Synthesis-Specific Verification

(Extended Abstract)

David J. Musliner and Robert P. Goldman
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{musliner,goldman}@htc.honeywell.com

Overview

To build robust, reliable autonomous systems, we have been developing the CIRCA approach to real-time intelligent control. Our goal is to give a CIRCA-controlled autonomous system models of what it can do, what its goals are, and what the environment can do. From those models, we want CIRCA to automatically generate and execute hard-real-time controllers that are guaranteed to avoid failure and achieve the system's goals whenever possible.

A key component of our approach is the integration of *formal verification* into the synthesis process. We use formal verification to ensure that the controllers CIRCA builds are guaranteed to avoid system failure states. This abstract briefly describes the current status of our verification system, and the motivation for a set of improvements we are making to form a new verifier system (the Synthesis-Specific Verifier (SSV)) specialized to the controller synthesis process.

Brief Review of CIRCA

CIRCA was designed to integrate complex algorithms for intelligent control with a hard real-time control execution environment (Musliner, Durfee, & Shin 1995; Musliner *et al.* 1999). CIRCA generates controllers by reasoning about models of the system to be controlled, its primitive functions, and the goals to be achieved. As illustrated in Figure 1, CIRCA's controller synthesis and execution subsystems operate in parallel. The Controller Synthesis Module (CSM) reasons about an internal model of the world and dynamically generates discrete-event controllers. Within the CSM, as shown in Figure 2, the State-Space Planner (SSP) and Scheduler modules cooperate to develop controllers that will ensure system safety and attempt to achieve system goals when executed by the Real-Time Subsystem (RTS). The separate RTS reactively executes the SSP-generated controller and enforces guaranteed response times. While the RTS is executing this controller, maintaining system safety and stability, the SSP is able to continue executing controller synthesis methods to find the next appropriate controller. When the new controller has been developed, it can be downloaded to the RTS. CIRCA's dynamic controller synthesis helps address the problem of large state spaces: dynamic synthesis permits CIRCA to use a sequence of smaller, situation-specific controllers rather than a single, much larger controller that can handle all eventualities.

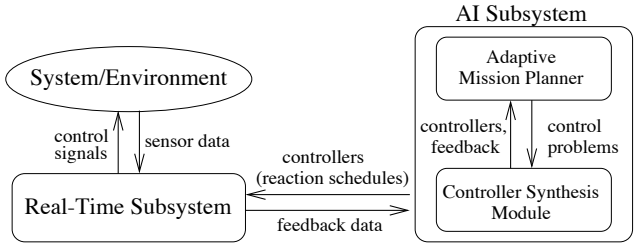

Figure 1: The Cooperative Intelligent Real-Time Control Architecture.

The Role of Verification

CIRCA's State-Space Planner builds controllers based on a world model and a set of formally-defined safety conditions (Musliner, Durfee, & Shin 1995). To describe a domain to CIRCA, the system designer inputs a set of transition descriptions capturing the significant events that can occur in the uncontrolled system/world. These transition descriptions implicitly define the set of reachable states. The SSP plans by generating a clocked finite state machine (FSM) from these transition descriptions. The SSP assigns to each

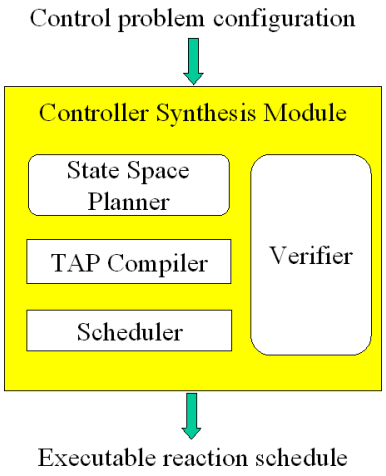

Figure 2: The CSM uses search and verification to build controllers.

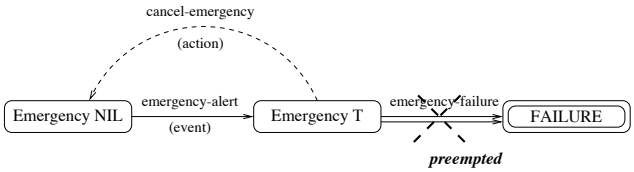

Figure 3: Preemption is the basis for automatic synthesis of guaranteed safe/stable real-time discrete-event controllers.

reachable state either an action transition or `no-op`. It selects actions to drive the system towards goal states and to *preempt* transitions that lead to failure. Figure 3 shows a trivial preemption example in which the SSP has recognized the reachability of a failure state (potentially representing an undesirable instability or a real catastrophic failure) and has selected a suitable action that is guaranteed to definitely occur before the transition to failure can possibly occur.

System safety is guaranteed by planning action transitions that preempt *all* transitions to failure, making the failure state unreachable (i.e., constructing a stable set of states). In order to verify that the CIRCA SSP's plans are safe, we must project what will happen when they are executed. To do this, the SSP uses a separate verifier 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). Initially, we used an off-the-shelf, general-purpose timed automaton verifier, KRONOS (Yovine 1998). We have since moved to using one of our own.

During synthesis, if the verifier detects that a failure state is reachable, it returns a path (an execution trace) from an initial state of the system to the failure state. The SSP uses the verification system's output to identify and repair the problem. The SSP maps entries in the execution trace into entries in its search stack. Each search stack entry corresponds to the assignment of a control action to a reachable state.[1] Thus for each location that appears in an execution trace, there is a corresponding decision in the SSP's stack. This set of decisions, taken together, make up a *nogood*: 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 nogood. 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 automatically exploit 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 by hand to evaluate a design must locate errors by extensive inspection of a program trace leading to failure. CIRCA automates this process.

The use of the verifier in the synthesis process means that CIRCA controllers are "correct by construction." That is, if the model, the SSP and the verifier are correct, the controller they generate will be correct. In turn, the RTS will ensure that the correctly-constructed controller will execute correctly.

Synthesis-Specific Verification

Our current integration of the controller synthesis and verification processes is simple and uses existing verification techniques. Unfortunately, interfacing our search-based synthesis methods to off-the-shelf verification systems is not an ideal solution, for several reasons:

Inefficient representations — While synthesis systems such as CIRCA reason about factored representations of state spaces, verification systems such as Kronos and HyTech (Henzinger, Ho, & Wong-Toi

[1] This is a slight oversimplification, but sufficient to grasp the essentials.

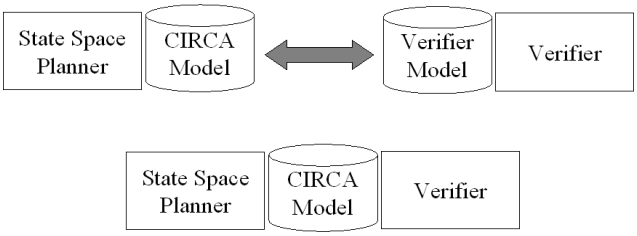

Figure 4: Using a synthesis-specific verifier will avoid model translation costs.

1997) do not. CIRCA does not generate the entire state space of a system it is trying to control; instead, it uses a factored "generator" representation of transitions to lazily create its representation of each state *only* when that state is found to be reachable. As a result, the huge number of system states that are either innately unreachable or unreachable because of designed controller actions are never generated, explored, or reasoned about in any way. Standard verification systems have no similar ability to lazily generate representations of system states. Instead, they must suffer the enormous exponential cost, in both computation time and space, of representing every possible state. Leveraging our experience with the factored representation approach, we will build this more efficient behavior into the SSV.

Data replication —
Translating state machine models from the synthesis representation into the input language required by a verifier (e.g., timed automata for Kronos) is both time-consuming and wasteful, as it extensively duplicates the synthesis model to form the verifier model, which is promptly discarded after verification. The costs are incurred on the state representations (as above) and also on the representation of transitions. Each CIRCA-style model of a transition is actually a description of many possible instantiated edges in a standard verifier model. Since the verifier is used repeatedly to verify partial controllers during the synthesis process, this overhead is very significant. By driving the new Synthesis-Specific Verifier module directly off the synthesis models, our new CIRCA system will avoid this inefficiency.

Inability to exploit discrete structure —
Existing verifiers for hybrid systems are very good at exploiting structure in the continuous domains they model. For example, timed automata verifiers such as Kronos are designed to reason efficiently about temporal information; they provide very powerful automated techniques to recognize useful abstractions in time values (e.g., "clock-zones" and "regions"), but they are not so cunning with discrete information such as state features. If a verification system could have more in-depth knowledge of the behavioral structure of a state space (i.e., how the states can be connected by transitions), it could use that information to computational advantage. The SSV will have access to precisely that information, through the factored representation of the synthesis engine.

To avoid these pitfalls and provide highly efficient, scalable hybrid systems verification services tailored to the needs of CIRCA's controller synthesis algorithms, we are designing a new SSV with features including:

Close integration with CIRCA — By sharing modeling languages with the CIRCA synthesis engine, the SSV will avoid translation costs and storage penalties.

Factored state space handling —
Working directly off the CIRCA system model, the SSV will take advantage of the built-in facilities to reason about connectivity without pre-enumerating all states.

Efficient dual abstractions — The SSV will exploit the same types of temporal abstractions used by other verifiers, with the added benefit of also exploiting discrete-space abstractions provided by Dynamic Abstraction Planning (Goldman *et al.* 1997).

Optimized verification functions — Because the SSV is not intended to be a completely general purpose verifier, we can restrict the scope of its functionality and optimize the verification queries it does support. Specializing to supporting controller synthesis may, for example, avoid the need to maintain certain types of bookkeeping information.

SSV Status

We have completed a preliminary implementation of the SSV concept that addresses some of the concerns described above. In particular, the current version is able to lazily generate discrete states, and it works directly off the synthesis model. However, the SSV has not yet been integrated and tested with the Dynamic Abstraction Planning version of the SSP, which we expect to result in very large performance improvements.

Despite these limitations, preliminary experiments show that the performance of the unoptimized SSV is

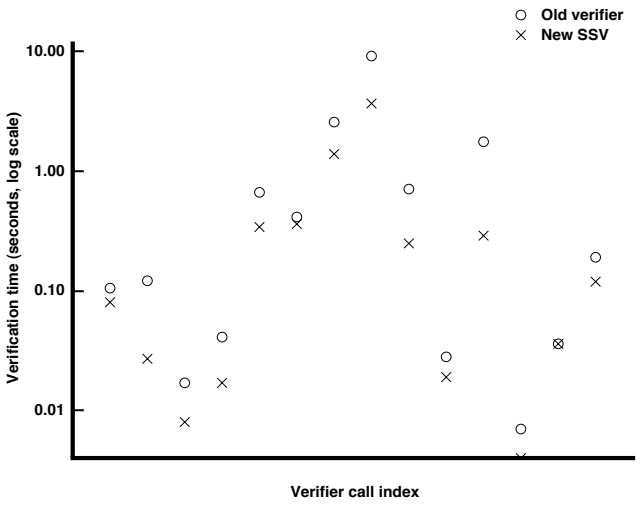

Figure 5: Scatter plot of verification times for the old verifier and the new SSV.

a dramatic improvement over the prior version. Running the old and new verifiers on a variety of different domains, the cumulative verification time was 31.4 seconds for the old verifier and just 13.2 seconds for the SSV, a 58% improvement. Figure 5 shows a scatter plot contrasting the verification time for 14 different calls to the verifiers, showing the consistent improvement in performance. We expect further improvements as we take advantage of more of the synthesis-specific characteristics noted above.

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*.

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

Goldman, R. P.; Musliner, D. J.; Krebsbach, K. D.; and Boddy, M. S. 1997. Dynamic abstraction planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 680–686. Menlo Park, CA: American Association for Artificial Intelligence.

Henzinger, T. A.; Ho, P.-H.; and Wong-Toi, H. 1997. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer* 1:110–122.

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. 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.