IEEE Transactions on Consumer Electronics, Vol. 40, No. 4, NOVEMBER 1994

STATE PROBLEMS IN PROGRAMMING HUMAN-CONTROLLED DEVICES

Joseph A. Konstan
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

Many consumer goods are complicated enough to benefit from programmed control. Today's home electronics devices support a wide range of options and controls. At the same time, personal digital assistants and programmable remote controls are now capable of learning and generating control sequences to control a wide range of devices. Unfortunately, most device interfaces are designed for interactive human control rather than programmed control.

This paper analyzes state-based obstacles to programming devices designed for interactive human control. It develops a theory of statelock, a condition in which a control program is unable to synchronize with the state machine underlying the controlled device.The paper also presents design strategies to avoid statelock and applies these strategies to the home audio/video and telephone autodialer domains.

KEYWORDS: Device interface, programmable remote control, automata, user/machine systems, audio/video control, telephone autodialers.

INTRODUCTION

Many consumer goods are complicated enough to benefit from programmed control. Today's home electronics devices support a wide range of options and controls.

At the same time, the emergence of personal digital assistants has created new possibilities for programmed device control. Basic PDA's can dial stored phone numbers. More advanced ones can also send messages to computers or facsimile machines, control remote devices using tone dialing, or even store and play back infrared control sequences such as are used for controlling televisions and other consumer audio/video devices.

Unfortunately most devices are not designed for programmed control. Consumer electronics devices can easily be controlled by a human with an infrared remote control, but only because the human can observe the state of the device and act accordingly. Programmed control units lack state awareness, and accordingly are generally unable to achieve even simple goals.

This paper discusses state awareness problems in building programmable controllers for devices with interfaces designed for interactive use. The next section describes in detail the problems involved with programming a controller for home audio/video equipment. It also formalizes the problems by defining *statelock*—a condition that inhibits programmability. The following sections present theoretical results to show that state-awareness problems are fundamental and present design strategies for avoiding statelock. The last section discusses the results and shows how they apply to a different control problem—the use of automatic telephone dialers-- and presents other related obstacles to building programmable controls for consumer electronics devices.

THE PROBLEM

"How can this 'universal' remote be programmed to enter 'home theater' mode?" A learning universal remote control unit controls multiple devices (e.g., televisions, video cassette recorders, stereo systems, etc.) by learning the infrared commands from those devices' individual controls. Many units come with sequence programming features to allow the user to define complex operations that are invoked by a single button. A typical goal is to define a "home theater" button that turns the television to a designated channel (typically channel 3 or 4 in the U.S.), turns the VCR on and sets it to display onto the TV, selects VCR input on the stereo system, and sets a moderate volume on the stereo (and no volume on the TV). All together, this

Revised manuscript received September 15, 1994

mode would control three devices directly and perhaps others indirectly (selecting VCR input turns off the CD and cassette players on some models of receiver) to provide the type of home entertainment so often advertised!

Unfortunately, there is one problem involved in programming the home theater button: in most cases, it cannot be done. In fact, there are many simpler operations that cannot be programmed. This is not due to the incompetence of the programmer, nor due to the weakness of the universal remote control device. Rather, the problem is one of poor programmability in the devices being controlled.

To illustrate the fundamental problem in programming remote-controllable devices, examine the simplest operation that cannot be programmed on many TV sets. There is no way to program a button to turn on the TV. There is a power button, but it operates as a toggle. It will turn on the TV if it is already off, and it will turn it off if it is already on. This implementation usually works well when a live human is operating the remote control (at worst, an error can be easily corrected), but a program has no way of knowing whether the TV is on or off. For the home theater button, the user would have to either define two buttons, one when starting with the TV off and another when starting with it on, or assume the TV is always on (or off) before entering theater mode (and therefore having to remember to make it so before pressing the magic key).

The problem, simply stated, is that many TVs, VCRs, and other devices have internal states that are not always known to the remote control. Controllable devices are simple finite state machines and remote control units generate command tokens that trigger transitions between states. Figure 1, for example, shows the simplest useful device—one which has two states, on and off. This state machine could support

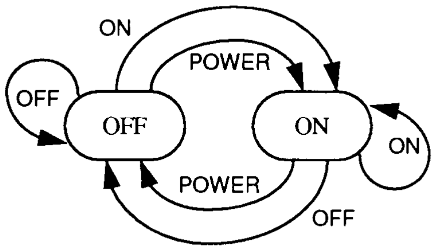
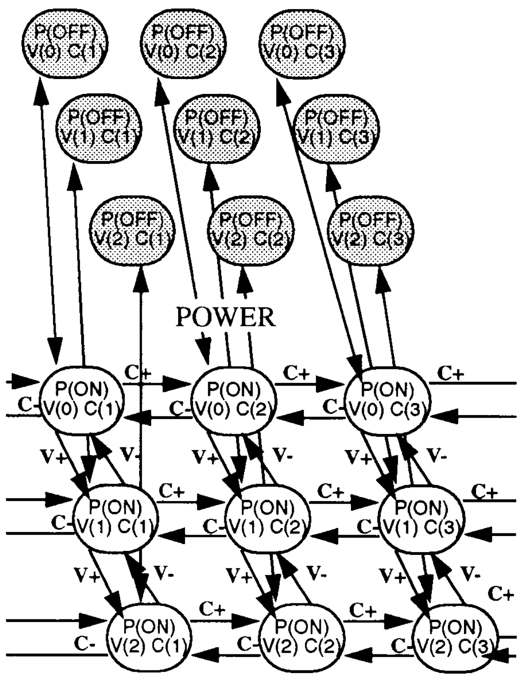


Figure 1. TV Power State Machine

three useful command tokens, ON, OFF, and POWER (a toggle). With ON and OFF, the machine is completely programmable remotely, since commands can force it into a specific state regardless of the state in which the device starts. With POWER only, the machine is not remotely programmable if the starting state is not known. Interestingly, a machine with POWER and either ON or OFF is completely programmable (e.g., with POWER and OFF, OFF turns the machine off, OFF followed by POWER turns it ON) as long as excess intermediate states do not matter.

An actual home entertainment system has many more controls and many more states. Most of these states are made visible to a live user (or can be made visible by using a display function), though many of them are hidden until a transition moves the machine into a more visible state. Figure 2 shows a simplified model of a typical TV with states corresponding to 3 volume levels, three channels, and power and with transitions for volume up and down, channel up and down, and power toggle (adding direct channel access would render the diagram completely unreadable; an alternative representation for complex state machines is presented in [3]). The states shown in grey are "hidden." They can-



note: unmarked transitions remain in same state

Figure 2. Simple TV State Diagram

not be distinguished merely by looking at the device. They are not the same state, however, since each encodes a different volume/channel pair.

The problem of remote programmability is the problem of creating a sequence of command tokens (i.e., and input string) that always leave the device(s) being controlled in the same desired state. (several analogous problems are discussed below).

In general, absolute access is easy to program and toggles are hard to program. In an n-dimensional state space, absolute access can be used along each dimension to reach the desired state. Relative access can only be programmed if there are fixed ends (e.g., a minimum and maximum volume) after which the command token causes the device to remain in the same state. Any level can be attained by first attaining a fixed level (e.g., by transmitting a number of VOLUME DOWN tokens larger than the number of volume levels) and then moving to the destination level through relative commands (e.g., VOLUME UP). Relative access in a cycle, including the two-element toggle, cannot generally be programmed.

In practice, device state machines are more complicated than this simple model suggests. Even the basic TV state machine shown in figure 2 has an asymmetry in it. When the power is off, channel and volume command tokens do not cause transitions to other states. Accordingly, any program must first turn the power on and then use volume and channel commands.

More complicated devices have still more complicated state machines. A typical VCR has several toggles with unusual interactions: the POWER toggle, the TV/VCR toggle, the TIMER toggle, and various PLAY/STOP, RECORD/STOP, and PAUSE/SLOW toggles. Figure 3 shows the interaction pattern for the first three of these states in a particular model, leaving out other states and transitions such as channel selection and play/record/stop/pause. Programming the VCR requires a nearly complete knowledge of the state machine, and some programs are still not possible because toggles such as PAUSE and SLOW are time- and context-dependent.

Problem Summary

Most devices designed for human control model state machines with internal state and externally defined command tokens. The user selects command tokens based on the externally visible state attributes of the machine. Programmable control, however, is more difficult because the start state of the machine is unknown. Accordingly, many such devices cannot be programmed to reach a specific state.

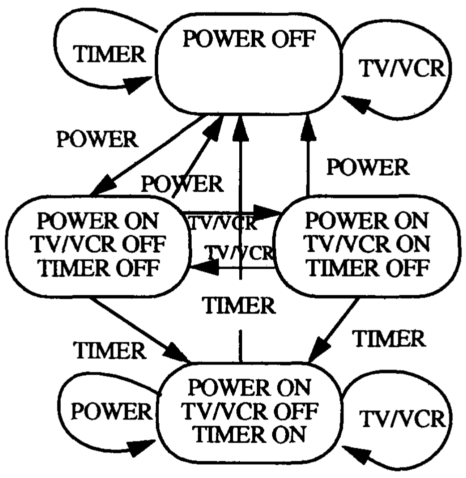

POWER OFF
TIMER
TV/VCR
POWER
POWER
POWER
POWER ON TV/VCR OFF TIMER OFF
TV/VCR
TV/VCR
POWER ON TV/VCR ON TIMER OFF
TIMER
TIMER
TIMER
POWER
POWER ON TV/VCR OFF TIMER ON
TV/VCR

Figure 3. VCR State Interaction

The term *statelock* is used to refer to this lack of programmability. There are four conditions for statelock:

- Controlled devices must have internal state.
- The remote control program cannot determine the state of the device.
- There is no fixed string of command tokens to bring the device to a known state.
- The device state can change without the program being aware.

The first three conditions were discussed above.

Even with these conditions, however, it is possible to synchronize the device and the remote control program if the program is always made aware of any commands. A programmable remote control for a television, for example, could be synchronized with the television at a certain channel, volume, power status, etc. This possibility leads to the fourth condition which states that external agents can change the device state without the program's knowledge (e.g., a user can control the device manually or the device state can be altered by environmental conditions such as power failure).

It is also useful to include an assumption that any state can be reached from any other through a sequence of command tokens. Without this assumption, statelock could occur simply by leaving the device in a state from which the goal state is inaccessible. For practical purposes, this assumption is true for almost any consumer device. Some notable exceptions are discussed at the end of this paper.

The four conditions of statelock are formulated so as to allow programmability to be established by nullifying a single condition. The first condition is fundamental to both the home audio/video example and the telephone example discussed below, and is likely to be true for any interesting device. The other three conditions, however, can be avoided. The next section presents prior theoretical work on determining whether a given device can be forced into a known state with a fixed command string. The following section discusses device and command set design options that avoid statelock by nullifying each of the last three conditions.

THEORETICAL BACKGROUND

While little theoretical research has been done on remote control applications themselves, results in automata theory and its applications can be applied to the remote control problem. The second and third conditions of statelock (i.e., unknown state and no fixed string to lead to a known state) correspond to the distinguishing sequence and synchronizing sequence problems for finite state machines [2].

The distinguishing sequence problem seeks either a string of tokens that generates a different output for each initial state in a finite state machine. Not every finite state machine has a preset distinguishing sequence (and it is a PSPACE-complete problem to find a preset distinguishing sequence). Adaptive distinguishing sequences, which change the input string based on output, can be found in polynomial time and have a bounded length of $O(n^2)$ [6]. Remote control devices cannot generally take advantage of the machines that have distinguishing sequences, even when the sequences are known, since they are not capable of observing and analyzing output.

Synchronizing sequences are strings of input tokens that take a finite state machine to a specific state regardless of the initial state. Any machine with a synchronizing sequence can be programmed by first using the synchronizing sequence to reach the known state and then sending a command string to reach the goal state from the known state.

Not all finite state machines have synchronizing sequences. Further, those synchronizing sequences that exist have a length bounded by $O(n^3)$ which is impractical for use with most current consumer electronic devices (e.g., a mid-range television typically has at least 3200 states: 80 channels * 10 volume levels * 2 power states * 2 mute states). Because of this impracticality, the third condition for statelock can be extended to state that there is no short fixed string of command tokens to bring the device to a known state. "Short" can be defined by context as the number of command tokens that can be transmitted, received, and processed in a suitable time interval (e.g., three to five seconds if a user activates the option, perhaps a minute if the option is timer-activated).

Since finite state machine theory cannot nullify the second and third conditions of statelock, and since it does not address the first and fourth conditions, it is necessary to design more restrictive interfaces or automata to ensure programmability.

DESIGN STRATEGIES

This section presents four design strategies for ensuring programmability of remote controlled devices by avoiding statelock.

Ask and Ye Shall Know

One way to avoid statelock is to allow the remote control program to determine the state of the controlled device. Some high-end video products (specifically frame addressable video disk and video cassette players) provide a two-way communication link (generally serial RS232 communications) between the device and a controlling computer. The command set includes state queries (e.g., what frame is displayed, what is the play/pause/stop status, etc.) that generate replies. Remote control programs operate by querying the device state and sending appropriate commands to reach the goal state.

A full communication interface not only solves the statelock problem but also has other programming benefits. The remote control device can include conditional execution (e.g., eject only if there is a tape or disk loaded) and can be given access to any information available in the device itself.

Full communication interfaces, however, are much more expensive and complicated to implement. They require a two-way communication link between the remote control and the controlled device. Wire links are economical, but they limit portability and mobility. Worse yet, the programming complexity requires that either the remote control or the program itself accurately model the state machine of the controlled device in order to determine the correct sequence of commands needed to reach the goal state.

Riding the Bus

Many consumer-level electronics devices are designed to work together with other devices made by the same manufacturer and to share a single remote control. They communicate through a command bus that broadcasts all significant state-changing actions. One company's consumer audio components, for example, communicate through a wired bus system to ensure that active devices are switched through the receiver and other devices are inactive (i.e., selecting PLAY, either manually or through a remote control, switches the receiver input and also selects STOP on other devices).

Programmable remote control devices could be designed to monitor this command bus. In doing so, they could prevent statelock by preventing the device from changing state without the program being aware.

Unfortunately, there are three major obstacles preventing widespread use of bus-monitoring remote controls: 1) the majority of devices do not yet support command busses, 2) even devices that support a command bus only broadcast commands of interest to other components—volume control and other "local" commands are not broadcast, and 3) bus-monitoring requires the same complexity and wire interface as full communication without presenting any significant long-term advantages.

Taking Control

Centralized control interfaces are a more practical alternative for ensuring that the remote control program is aware of all state changes and thereby avoiding statelock. Centralized control interfaces force all device commands to be routed to the remote control first, and then relayed to the controlled device. Several vendors sell components for assembling these interfaces for home automation. The typical kit includes infrared transmitters and receptors along with computer hardware and software to control the devices. User actions transmit signals to the computer program which then formulates an implementation and transmits commands to the individual devices. Interference between user-computer and computer-device communications is avoided either by physically isolating the infrared receptors of the devices or by using different frequencies and patterns. Users also must be prevented from using any manual controls on the device itself.

Centralized control has several advantages. First, it can be implemented to control any device that supports remote-control access. No communication interface or command bus is needed. The remote control system merely learns the commands that the device understands and uses them to control it. Second, centralized control separates human input from programmed control. Since the system requires a general-purpose computer anyway, it is more practical to develop a high-level programming environment. Third, users can use conventional remote control devices for interactive control simplifying the user interface for new users. Finally, centralized control is a well understood model in home automation and can be easily integrated into a home automation system.

Centralized control also has several disadvantages. First, there are situations in which centralized control loses synchronization with the devices. Even when user inputs are reliably directed to the central controller, certain state changes occur directly at the device level. Power failures, for example, tend to force state changes (even when the device is merely unplugged for a few minutes). Similarly, tape and disk player/recorders have physical state corresponding to the presence of (and perhaps writeability of) a tape or disk.

Second, centralized control is inherently non-portable. Centralized control also requires expensive hardware and extensive software and removes the customary manual command access method with which most users are familiar.

Accordingly, centralized control is best suited for environments where home automation is a design priority. Centralized controllers can use two-way communication when available and also provide interfaces to telephone input, sensors, and other home automation components.

Starting From Scratch

The final solution requires redesigning the command set and state machines used in remote controlled devices. It has already been shown that it is neither possible nor feasible to find a reset sequence for the state machine associated with an arbitrary device. It is possible, however, to design machines with easily-accessible reset sequences and more regular state machines. Doing so nullifies the third condition for statelock.

There are three approaches to this solution. First, one can simply define a ground state and implement a RESET command that always causes a transition to the ground state. On a television set, for example, the

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.