Verifying Real-Time Plans with Rigorous Execution Semantics

IJCAI Paper ID Number : 715

Content areas: reactive control, planning, real-time systems, validation and verification.

Abstract

We are developing autonomous control systems for mission-critical domains that require hard real-time performance guarantees. To automatically build reactive plans that meet these requirements, we use formal verification techniques to verify the quality of plans as they are built. The verification process uses precise *timed automaton* models of the plan executive that will run the resulting reactive plan. This reflexive modeling allows our system to formally verify not just that its plans are correct, but that they will be executed correctly.

Introduction

In the last few years, formal verification techniques have received increasing attention in the AI and applications communities, as formal methods scale up more effectively and Moore's Law makes them practical for realistic problems. Applying model checking and other verification methods to planning problems lends additional support to claims of plan correctness.

However, to ensure that automatically-generated plans are realistic, we must have a strong model of the executive that will run them. Too often, planning systems assume unrealistic execution models that do not account for uncontrollable concurrent events and the time required to sense world states and initiate actions. In our work on developing the CIRCA architecture for mission-critical real-time autonomous systems (Musliner, Durfee, & Shin 1993; 1995), we actually include an accurate executive model in our planning and verification process, so that CIRCA's plans are not just guaranteed to be correct according to an abstract world model, but to be correctly executable by an implemented real-time executive.

CIRCA uses concurrently-operating planning and plan-execution subsystems: the Controller Synthesis Module (CSM) reasons about models of the world and its own goals to build time-constrained safety-preserving plans (controllers), while a separate Real-Time Subsystem (RTS) reactively executes the automatically-generated plans and enforces guaranteed response times. In this paper, we describe how CIRCA incorporates formal verification within its planning process, using increasingly accurate models of its executive to verify that its plans will be both logically correct and correctly executable.

The Controller Synthesis Module

CIRCA's CSM automatically synthesizes real-time reactive controllers that guarantee system safety when run on CIRCA's real-time subsystem. The CSM takes in a description of the processes in the system's environment, represented as a set of time-constrained transitions that modify world features. These transition descriptions are similar to STRIPS operators with the addition of timing information and nondeterministic outcomes. For example, Figure 1 shows several transitions taken from a problem where CIRCA is to control the Cassini spacecraft in Saturn Orbital Insertion.[1]

The CSM reasons about transitions of three types:

Action transitions represent actions performed by the RTS. These parallel the operators of a conventional planning system. Associated with each action is a worst case execution time, an *upper bound* on the delay ($\Delta(a) \leq t$) before the action occurs.

Temporal transitions represent uncontrollable processes, some of which may need to be preempted. Associated with each temporal transition is a *lower bound* on its delay ($\Delta(\text{tt}) \geq t$). Transitions that have a delay lower bound of zero are referred to as "events," and are handled specially for efficieny reasons.

Reliable temporal transitions represent continuous processes that may need to be employed by the CIRCA agent. For example, when CIRCA turns on an IRU it initiates the process of warming up that equipment; the process will complete after some delay. Reliable temporal transitions have both upper and lower bounds on their delays.

[1] This example is adapted from Erann Gat's "From the Trenches" (Gat 1996).

```
;; Turning on an Inertial Reference Unit (IRU)
ACTION start_IRU1_warm_up
   PRECONDITIONS: '((IRU1 off))
   POSTCONDITIONS: '((IRU1 warming))
   DELAY: <= 1

;; the process of the IRU warming
RELIABLE-TEMPORAL warm_up_IRU1
   PRECONDITIONS: '((IRU1 warming))
   POSTCONDITIONS: '((IRU1 on))
   DELAY: [45 90]

;;sometimes the IRUs break without warning
EVENT IRU1_fails
   PRECONDITIONS: '((IRU1 on))
   POSTCONDITIONS: '((IRU1 broken))

;; if the engine is burning while the active
;; IRU breaks, we must quickly fix problem before
;; the spacecraft gets too far out of control
TEMPORAL fail_if_burn_with_broken_IRU1
   PRECONDITIONS: '((engine on)(active_IRU IRU1)
                    (IRU1 broken))
   POSTCONDITIONS: '((failure T))
   DELAY: >= 5
```

Figure 1: Example transition descriptions given to CIRCA's planner.

CSM Algorithm

Given problem representations as above, the controller synthesis (planning) problem can be posed as *choosing a control action for each reachable state (feature-value assignment) of the system.* This problem is not as simple as it sounds, because the set of reachable states is not a given — by the choice of control actions, the CSM can render some states (un)reachable.

Indeed, since the CSM focuses on generating *safe* controllers, a critical issue is making failure states unreachable. In controller synthesis, this is done by the process we refer to as *preemption*. A transition t is preempted in a state s iff some other transition t' from s must occur before t could possibly occur. The CSM achieves preemption by choosing a control action that is fast enough that it is guaranteed to occur before the transition to be preempted.

At the highest level of abstraction, the controller synthesis algorithm is as follows:

1. Choose a state from the set of reachable states (at the start of controller synthesis, only the initial state(s) is(are) reachable).
2. For each uncontrollable transition enabled in this state, choose whether or not to preempt it (any transition that leads to a failure state *must* be preempted).
3. Choose a control action or `no-op` for that state.

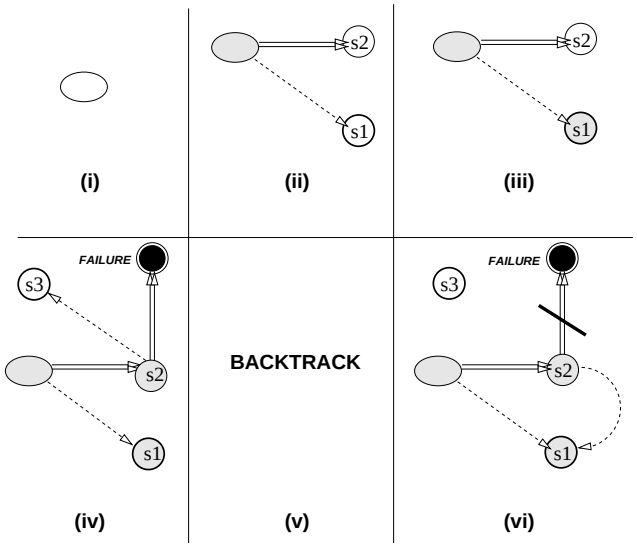

Figure 2: A simple example of controller synthesis.

4. Invoke the verifier to confirm that the (partial) controller is safe.
5. If the controller is *not* safe, use information from the verifier to direct backtracking.
6. If the controller *is* safe, recompute the set of reachable states.
7. If there are no unplanned reachable states (reachable states for which a control action has not been chosen), terminate successfully.
8. If some unplanned reachable states remain, loop to step 1.

Figure 2 provides a simple example of the process of controller synthesis. Initially (i), there is only one state reachable, the initial (oval) state. In (ii), the CSM has chosen a control action (dashed line) for the initial state (planned states are shaded gray), that will carry the system to a goal state, *s1* (goal states are indicated by bold outlines). There is also a temporal transition (double line) that may carry the system to *s2*. In (iii), we see the CSM decide to assign `no-op` as the control action for *s1*. This is permissible because *s1* is a safe state (there are no transitions to failure from that state), and is desirable because *s1* is a goal state. In (iv), the CSM attempts to complete the controller synthesis process by assigning an action to *s2* that will carry the system to *s3*. However, this action does *not* preempt the transition from *s2* to the failure state (black). This triggers a backtrack (v), and the system chooses an alternative action for *s2* (vi) that will carry the system to *s1*. This alternative action *does* preempt the transition to the failure state, so the controller is safe. All reachable states have been planned for, so the controller synthesis process has terminated successfully.

During the course of the controller synthesis run above, the CSM will have employed the verifier module after each assignment of a control action (i.e., after ii, iii, iv and vi). However, at stages ii, iii and iv, the controller is not complete. At such points we use the verifier as a conservative heuristic by treating all unplanned states (e.g., *s2* in iii) as if they are "safe havens." Unplanned states are treated as absorbing states of the system, and any verification traces that enter these states are regarded as successful. When the verifier indicates that a CSM-generated controller is *unsafe*, the CSM will query it for a path to the distinguished failure state. The set of states along that path provides a set of candidate decisions to revise.

Modeling for Verification

The temporal model underlying the CSM plan graphs is deceptively complex because it is non-Markovian; we do not include time in the state description. This has the advantage that the looping nature of reactively-executed plans is captured as loops in the plan graph topology. However, a path-dependent computation is required to determine how much time remains on a transition's delay when it applies to two or more connected states (e.g., when IRU1 has failed and the system progresses through several transitions while `fail_if_burn_with_broken_IRU1` continues to apply). To complicate matters further, we cannot assume that the planned actions are completely independent: they will be executed by a real executive with limited abilities to sense and react to the world, so the planned actions will compete for this bounded reactivity.

To efficiently reason about the timing in this world model *and account for the executive's bounded reactivity*, the CSM relies on an automatic verification system. The verifier ensures that the controllers that the CSM builds are safe. When making action decisions, the CSM uses very simple reasoning, non-path-dependent to make "guesses" about transition preemptions (the only really important temporal issue in these plans). Then each of these guesses is formally verified using a faithful model of the RTS.

Execution Semantics

The controllers of the CIRCA RTS are not arbitrary pieces of software; they are intentionally very limited in their computational power.[2] The controller generated by the CSM is compiled into a set of *Test-Action Pairs* (TAPs) to be run by the RTS. Each TAP has a boolean test expression that distinguishes between states where a particular action is and is not to be executed. The test expression is a function of the plan as a whole, because the same action may be assigned to more than one state. A sample TAP for the Saturn Orbit Insertion domain is given in Figure 3.

```
#<TAP 2>
 Tests: (AND (IRU1 BROKEN)
             (OR (AND (ACTIVE_IRU NONE) (IRU2 ON))
                 (AND (ACTIVE_IRU IRU1) (ENGINE ON))))
 Acts : select_IRU2
```

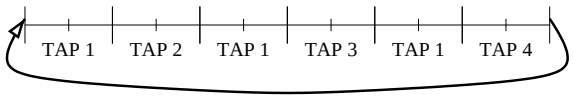

Figure 3: A sample Test-Action Pair and TAP schedule loop from the Saturn Orbit Insertion problem.

The set of TAPs that make up a controller are assembled into a loop and scheduled to meet all the TAP deadlines. The deadlines are computed from the delays of the transitions that the control actions must preempt.[3] If scheduling does not succeed, the CSM will backtrack to revise the controller, generating and scheduling a new set of TAPs.

Timed Automata

Now that we have a sense of the execution semantics of CIRCA's RTS, we briefly review the modeling formalism, timed automata, before presenting the model itself. A timed automaton is a finite automaton augmented with timing information.

Definition 1 (Timed Automaton) *A timed automaton A is a tuple $\langle \mathcal{S}, s^i, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I} \rangle$ where*

1. *$\mathcal{S}$ is a finite set of locations;*
2. *s^i is the initial state;*
3. *$\mathcal{X}$ is a finite set of clocks;*
4. *$\mathcal{L}$ is a finite set of labels;*
5. *$\mathcal{E}$ is a finite set of edges; and*
6. *$\mathcal{I}$ is the set of invariants.*

Each edge $e \in \mathcal{E}$ is a tuple (s, L, ψ, ρ, s') where $s \in \mathcal{S}$ is the source, $s' \in \mathcal{S}$ is the target, $L \subseteq \mathcal{L}$ are the labels, $\psi \in \Psi_{\mathcal{X}}$ is the *guard*, and $\rho : \mathcal{X} \rightarrow \mathcal{X} \cup \{0\}$ is a clock assignment (Daws *et al.* 1996).

Timing constraints appear in guards, invariants and clock assignments. In our modeling, all clock constraints are of the form $c_i \leq k$ or $c_i > k$ for some clock c_i and integer constant k. Informally, guards dictate when the model *may* follow an edge, invariants indicate

[2] These limitations serve to make controller synthesis computationally efficient and make it simpler to build an RTS providing timing guarantees.

[3] The tests and actions that the RTS can execute as part of its TAPs have associated worst-case execution times that are used to create and verify the TAP schedule.

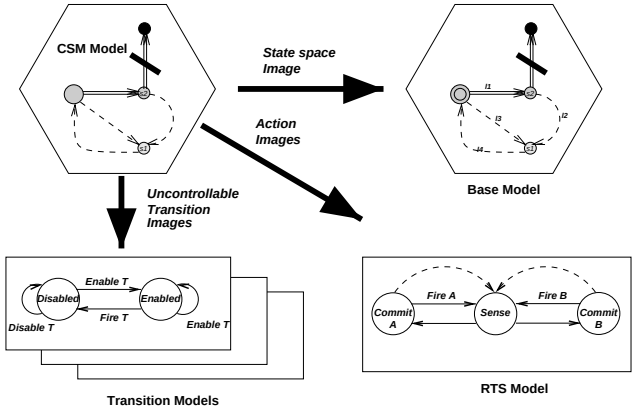

Figure 4: A pictorial summarization of the verifier model and its relation to the CSM model.

when the model *must* leave a state, and clock assignments are used to start and reset processes.

It often simplifies the representation of a complex system to treat it as a product of some number of simpler automata. The labels $\mathcal{L}$ are used to synchronize edges in different automata when creating their product.

Definition 2 (Product Automaton) *Given two automata A_1 and A_2, $A_1 = \langle \mathcal{S}_1, s_1^i, \mathcal{X}_1, \mathcal{L}_1, \mathcal{E}_1, \mathcal{I}_1 \rangle$ and $A_2 = \langle \mathcal{S}_2, s_2^i, \mathcal{X}_2, \mathcal{L}_2, \mathcal{E}_2, \mathcal{I}_2 \rangle$, their product A_p is $\langle \mathcal{S}_1 \times \mathcal{S}_2, s_p^i, \mathcal{X}_1 \cup \mathcal{X}_2, \mathcal{L}_1 \cup \mathcal{L}_2, \mathcal{E}_p, \mathcal{I}_p \rangle$, where $s_p^i = (s_1^i, s_2^i)$ and $\mathcal{I}(s_1, s_2) = \mathcal{I}(s_1) \wedge \mathcal{I}(s_2)$. The edges are defined by:*

1. *for $l \in \mathcal{L}_1 \cap \mathcal{L}_2$, for every $\langle s_1, l, \psi_1, \rho_1, s_1' \rangle \in \mathcal{E}_1$, and $\langle s_2, l, \psi_2, \rho_2, s_2' \rangle \in \mathcal{E}_2$, $\mathcal{E}_p$ contains $\langle (s_1, s_2), a, \psi_1 \cup \psi_2, \rho_1 \cup \rho_2, (s_1', s_2') \rangle$.*
2. *for $l \in \mathcal{L}_1 \setminus \mathcal{L}_2$, for $\langle s_1, l, \psi_1, \rho_1, s_1' \rangle \in \mathcal{E}_1$ and $s_2 \in \mathcal{S}_2$, $\mathcal{E}_p$ contains $\langle (s_1, s_2), l, \psi_1, \rho_1, (s_1', s_2) \rangle$. Likewise for $l \in \mathcal{L}_2 \setminus \mathcal{L}_1$.*

Modeling CIRCA with Timed Automata

CIRCA translates the CSM model into a set of interacting timed automata for a timed automaton verifier (see Figure 4). The use of multiple automata permits us to accurately and elegantly capture the interaction of multiple, simultaneously operating processes. The starting point of the translation is the CIRCA plan-graph, constructed by the CIRCA Controller Synthesis Module:

Definition 3 (CIRCA Plan Graph) $\mathcal{P} = \langle S, E, \vec{F}, \vec{V}, \phi, I, T, \iota, \eta, p, \pi \rangle$ *where*

1. *S is a set of states.*
2. *E is a set of edges.*
3. *$\vec{F} = [f_0 ... f_m]$ is a vector of features (in a purely propositional domain, these will be propositions).*
4. *$\vec{V} = [\mathcal{V}_0 ... \mathcal{V}_m]$ is a corresponding vector of sets of values ($\mathcal{V}_i = \{v_{i0} ... v_{ik_i}\}$) that each feature can take on.*
5. *$\phi : S \mapsto \vec{V}$ is a function mapping from states to unique vectors of value assignments.*
6. *$I \subset S$ is a distinguished subset of initial states; the world is assumed to begin in one of I.*
7. *$T = U \cup A$ is the set of transitions, made up of an uncontrollable (U) subset, the temporals and reliable temporals, and a controllable (A) subset, the actions.*
8. *ι is an interpretation of the edges: $\iota : E \mapsto T$.*
9. *$\eta : S \mapsto 2^T$ is the* enabled *relationship — the set of transitions enabled in a particular state. Without loss of precision, we may refer to the enabling relation as having as its domain either a set of states, or a set of feature-value assignments (i.e., a vector of values).*
10. *$p : S \mapsto A \cup \epsilon$ (where ϵ is the "action" of doing nothing) is the actions that the CSM has planned. Note that p will generally be a partial function, both because we may be verifying a plan that is only partially constructed, and because even in a fully-constructed plan there may be unreachable states.*
11. *$\pi : S \mapsto 2^U$ is a set of preemptions the CSM expects. For any $s \in S$, let the set of uncontrollable transitions, $u(s)$, that label edges out of s be $\{u \in U \mid (\exists e \in E \mid e = s \rightarrow s' \wedge \iota(e) = u)\}$. Then $\eta(s) = u(s) \cup \pi(s)$ and $\pi(s) \cap u(s) = \emptyset$.*
    That is, every uncontrollable transition enabled in a state will either be a member of the set of preemptions for that state, or (exclusively) will label one or more edges in the plan graph.

For every CIRCA plan graph, $\mathcal{P}$, we construct a timed automaton model, $\theta(\mathcal{P})$. $\theta(\mathcal{P})$ is the product of a number of individual automata. There is one automaton, which we call the *base model*, that models the feature structure of the domain. There is an *RTS model* that models the actions of the CIRCA agent. Finally, for every uncontrollable transition, there is a separate timed automaton modeling that process. The interaction between the base machine and the other two automata is regulated by the labels on the transitions of the various machines. Proper synchronization ensures that the base machine state reflects the effect of the transitions and that the state of the other automata accurately indicate whether or not a given process will (may) be underway.

Definition 4 (Translation of CIRCA Plan Graph) $\theta(\mathcal{P}) = \beta(\mathcal{P}) \times \rho(\mathcal{P}) \times \prod_{u \in U(\mathcal{P})} \upsilon(u)$ *where $\beta(\mathcal{P})$ is the base model; $\rho(\mathcal{P})$ is the rts model; and $\upsilon(u)$ is the automaton modeling the process that corresponds to uncontrollable transition u.*

Definition 5 (Base model)
$\beta(\mathcal{P}) = \langle \theta(S), \{l^0\}, \Sigma(\mathcal{P}), \emptyset, I_\top, \theta_E(\mathcal{P}) \rangle$ *where:*

1. $\theta(S) = \{s \in S \mid \theta(s)\} \cup \{l^{\mathcal{F}}, l^0\}$ *is the image under* θ *of the state set of* $\mathcal{P}$. *This image contains a location for each location in* $\mathcal{P}$, *as well as a distinguished failure location,* $l^{\mathcal{F}}$, *and initial location,* l^0.
2. $\Sigma(\mathcal{P})$ *is the label set; it is given as Definition 6.*
3. $\theta_E(\mathcal{P})$ *is the edge set of the base model. It is given as Definition 7.*

Note that there are no clocks in the base machine; all timing constraints will be handled by other automata in the composite model. Thus, the invariant for each state in this model is simply $\top$. We have notated this vacuous invariant as $I_\top$. Similarly, all of the edges have a vacuous guard. The labels of the translation model ensure that the other component automata exert the appropriate control.

Definition 6 (Label set for $\theta(\mathcal{P})$)

$$\begin{aligned} \Sigma(\mathcal{P}) \quad = \quad & \{\forall u \in U \mid \mathbf{e}_u, \mathbf{d}_u, \mathbf{f}_u\} \cup & (1)\\ & \{\forall a \in A \mid \mathbf{f}_(a)\} \cup & (2)\\ & \{\mathbf{r}_a\} & (3) \end{aligned}$$

The symbols in (1) are used to synchronize the automata for uncontrollable transitions with the base model. The symbols in (2) together with the distinguished reset symbol $\mathbf{r}_a$ are used to synchronize the automaton modeling the RTS with the Base Model. The Base Model edge set, $\theta_E(\mathcal{P})$, captures the *effect* on the agent and environment of the various transitions.

Definition 7 (Base Model edge set) $\theta_E(\mathcal{P})$ *is made up of the following subsets of edges:*[4]

(1) $\{s \in I \mid \langle l^0, \theta_\sigma(\text{init}, s), \theta(s) \rangle\}$
(2) $\{s \in S, \forall u \in \pi(s) \mid \langle \theta(s), \{\mathbf{f}_u\}, l^{\mathcal{F}} \rangle\}$
(3) $\{s \in S, u \notin \pi(s), s' \in u(s) \mid \langle \theta(s), \{\mathbf{f}_u\} \cup \theta_\sigma(u, s'), \theta(s') \rangle\}$
(4) $\{s \in S, a = p(s) \mid \langle \theta(s), \{\mathbf{c}_a\}, s \rangle\}$
(5) $\{s \in S, a \in \eta(s), s' \in a(s) \mid \langle \theta(s), \{\mathbf{f}_a\} \cup \theta_\sigma(a, s'), \theta(a(s)) \rangle\}$
(6) $\{s \in S, a \notin \eta(s) \mid \langle \theta(s), \{\mathbf{f}_a\}, l^{\mathcal{F}} \rangle\}$

Edge set (1) is merely a set of initialization edges, that carry the base model from its distinguished single initial location to the image of each of the initial states of $\mathcal{P}$. (2) takes the base model to its distinguished failure location, $l^{\mathcal{F}}$, whenever a transition that the CSM had preempted actually occurs (i.e., when a preemption fails). (3) captures the effects of the uncontrollable transitions the CSM expected to happen (didn't preempt). (4) synchronizes with the RTS transitions that capture the RTS committing to execute a particular action (i.e., the test part of the TAP). (5) captures the effects of a successfully-executed action. (6) captures a failure due to a race condition. We discuss this class of failure at the end of the section.

In Definition 7, event sets $\theta_\sigma(t, s)$ are used to capture the effects on the various processes of going to s by means of t.

Definition 8 ($\theta_\sigma(t, s)$)

$$\begin{aligned} \theta_\sigma(s) \quad = \quad & \{\mathbf{e}_u \mid u \in \eta(s)\} \cup \\ & \{\mathbf{d}_u \mid u \neq t \wedge u \notin \eta(s)\} \cup \{\mathbf{r}_a\} \end{aligned}$$

The symbol set $\theta_\sigma(t, s)$ contains an enable symbol for each u enabled in s, and a disable symbol for each u not enabled in s. The addition of the symbol $\mathbf{r}_a$ ensures that the RTS machine will stay in synchronization; i.e., that it will "notice" the state transition.

There will be one automaton, $\upsilon(u)$ for every uncontrollable transition, u. Each such model will have two states, enabled, e_u, and disabled, d_u, and transitions for enabling, disabling, and firing: $\mathbf{e}_u, \mathbf{d}_u$, and $\mathbf{f}_u$, respectively (see Figure 4). It will also have a clock, c_u, and the guards and invariants will be derived from the timing constraints on u:

Definition 9 (Uncontrollable Transition Automata)

$$\upsilon(u) = \langle \{e_u, d_u\}, d_u, \{c_u\}, \{\mathbf{e}_u, \mathbf{d}_u, \mathbf{f}_u\}, E, I \rangle$$

Where the edgeset, E, is defined as follows:

$$\begin{aligned} & \langle d_u, \{\mathbf{d}_u\}, \top, \emptyset, e_u \rangle \\ & \langle d_u, \{\mathbf{e}_u\}, \top, c_u := 0, e_u \rangle \\ & \langle e_u, \{\mathbf{e}_u\}, \top, \emptyset, e_u \rangle \\ & \langle e_u, \{\mathbf{d}_u\}, \top, \emptyset, d_u \rangle \\ & \langle e_u, \{\mathbf{f}_u\}, c_u \geq lb(\Delta_u), \emptyset, d_u \rangle \end{aligned}$$

and the invariants are defined as follows:

$$I(e_u) = c_u \leq ub(\Delta_u), I(d_u) = \top$$

The model for the RTS, ρ, contains all of the planned actions in a single automaton. The execution of each planned action is captured as a two stage process: the first is the process of committing to the action (going to the state c_a), and the second is the action's execution (returning to s_0 through transition $\mathbf{f}_a$).

Definition 10 (RTS Model)

$$\begin{aligned} \rho \quad = \quad & \langle \{s_0\} \cup \{\forall a \in p, c_a\}, s_0, \{c_{RTS}\}, \{\mathbf{r}_a\} \cup \{\mathbf{c}_a, \mathbf{f}_u\}, \\ & \{a \in p \mid \langle s_0, \{\mathbf{c}_a\}, c_{RTS} \leq 0, c_{RTS} := 0, c_a \rangle, \\ & \quad \langle c_a, \{\mathbf{f}_a\}, c_{RTS} \geq ub(\Delta_a), c_{RTS} := 0, s_0 \rangle, \\ & \quad \langle c_a, \{\mathbf{r}_a\}, c_{RTS} < ub(\Delta_a), c_{RTS} := 0, s_0 \rangle\} \\ & \{I(c_a) = c_{RTS} \leq ub(\Delta_a), I(s_0) = c_{RTS} \leq 0\} \rangle \end{aligned}$$

[4] The clock resets of these transitions are all $\emptyset$, and the guards are all $\top$, so we have omitted them.

There are two classes of safety violations that we look to the verifier to detect. The obvious one is a failure to successfully preempt some nonvolitional transition. This case is caught by transitions (2) of Definition 7. The less obvious one is a race condition: here the failure is to plan a for state s but not complete it before an uncontrolled process brings the world to another state, s', that does not satisfy the preconditions of a. The latter case is caught by transitions (6) of Definition 7.

Final Verification of Complete Controller

The above model correctly represents the interaction between the environment and the RTS. However, it is necessarily only heuristic in nature, until we have a complete plan. The reason is that until we know the entire plan, we cannot know the shape of the TAP loop (see Figure 3), and hence cannot know the true delays before actions occur. Currently, we take the admissible, but overoptimistic, step of assuming the agent will immediately choose to take the action for its current state (see (4) of Definition 7).

Hence, once the controller synthesis process has completed, and the TAP loop has been generated, we run the verification algorithm one final time. At this time, we know the exact shape of the TAP loop, and hence the exact execution semantics of the plan. We do not have space here to outline the TAP loop translation process, but it is a straightforward adaptation of the model described above to loops like those in Figure 3. Details will be given in the full paper.

Related Work

The "planning as model checking" approach uses the verification system as a prover to solve the problem itself (Giunchiglia & Traverso 1999), rather than as a component of a synthesis program, as in CIRCA. This work also has a much simpler model of execution, ignoring the duration of actions, the possibility of external, uncontrolled transitions, and the question of how the controller is to be implemented.

Asarin *et al.* (1995) developed a similar, game-theoretic method of synthesizing real-time controllers that do account for time and uncontrollable events. This work stopped at the design of the algorithm and derivation of complexity bounds; to our knowledge it was not implemented. This approach has been implemented for the special case of automatically synthesizing schedulers (Altisen *et al.* 1999).

Kabanza (1996) has developed work very similar to ours in scope and intention. He incorporates time 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.

Conclusions

The CIRCA CSM is a novel application of automatic verification systems to automatic synthesis of controllers (planning). Previous attempts to use automatic verifiers in planning have limited themselves to simpler execution semantics and/or simply assumed that the plans could be implemented correctly. Our system has a rich model of the execution of its timed controller, that reflects the behavior of a hard real-time executive.

The model described above cannot be implemented in a naive way, as this leads to a state space explosion. Our implementation uses a more efficient algorithm that exploits features of the CIRCA domain model and lazily builds the product automaton, producing speedups of two orders of magnitude on our test problems.[5] Details of the algorithm will be presented in forthcoming publications.

References

Altisen, K.; Goessler, G.; Pnueli, A.; Sifakis, J.; S.Tripakis; and S.Yovine. 1999. A framework for scheduler synthesis. In *Proceedings of the 1999 IEEE Real-Time Systems Symposium (RTSS '99).* Phoenix, AZ: IEEE Computer Society Press.

Asarin, E.; Maler, O.; and Pneuli, 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.

Daws, C.; Olivero, A.; Tripakis, S.; and Yovine, S. 1996. The tool Kronos. In *Hybrid Systems III.*

Gat, E. 1996. News from the trenches: An overview of unmanned spacecraft for AI. In Nourbakhsh, I., ed., *AAAI Technical Report SSS-96-04: Planning with Incomplete Information for Robot Problems.* American Association for Artificial Intelligence.

Giunchiglia, F., and Traverso, P. 1999. Planning as model-checking. In *Proceedings of ECP-99.* Springer Verlag.

Kabanza, F. 1996. On the synthesis of situation control rules under exogenous events. In Baral, C., ed., *Theories of Action, Planning, and Robot Control: Bridging the Gap,* number WS-96-07, 86–94. AAAI Press.

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.

[5] Problem definitions will be available at *http://anonymous.url.*