Model-based Verification for Automatic Synthesis of Real-time Controllers
Extended Abstract

Robert P. Goldman, David J. Musliner, Michael J. S. Pelican
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418
{goldman; musliner; pelican}@htc.honeywell.com

Introduction

We have developed a novel technique for automatically synthesizing hard real-time reactive controllers using model-checking verification. Our algorithm builds a controller incrementally, using a timed automaton model to check each partial controller for correctness. The verification model captures both the controller design and the semantics of its execution environment. If the controller is found to be incorrect, information from the verification system is used to direct the search for improvements. This paper describes how our controller synthesis process uses verification, and explains in detail how we model the execution of the real time subsystem of the CIRCA intelligent control architecture.

We are developing autonomous, flexible control systems for mission-critical applications such as Unmanned 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 intelligent control 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.

CIRCA uses model-checking techniques for timed automata (Alur 1998; Yovine 1998) as an integral part of its controller synthesis algorithm. CIRCA's *Controller Synthesis Module* (CSM) incrementally builds a hard real time reactive controller from a description of the processes in its environment, the control actions available and a set of goal states. To do this, the Controller Synthesis Module must build a model of the controller it is constructing that is faithful to its execution semantics, and use this model to verify that the controller will function safely in its environment.

CIRCA

The CIRCA architecture is intended to provide intelligent control to autonomously-operating systems.[1] To do this, CIRCA must operate at multiple time scales. CIRCA must be able to reason about the profile of a mission as a whole. For example, if CIRCA is operating an Uninhabited Combat Aerial Vehicle (UCAV), its mission-level planning must be able to reason about issues like fuel use and navigation to its goal. At a lower level, CIRCA must have a controller that is able to react to threats and opportunities that arise in its immediate environment. For example, when targeted by enemy radar, the CIRCA-controlled UCAV must carry out countermeasures (e.g., release chaff) and initiate evasive maneuvers. Furthermore, CIRCA must guarantee that these reactions will be taken in time. It is not enough to *eventually* release chaff; CIRCA must inspect its environments for threats sufficiently often, and must react to those threats within specified time bounds.

CIRCA employs two strategies to manage this complex task. First, its mission planner decomposes the mission into more manageable subtasks that can be planned in detail. Second, CIRCA itself is decomposed into two concurrently-operating subsystems (see Figure 1): an *AI Subsystem* (AIS) reasons about high-level problems that require powerful but potentially unbounded computation, while a separate *real-time subsystem* (RTS) reactively executes the AIS-generated plans and enforces guaranteed response times. The AIS

[1] CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics, simulated autonomous aircraft, space probe challenge problems (Musliner & Goldman 1997) and controlling a fixed-wing model aircraft (Atkins *et al.* 1998).

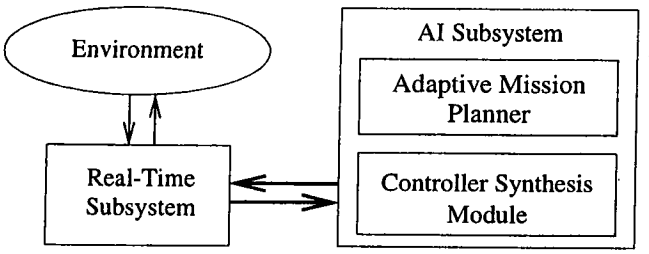

Figure 1: Basic CIRCA architecture.

contains the CSM, which is the focus of this paper, as well as the mission planner and some support modules, none of which we will discuss here.

The Controller Synthesis Module (CSM) bridges mission-level planning and reactive control. It takes descriptions of a phase of a system mission and dynamically, automatically, synthesizes a set of reactions that maintain the system's safety and move it towards its goals. When this controller is operating, the CSM will be working to generate controllers for other phases of the mission.

The Controller Synthesis Module

The CIRCA CSM builds reactive discrete controllers that observe the system state and some features of its environment and take appropriate control actions. In constructing such a controller, the CSM takes a description of the processes in the system's environment, represented as a set of transitions that modify world features and that have worst case time characteristics. From this description, CIRCA incrementally constructs a set of reactions and checks them for correctness using a timed automaton verifier.

The real-time controllers that CIRCA builds sense features of the system's state (both internal and external), and execute reactions based on the current state. That is, the CIRCA RTS runs a *memoryless* reactive controller. Given the above limitation on the form of the controller, the controller synthesis 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.

Note that the question of whether a transition is preempted is not a question that can be answered based on local information: preemption of a transition, t in a state, s is a property of the controller as a whole, not of the individual state. For example, to know when a bomb is going to go off in a room with you, you can't just consider how fast you can throw the bomb out the window — you must also consider how long its timer has been running before you got to the state in which you will throw it out the window. It is this non-local aspect of the controller synthesis problem that has led us to use automatic verification.

Representing a control problem

To describe a domain to CIRCA, the user inputs a set of transition descriptions that *implicitly* define the set of possible system states. These transitions are of four types:

Action transitions represent actions performed by the RTS.

Temporal transitions represent the progression of time and continuous processes that may need to be preempted.

Event transitions represent world occurrences as instantaneous state changes.

Reliable temporal transitions represent continuous processes (such as the operation of a control law) that may need to be employed by the CIRCA agent.

For example, Figure 2 shows several transitions used in a situation where CIRCA is to control the Cassini spacecraft in Saturn Orbital Insertion.[2]

CSM algorithm

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

1. Choose an element of the set of reachable states (at the start of controller synthesis, only the initial state(s) is(are) reachable).
2. Choose a control action (an action or a reliable temporal) for that state.
3. Invoke the verifier to confirm that the (partial) controller is safe.
4. If the controller is *not* safe, use information from the verifier to direct backtracking.
5. If the controller *is* safe, recompute the set of reachable states.
6. If there are no unplanned reachable states (reachable states for which a control action has not been chosen), terminate successfully.
7. If some unplanned reachable states remain, loop to step 1.

Figure 3 provides a simple "comic-book" illustration of the process of controller synthesis. Initially (i), there

[2]The problem is taken from Erann Gat's "From the Trenches" (Gat 1996).

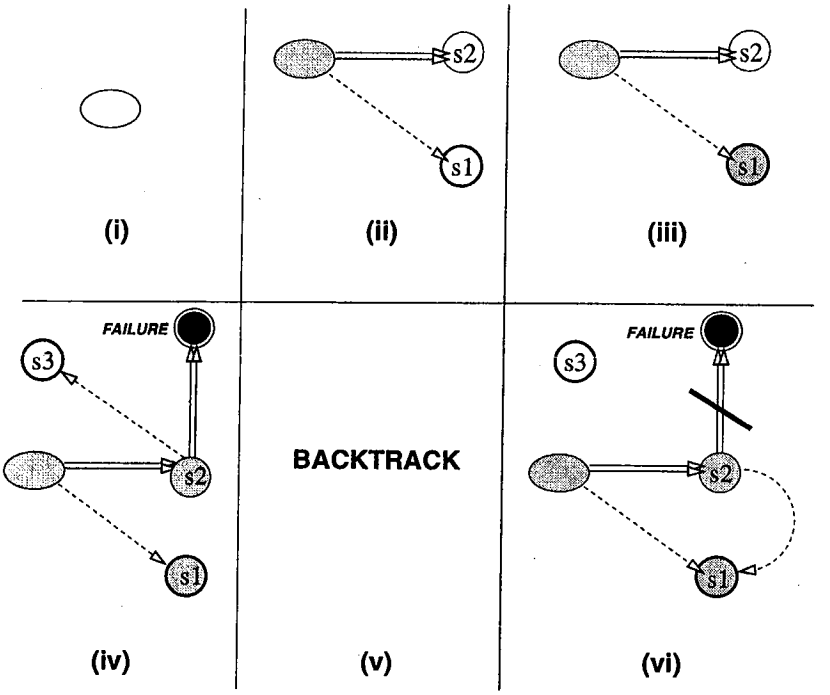

Figure 3: A simple example of controller synthesis.

```
;; 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
```

Figure 2: Example transition descriptions given to CIRCA's planner.

is only one state reachable, the initial 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 to the failure state (black). This triggers a backtrack (v), and the system chooses an alternative action (vi) that will carry the system to *s1* (instead of *s3*). This alternative action *does* preempt the transition to the failure state (dark bar superimposed on the transition arrows), so the controller is safe. (vi) shows how the set of reachable states may vary as the controller synthesis process proceeds: at this point *s3* is no longer reachable, since the CSM has chosen not to employ the action that made it reachable in (iv). 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 trace that enters these states ends and is 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.

Acknowledgments

This material is based upon work supported by DARPA/ITO and the Air Force Research Laboratory under Contract No. F30602-00-C-0017.

References

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

Atkins, E. M.; Miller, R. H.; VanPelt, T.; Shaw, K. D.; Ribbens, W. B.; Washabaugh, P. D.; and Bernstein, D. S. 1998. Solus: An autonomous aircraft for flight control and trajectory planning research. In *Proceedings of the American Control Conference (ACC)*, volume 2, 689–693.

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. Available at `http://www-aig.jpl.nasa.gov/home/gat/gp.html`.

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.