosion.

Markov Decision Processes provide a theoretical basis for planning and action that is similar to discrete control theory, but they place the accent on uncertainty. The SSP techniques discussed in this paper do

not attempt to reason about quantified measures of uncertainty, they make the worst-case assumption: “anything bad that can happen will happen.” However, there has been some preliminary work on developing a probability model for CIRCA, to permit principled model-pruning decisions (Li *et al.* 1999).

### Current and future work

At the time of this writing, we are working on synthesizing mission-specific controllers for UCAVs. A typical scenario for UCAV controller synthesis involves a twelve state discrete controller, that reacts to threats from radar-guided missiles (by taking appropriate evasive action and deploying chaff), while attempting to follow a desired trajectory (returning to it when evasive maneuvers are done). This is a more extensive version of the example discussed in this paper. Generating this model requires approximately 18 seconds on a high-end desktop machine (dual Pentium 700); backtracking 43 times and invoking the verifier 24 times.<sup>6</sup> The largest model-checker query for this scenario explores approximately 6000 states using our own timed automaton verifier.

We have in the past experimented with a number of other domains, including controlling a simulated robot arm attached to a conveyer belt, synthesizing a controller for the railroad-crossing reference problem for hybrid control and the Cassini orbital insertion reference problem (Musliner & Goldman 1997). Some of these examples are substantially larger than the UAV problems.

Our approach to synthesizing discrete controllers using model checking is innovative but costly. Even with the optimizations discussed above and others, invoking model checkers on large problems can still lead to very large state spaces. We are currently developing several new techniques to improve performance by taking advantage of the unique aspects of the CIRCA SSP problem. We are also using model checking to verify not just the SSP plan, but also how it is implemented in reactive execution rules by the CIRCA executive.

### References

Alur, R. 1998. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*.

Asarin, E.; Maler, O.; and Pnueli, A. 1995. Symbolic controller synthesis for discrete and timed systems. In Antsaklis, P.; Kohn, W.; Nerode, A.; and Sastry, S., eds., *Proceedings of Hybrid Systems II*. Springer Verlag.

<sup>6</sup>Some plan defects can be detected without invoking the model-checker.

Dill, D. L. 1989. Timing assumptions and verification of finite-state concurrent systems. In Sifakis, J., ed., *Automatic Verification Methods for Finite State Systems*. Berlin: Springer Verlag. 197–212. Proceedings of the International Workshop.

Fröhlich, M., and Werner, M. 1995. Demonstration of the interactive graph visualization system *davinci*. In Tamassia, R., and Tollis, I., eds., *Proceedings of DIMACS Workshop on Graph Drawing '94*. Springer Verlag.

Gaschnig, J. 1979. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University.

Kabanza, F.; Barbeau, M.; and St.-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Kabanza, F. 1996. On the synthesis of situation control rules under exogenous events. Appeared in the working notes of the 1996 AAAI Workshop on *Theories of Action, Planning, and Robot Control: Bridging the Gap*.

Li, H.; Atkins, E.; Durfee, E.; and Shin, K. 1999. Resource allocation for a limited real-time agent using a temporal probabilistic world model. forthcoming.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In Mayr, E. W., and Puech, C., eds., *STACS 95: Theoretical Aspects of Computer Science*. Springer Verlag. 229–242.

Musliner, D. J., and Goldman, R. P. 1997. CIRCA and the Cassini Saturn orbit insertion: Solving a prepositioning problem. In *Working Notes of the NASA Workshop on Planning and Scheduling for Space*.

Musliner, D. J.; Goldman, R. P.; Pelican, M. J.; and Krebsbach, K. D. 1999. SA-CIRCA: Self-adaptive software for hard real time environments. *IEEE Intelligent Systems* 14(4):23–29.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1561–1574.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Yovine, S. 1998. Model-checking timed automata. In Rozenberg, G., and Vaandrager, F., eds., *Embedded Systems*. Springer Verlag.