Appears in Proc. 2002 Hybrid Systems: Computation and Control Workshop
March, 2002

Exploiting Implicit Representations in Timed Automaton Verification for Controller Synthesis

Robert P. Goldman, David J. Musliner, Michael J. S. Pelican*

Automated Reasoning Group, Honeywell Laboratories, 3660 Technology Drive,
Minneapolis, MN 55418, USA
{goldman, musliner, pelican}@htc.honeywell.com

Abstract. Automatic controller synthesis and verification techniques promise to revolutionize the construction of high-confidence software. However, approaches based on explicit state-machine models are subject to extreme state-space explosion and the accompanying scale limitations. In this paper, we describe how to exploit an implicit, transition-based, representation of timed automata in controller synthesis. The CIRCA Controller Synthesis Module (CSM) automatically synthesizes hard real-time, reactive controllers using a transition-based implicit representation of the state space. By exploiting this implicit representation in search for a controller and in a customized model checking verifier, the CSM is able to efficiently build controllers for problems with very large state spaces. We provide experimental results that show substantial speed-up and orders-of-magnitude reductions in the state spaces explored. These results can be applied to other verification problems, both in the context of controller synthesis and in more traditional verification problems.

1 Introduction

This paper describes techniques for exploiting implicit representations in timed automaton controller synthesis. We show how reachability search exploits the implicit representation to substantially improve its efficiency. We have developed and implemented a system, the CIRCA Controller Synthesis Module (CSM), for automatic synthesis and execution of hard real-time discrete controllers. Unlike previous, game-theoretic algorithms [2, 8], the CSM derives its controller "on-the-fly" [14]. The CSM exploits a feature- and transition-based implicit representation of its state space, both in searching for the controller and in checking its correctness. Finally, the CSM generates memoryless and clockless controllers. These design elements substantially decrease the number of states that must be explored in the synthesis process.

The CSM is a component of the CIRCA architecture for intelligent control of mission-critical real-time autonomous systems [10, 11]. To permit on-line

* This material is based upon work supported by DARPA/ITO and the Air Force Research Laboratory under Contract No. F30602-00-C-0017.The authors thank Stavros Tripakis for many helpful suggestions. Thanks also to our anonymous referees.

reconfiguration, CIRCA has concurrently-operating controller synthesis (planning) and control (plan-execution) subsystems. The CSM uses models of the world (plant and environment) to automatically synthesize hard real-time safety-preserving controllers (plans). Concurrently a separate Real-Time Subsystem (RTS) executes the controllers, enforcing response time guarantees. The concurrent operation means that the computationally expensive methods used by the CSM will not violate the tight timing requirements of the controllers.

This paper discusses how the CSM's controller synthesis algorithm interacts with a model-checking reachability search algorithm that exploits the implicit representation. This technique substantially improves verification efficiency; by two orders of magnitude for large examples. We start by introducing the CIRCA CSM and its transition- and feature-based representation. Then we outline the forward search algorithm that the CSM uses to synthesize controllers, pointing out the role played by timed automaton verification. Next we explain how to formulate the execution semantics of the CIRCA model as a construction of sets of timed automata. The timed automaton model provides the semantics, but does not provide a practical approach for verification. We describe methods for model-checking that exploit CIRCA's implicit, transition-based, state space representation. We conclude with a comparison to related work in controller synthesis and AI planning.

2 The Controller Synthesis Module

CIRCA's CSM automatically synthesizes real-time reactive discrete controllers that guarantee system safety when run on CIRCA's Real-Time Subsystem (RTS). 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. Discrete states of the system are modeled as sets of feature-value assignments. Thus the transition descriptions, together with specifications of initial states, implicitly define the set of possible system states.

For example, Fig. 1 shows several transitions taken from a problem where CIRCA is to control the Cassini spacecraft in Saturn Orbital Insertion [4, 12]. This figure also includes the initial state description.

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 before the action occurs.

Temporal transitions represent uncontrollable processes, some of which may need to be preempted. See Sect. 2.1 for the definition of "preemption" in this context. Associated with each temporal transition is a *lower bound* on its delay. Transitions whose lower bound is zero are referred to as "events," and are handled specially for efficiency reasons.

```
;; The action of switching 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 have a limited amount of time to fix the problem before
;; the spacecraft will go 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
```

Fig. 1. Example transition descriptions given to CIRCA's planner.

Reliable temporal transitions represent continuous processes that may need to be employed by the CIRCA agent. Reliable temporal transitions have both upper and lower bounds on their delays.

While in the worst case an implicit representation is not superior to explicit state space enumeration, in practice there are substantial advantages. In many problems, vast sub-spaces of the state space are unreachable, either because of the control regime, or because of consistency constraints. The use of an implicit representation, together with a constructive search algorithm, allow us to avoid enumerating the full state space. The transition-centered representation allows us to conveniently represent processes that extend over multiple states. For example, a single transition (e.g., warming up a piece of equipment) may be extended over multiple discrete states. A similar representational convenience is often achieved by multiplying together many automata, but expanding the product construction restores the state explosion. Finally, in this paper we show how the transition-based implicit representation can be exploited in a verifier.

2.1 CSM Algorithm

Given problem representations as above, the controller synthesis (planning) problem can be posed as *choosing a control action for each reachable discrete*

state (feature-value assignment) of the system. Note that this controller synthesis problem is simpler than the general problem of synthesizing controllers for timed automata. In particular, CIRCA's controllers are memoryless and cannot reference clocks. This restriction has two advantages: first, it makes the synthesis problem easier and second, it allows us to ensure that the controllers we generate are actually realizable in the RTS.

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.[1]

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. Transitions that lead to failure states *must* be preempted.
3. Choose a control action or `no-op` for that state.
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.

During the course of the search algorithm, the CSM will use the verifier module after each assignment of a control action (see step 4). This means that the verifier will be invoked before the controller is complete. At such points we use the verifier as a conservative heuristic by treating all unplanned states 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. Note that this process converges to a sound and complete verification when the controller synthesis process is complete. When the verifier indicates that a 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.

For those familiar with designs for game-theoretic synthesis of controllers for timed systems [2, 8], the CSM algorithm is the same in its purpose. One difference is that the CSM algorithm works starting from an initial state and building forward by search. The game-theoretic algorithms, on the other hand, typically use a fixpoint operation to find a controllable subspace, starting from unsafe states (or other synthesis failures). Another difference is that the CSM

[1] Note that in some cases a reliable temporal transition, e.g., the warming up of the backup IRU, can be the transition that preempts a failure.

algorithm heavily exploits its implicit state space representation. Because of these features, for many problems, the CSM algorithm is able to find a controller without visiting large portions of the state space.

Two further remarks are worth making. The first is that the search described here is *not* made blindly. We use a domain-independent heuristic, providing limited lookahead, to direct the search. We do not have space to describe that heuristic here; it is based on one developed for AI planning [9]. Without heuristic direction, even small synthesis problems can be too challenging. The second is that we have developed an alternative method of search that works by divide-and-conquer rather than reasoning forward [6]. For many problems, this supplies a substantial speed-up. Again, we do not have space to discuss this approach in depth here.

3 Modeling for Verification

The CSM algorithm described above operates entirely in the discrete domain of the timed problem. This ensures that the controllers may be easily implemented automatically. 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 states on a path. The CSM uses a timed automaton verification system to ensure that the controllers the CSM builds are safe. In this section, we discuss a formal model of the RTS, expressed in terms of timed automata. The following section describes how to reason about this model efficiently.

3.1 Execution Semantics

The controllers of the CIRCA RTS are not arbitrary pieces of software; they are intentionally very limited in their computational power. These limitations serve to make controller synthesis computationally efficient and make it simpler to build an RTS that provides timing guarantees. 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. Note that these test expressions do not have access to any clocks. A sample TAP for the Saturn Orbit Insertion domain is given in Fig. 2.

The set of TAPs that make up a controller are assembled into a loop and scheduled to meet all the TAP deadlines. Note that in order to meet deadlines, this loop may contain multiple copies of a single TAP. The deadlines are computed from the delays of the transitions that the control actions must preempt.

3.2 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.

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

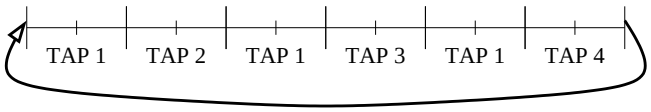

Fig. 2. A sample Test-Action Pair and TAP schedule loop from the Saturn Orbit Insertion problem.

Definition 1 (Timed Automaton [3].). *A timed automaton A is a tuple $\langle \mathcal{S}, s^i, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I} \rangle$ where $\mathcal{S}$ is a finite set of locations; s^i is the initial location; $\mathcal{X}$ is a finite set of clocks; $\mathcal{L}$ is a finite set of labels; $\mathcal{E}$ is a finite set of edges; and $\mathcal{I}$ 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 $\rho \subseteq \mathcal{X}$ is a clock reset. Timing constraints ($\Psi_{\mathcal{X}}$) appear in guards and invariants and clock assignments. In our models, all clock constraints are of the form $c_i \leq k$ or $c_i > k$ for some clock c_i and integer constant k. Guards dictate when the model* may *follow an edge, invariants indicate when the model* must *leave a state. In our models, all clock resets re-assign the corresponding clock to zero; they are used to start and reset processes. The state of a timed automaton is a pair: $\langle s, C \rangle$. $s \in \mathcal{S}$ is a location and $C : \mathcal{X} \rightarrow \mathbf{Q} \geq 0$ is a clock valuation, that assigns a non-negative rational number to each clock.*

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 $\langle (s_1, s_2), l, \psi_1 \cup \psi_2, \rho_1 \cup \rho_2, (s_1', s_2') \rangle$.*
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 $l \in \mathcal{L}_2 \setminus \mathcal{L}_1$.*

3.3 Modeling CIRCA with Timed Automata

We give the semantics of CSM models in terms of sets of interacting timed automata (see Fig. 3). Using multiple automata allows us to accurately capture the interaction of multiple, simultaneously operating processes. The starting point of the translation is the CIRCA plan-graph, constructed by the CIRCA CSM:

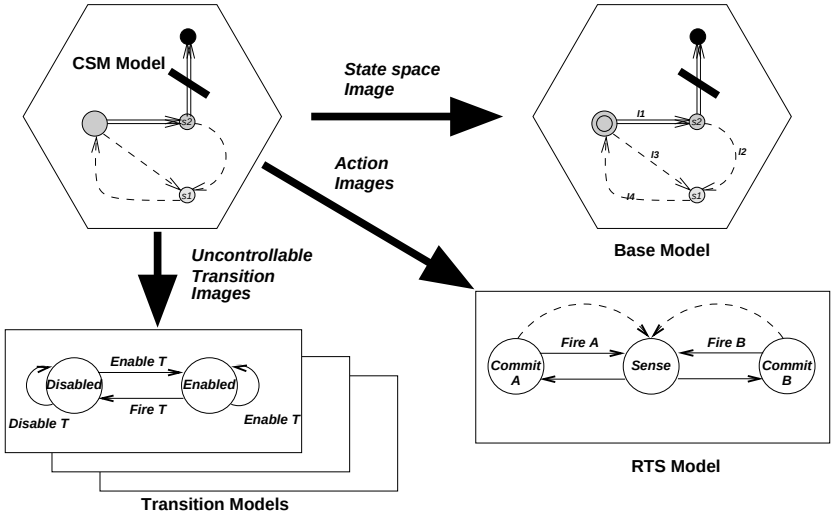

Fig. 3. The verifier model and its relation to the CSM model.

Definition 3 (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.*
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. Each transition,* t*, has an associated delay* (Δ_t) *lower and upper bound:* $lb(\Delta_t)$ *and* $ub(\Delta_t)$*. For temporals* $ub(\Delta_t) = \infty$*, for events* $lb(\Delta_t) = 0, ub(\Delta_t) = \infty$*.*
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.*
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.*
11. $\pi : S \mapsto 2^U$ *is a set of preemptions the CSM expects.*

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. 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). $\beta(\mathcal{P}) = \left\langle \theta(S), \left\{ l^0 \right\}, \emptyset, \Sigma(\mathcal{P}), \theta_E(\mathcal{P}), I_\top \right\rangle$ *where:*

1. $\theta(S) = \{\theta(s) \mid s \in S\} \cup \left\{ l^{\mathcal{F}}, l^0 \right\}$ *is the image under* θ *of the state set of* $\mathcal{P}$. *This image contains a location for each state 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 synchronize correctly.

Definition 6 (Label set for $\theta(\mathcal{P})$).
$\Sigma(\mathcal{P}) = \{\mathbf{e}_u, \mathbf{d}_u, \mathbf{f}_u \mid u \in U\}\ (1) \cup \{\mathbf{f}_a \mid a \in A\}\ (2) \cup \{\mathbf{r}_a\}$

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:*[2]

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

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

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, $l^{\mathcal{F}}$, when a preemption fails. (3) captures the effects of the uncontrollable transitions the CSM 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. 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.

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

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 "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 Fig. 3). 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).

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

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

$$I(e_u) = c_u \leq ub(\Delta_u) \text{ and } I(d_u) = \top$$

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

Definition 10 (RTS Model).

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

There are two classes of safety violations the verifier must detect. The first is a failure to successfully preempt some nonvolitional transition. This case is caught by transitions (2) of Definition 7. The second 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.[3]

4 Exploiting the Model in Verification

A direct implementation of the above model will suffer a state space explosion. To overcome this, we have built a CIRCA-specific verifier (CSV) able to exploit CIRCA's implicit state-space representation. The CSV constructs its timed automata, *both the individual automata and their product*, in the process of computing reachability. This on-the-fly computation relies on the factored representation of the discrete state space and on the limitations of CIRCA's RTS.

The efficiency gains from our factored state representation come in the computation of successor states. A naive implementation of the search would compute all of the locations (distinct discrete states) of the timed automaton up front, but many of those might be unreachable. We compute the product automaton lazily, rather than before doing the reachability search, thus constructing only reachable states.

The individual automata, as well as their product, are computed on-the-fly. The timed automaton formalism permits multiple automata to synchronize in arbitrary ways. However, CIRCA automata synchronize in only limited ways. There will be only one "primary transition" that occurs in any state of the CIRCA product automaton: either a controlled transition that is part of the RTS automaton, or a single uncontrolled transition. Thus we may dispense with component transitions and their labels.

The transitions that synchronize with the primary transition are of three types:

1. updates to the world automaton, recording the effect (the postconditions) of the primary jump on the discrete state of the world;
2. enabling and disabling jumps that set the state of uncontrolled transitions in the environment;
3. a jump that has the effect of activating the control action planned for the new state.

Accordingly, we can very efficiently implement a lazy successor generation for a set of states $S = \langle s, \mathbf{C} \rangle$, where s is a discrete state and $\mathbf{C}$ is a symbolic representation of a class of clock valuations, in our case a difference-bound matrix. When one needs to compute the successor locations for the location s, one

[3] Checking for the race condition is not fully implemented in our current version; its implementation is in progress as of this writing.

Table 1. Comparison of run times with different search strategies (Forward and DAP), timed automaton verifier (RTA) versus CIRCA-specific verifier (CSV). Times are given in milliseconds.

		Forward			DAP		
Scenario	Size	Kronos	RTA	CSV	Kronos	RTA	CSV
1	1920	9288	190	188	22431	483	417
2	72	6777	173	124	7070	385	309
3	100	4765	114	97	4399	783	385
4	560	5619	138	156	5599	366	288
5	3182592	∞	∞	∞	∞	∞	16278
6	40304	16983	762	568	506897	3035	1349
7	191232	258166	23030	25194	12919	4102	1833
8	191232	14637	652	533	436450	1157849	79855
9	991232	231769	21923	15474	∞	∞	2254
10	448512	∞	1063321	466631	∞	∞	5661
11	411136	∞	1064518	444657	∞	∞	5571
12	193536	37500	2585	1568	321626	3382	1626
13	129024	56732	3453	2933	77022	9958	1218
14	4592	16025	478	427	20220	1251	1036
15	7992	4680	183	176	75941	7672	5568
16	768	11535	426	337	13983	859	621
17	120	5730	100	368	5695	754	680
18	2880	16425	1349	1102	28922	2484	1669
19	192	6474	170	117	5715	331	308
20	768	9016	303	246	6870	564	416

need only compute a single outgoing edge for the RTS transition and make one outgoing edge for each uncontrollable transition.

Making the outgoing edges is a matter of (again lazily) building the successor locations and determining the clock resets for the edge. The clocks that must be reset are: (a) For each uncontrolled transition that is enabled in the successor location, but not enabled in the source location, s, add a clock reset for the corresponding transition; (b) If the action planned for the successor location is different from the action planned for the source location, reset the action clock. These computations are quite simple to make and much easier than computing the general product construction.

Our experimental results show that the CSV substantially improves performance over KRONOS [15] and also over a conventional model checker (denoted "RTA") that we built into CIRCA before developing the CSV. Table 1 contains comparison data between the conventional verifiers and the CSV, for two different search strategies.[4] The columns marked "forward," correspond to the algorithm described in this paper. The columns marked "DAP" correspond to the divide-and-conquer alternative [6]. The times, given in milliseconds, are for

[4] The problems are available at: `http://www.htc.honeywell.com/projects/ants/`

runs of the CSM on a Sun UltraSparc 10, SPARC v. 9 processor, 440 MHz, with 1 gigabyte of RAM. An ∞ indicates a failure to find an automaton within a 20 minute (i.e., $t > 1,200,000$) time limit.

To give a sense of the raw size of the problems, the "Size" column presents a worst-case bound on the number of *discrete* states for the final verification problem of each scenario. This value is computed by multiplying the number of possible CSM world model states (for the base model) times the number of transition model states ($2^{|U|}$) times the number of RTS model states ($|A|+1$).

Using the forward search strategy, the CSV is faster on 16 out of 20 scenarios. Using DAP, the CSV is faster on all 20 trials. The probability of these occurring, if the CSV and the conventional verifier were equally likely to win on any given trial, is .0046 and .000019, respectively. Table 1 indicates a speed-up of two orders of magnitude on the larger scenarios, numbers 9-11, using DAP.

Table 2 shows the state space reductions achieved by exploiting the implicit representation. This table compares the total number of states visited by each verifier in the course of controller synthesis.

A few facts should be noted: A verifier will be run many times in the course of synthesizing a controller. To minimize this, a number of cheaper tests filter controller synthesis choices in advance of verification, in order to avoid verification search whenever possible. The comparison is only with KRONOS used *as a component of the CSM*, not KRONOS as a general verification tool. Finally, the computations done by KRONOS and RTA are of a special-purpose product model that is slightly simpler and *less* accurate than the CSV's model.

5 Related Work

Asarin, Maler, Pneuli and Sifakis (AMPS) [2, 8] independently developed a game-theoretic method of synthesizing real-time controllers. This work stopped at the design of the algorithm and derivation of complexity bounds; to our knowledge it was not implemented. The AMPS approach has been implemented for the special case of automatically synthesizing schedulers [1]. The "planning as model checking" [5] approach is similar to work on game-theoretic controller synthesis, but limited to purely discrete systems.

Kabanza [7]'s SIMPLAN is very similar to our CSM. However, SIMPLAN adopts a discrete time model and uses domain-specific heuristics.

Tripakis and Altisen (TA) [14] have independently developed a controller synthesis algorithm for discrete and timed systems, that also uses forward search with on-the-fly generation of the state space. Note that on-the-fly synthesis has been part of the CIRCA system since its conception in the early 1990s [10, 11]. TA's on-line synthesis has some different features from ours. They allow for multiple control actions in a single state, and they allow the controller to consult clocks. TA's implicit representation of the state space is based on composition of automata, as opposed to our feature and transition approach. We hope to compare performance of CIRCA and a recent implementation of the TA algorithm [13].

Table 2. Comparison of state spaces explored with different search strategies (Forward and DAP), timed automaton verifier (RTA) versus CIRCA-specific verifier (CSV). Units are verifier state objects, i.e., a location $\times$ a difference-bound matrix.

Scenario	Forward RTA	Forward CSV	DAP RTA	DAP CSV
1	30	30	30	33
2	34	34	33	33
3	15	15	15	15
4	18	18	18	18
5	153147	229831	170325	3069
6	122	120	500	54
7	2826	5375	631	83
8	146	131	301165	24065
9	4361	4799	163133	259
10	219885	129329	184972	871
11	219885	129329	189184	871
12	585	513	509	99
13	685	675	1782	93
14	106	106	141	142
15	17	17	1054	1389
16	117	101	131	116
17	27	27	29	25
18	284	290	355	269
19	18	18	18	18
20	63	60	33	34

TA do not have a fully on-the-fly algorithm for timed controller synthesis. Their algorithm requires the initial computation of a quotient graph of the automata, in turn requiring a full enumeration of the discrete state space. The disadvantages of such an approach can be seen by considering the state space sizes of some examples, given in Table 1. We do not need to pre-enumerate the quotient, since we build only a clockless reactive controller and so can use the cruder time abstraction, which we compute on-the-fly. Note that this means that there are some controllers that TA (and AMPS) can find, that we cannot. However, clockless reactive controllers are easy to implement automatically, and this is not true of controllers that employ clocks. Also, and again because of the size of the state space, we use heuristic guidance in our state space search.

6 Conclusions

In this paper, we have presented the CIRCA controller synthesis algorithm, provided a timed automaton model for CIRCA CSM problems, and shown how a CIRCA-specific verifier (CSV) algorithm can exploit the features of the model. The CSV shows dramatic speed-up over a general-purpose verification algorithm.

While our model was developed for CIRCA, it is a general model for supervisory control of timed automata, and could readily be used in other applications.

References

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

[2] E. Asarin, Oded Maler, and Amir Pneuli. Symbolic controller synthesis for discrete and timed systems. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Proceedings of Hybrid Systems II*. Springer Verlag, 1995.

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

[4] Erann Gat. News from the trenches: An overview of unmanned spacecraft for AI. In Illah Nourbakhsh, editor, *AAAI Technical Report SSS-96-04: Planning with Incomplete Information for Robot Problems*. American Association for Artificial Intelligence, March 1996.

[5] Fausto Giunchiglia and Paolo Traverso. Planning as model-checking. In *Proceedings of ECP-99*. Springer Verlag, 1999.

[6] Robert P. Goldman, David J. Musliner, Kurt D. Krebsbach, and Mark S. Boddy. Dynamic abstraction planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 680–686, Menlo Park, CA, July 1997. American Association for Artificial Intelligence, AAAI Press/MIT Press.

[7] Froduald Kabanza. On the synthesis of situation control rules under exogenous events. In Chitta Baral, editor, *Theories of Action, Planning, and Robot Control: Bridging the Gap*, number WS-96-07, pages 86–94. AAAI Press, 1996.

[8] Oded Maler, Amir Pneuli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In Ernst W. Mayr and Claude Puech, editors, *STACS 95: Theoretical Aspects of Computer Science*, pages 229–242. Springer Verlag, 1995.

[9] Drew McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1 – 2):111–159, April 1999.

[10] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1561–1574, 1993.

[11] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74(1):83–127, March 1995.

[12] David J. Musliner and Robert P. Goldman. CIRCA and the Cassini Saturn orbit insertion: Solving a prepositioning problem. In *Working Notes of the NASA Workshop on Planning and Scheduling for Space*, October 1997.

[13] Stavros Tripakis, January 2002.

[14] Stavros Tripakis and Karine Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In J. Wing, J. Woodcock, and J. Davies, editors, *Formal Methods 1999*, volume I of *Lecture Notes in Computer Science*, pages 233–252. Springer Verlag, Berlin, 1999.

[15] S. Yovine. Kronos: A verification tool for real-time sytems. In *Springer International Journal of Software Tools for Technology Transfer*, volume 1, October 1997.