Chapter 9
Reactions & Delays

Synopsis Time-triggered control systems are an important control paradigm. Event-triggered controllers focus on correct responses to appropriate events, which are assumed to be detected perfectly, which simplifies their design and analysis but makes them hard to implement. Time-triggered controllers, instead, focus on reacting to changes within certain reaction delays. Implementations become more straightforward using controllers that repeatedly execute within a certain maximum time period, or execute periodically with at least a certain frequency. While time-triggered control models can be easier to develop than event-triggered control models, the additional effects of reaction delays complicate the control logic and safety arguments.

9.1 Introduction

Chapter 7 explained the central proof principle for loops using invariants. Chapter 8 studied the important feedback mechanism of event-triggered control and made crucial use of invariants for rigorously reasoning about event-triggered control loops. Those invariants uncovered important subtleties with events that could be easily missed. In Chap. 8, we, in fact, already noticed these subtleties thanks to our "safety first" approach to CPS design, which guided us to exercise the scrutiny of Cartesian Doubt on the CPS model before even beginning a proof.

However, even if the final answer for the event-triggered controller for the ping-pong ball was rather clear and systematic, event-triggered control had an unpleasantly large number of modeling subtleties in store for us. Even in the end, event-triggered control has a rather high level of abstraction, because it assumes that all events are detected perfectly and right away with continuous sensing. The event-triggered model has $x \leq 5$ as a hard limit in the evolution domain constraint of the differential equation to ensure that the event $4 \leq x \leq 5$ will never ever be missed as the ball is rushing upwards.

As soon as we want to implement such a perfect event detection, it becomes clear that real controller implementations can usually only perform discrete sensing, i.e.,

A. Platzer, *Logical Foundations of Cyber-Physical Systems*,
https://doi.org/10.1007/978-3-319-63588-0_9

checking sensor data every once in a while at certain discrete points in time, whenever new measurements come from the sensor and when the controller has a chance to check whether the measurement is about to exceed height 5. Most controller implementations, thus, only end up checking for an event every once in a while, whenever the controller happens to run, rather than permanently as event-triggered controllers pretend.

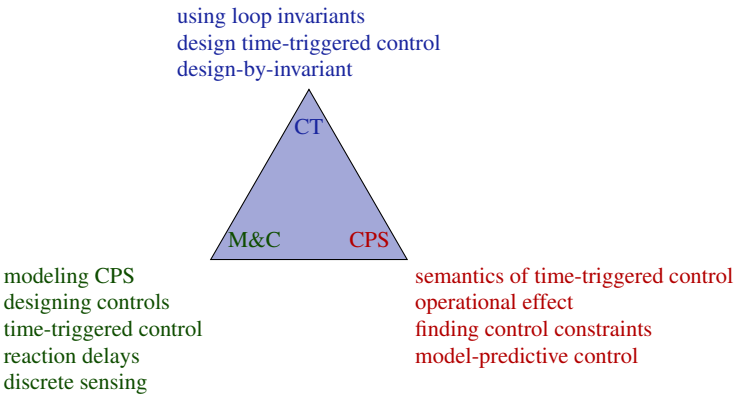
This chapter, thus, focuses on the second important paradigm for making cyber interface with physics to form cyber-physical systems: the paradigm of *time-triggered control*, which uses periodic actions to affect the behavior of the system only at discrete points in time with certain frequencies. This is to be contrasted with the paradigm from Chap. 8 of *event-triggered control*, where responses to events dominate the behavior of the system and an action is taken whenever one of the events is observed. The two paradigms play an equally important rôle in classical embedded systems and both paradigms arise naturally from an understanding of the hybrid program principle for CPS.

Based on the understanding of loops from Chap. 7, the most important learning goals of this chapter are:

Modeling and Control: This chapter provides a number of crucial lessons for modeling CPSs and designing their controls. We develop an understanding of time-triggered control, which is an important design paradigm for control loops in CPS. This chapter studies ways of developing models and controls corresponding to this feedback mechanism, which is easier to implement but will turn out to be surprisingly subtle to control. Knowing and contrasting both event-triggered and time-triggered feedback mechanisms helps with identifying relevant dynamical aspects in CPS coming from events and reaction delays. This chapter focuses on CPS models assuming discrete sensing, i.e., sensing at (nondeterministically chosen) discrete points in time.

Computational Thinking: This chapter uses the rigorous reasoning approach from Chapters 5 and 7 to study CPS models with time-triggered control. As a running example, the chapter continues to develop the extension from bouncing balls to vertical ping-pong balls, now using time-triggered control. We again add control decisions to the bouncing ball, turning it into a ping-pong ball, which retains the intuitive simplicity of the bouncing ball, while enabling us to develop generalizable lessons about how to design time-triggered control systems correctly. The chapter will crucially study invariants and show a development of the powerful technique of design-by-invariant in a concrete example.

CPS Skills: This chapter develops an understanding of the semantics of time-triggered control. This understanding of the semantics will guide our intuition about the operational effects of time-triggered control and especially the impact it has on finding correct control constraints. An understanding of both is crucial for finding good tradeoffs to determine which parts of a model are faithfully understood as event-triggered and where time-triggered control is more accurate. Finally, the chapter studies some aspects of higher-level model-predictive control.



9.2 Delays in Control

Event-triggered control is a useful and intuitive model matching our expectation that controllers react in response to certain critical conditions or events that necessitate intervention by the controller. However, one of its difficulties is that event-triggered control with its continuous-sensing assumption can be hard or impossible to implement in reality. On a higher level of abstraction, it is very intuitive to design controllers that react to certain events and change the control actuation in response to what events have happened. Closer to the implementation, this turns out to be difficult, because actual computer control algorithms do not actually run all the time, only sporadically every once in a while, albeit sometimes very often. Implementing event-triggered control faithfully would, in principle, require permanent continuous monitoring of the state to check whether an event has happened and respond appropriately. That is not particularly realistic, because fresh sensor data will only be available every once in a while, and controller implementations will only run at certain discrete points in time, causing delays in processing. Actuators may sometimes take time to get going. Think of the reaction time it takes you to turn the insight "I want to hit this ping-pong ball there" into action so that your ping-pong paddle will actually hit the ping-pong ball. Sometimes the ping-pong paddle acts early, sometimes late; see Fig. 9.1. Or think of the time it takes to react to the event "the car in front of me is turning on its red taillights" by appropriately applying the brakes.

Back to the drawing board. Let us reconsider the original dL formula (8.3) for the ping-pong ball (Fig. 9.1) from which we started out to design the event-triggered version in (8.7).

$$\begin{aligned} &0 \le x \land x \le 5 \land v \le 0 \land g > 0 \land 1 \ge c \ge 0 \land f \ge 0 \rightarrow \\ &\quad \big[\big(\{x' = v, v' = -g \,\&\, x \ge 0\}; \\ &\qquad \mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}(4 \le x \le 5)\, v := -fv\big)^*\big](0 \le x \le 5) \end{aligned} \tag{8.3*}$$

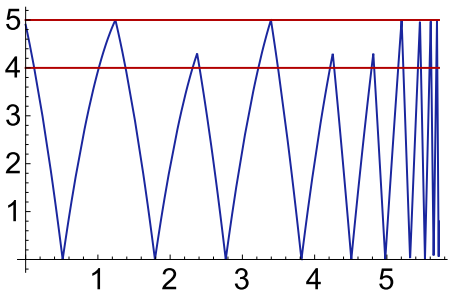


Fig. 9.1 Sample trajectory of a ping-pong ball (plotted as position over time) with the indicated ping-pong paddle actuation range, sometimes actuating early, sometimes late

This simplistic formula (8.3) turned out not to be valid, because its differential equation was not guaranteed to be interrupted when the event $4 \leq x \leq 5$ happens. Consequently, (8.3) needs some other evolution domain constraint to make sure all continuous evolutions are stopped at some point for the control to have a chance to react to situation changes. Yet, it should not be something like $\ldots \& x \leq 5$ as in (8.7), because continuously monitoring for $x \leq 5$ requires permanent continuous sensing of the height, which is difficult to implement.

Note 50 (Physical versus controller events) The event $x = 0$ in the (physics) controller as well as the (physics) evolution domain constraint $x \geq 0$ for detecting the event $x = 0$ are perfectly justified in the bouncing-ball and ping-pong-ball models, because both represent physics. Physics is very well capable of keeping a ball above the ground, no matter how much checking for $x = 0$ it takes to make that happen. The ball just does not suddenly fall through the ground because physics looked the other way and forgot to check its evolution domain constraint $x \geq 0$! In our controller code, however, we need to exercise care when modeling events and their reactions. The controller implementations will not have the privilege of running all the time, which only physics possesses. Cyber happens every once in a while (even if it may execute quite quickly and quite frequently), while physics happens all the time. Controllers cannot sense and compute and act literally all the time.

How else could the continuous evolution of physics be interrupted to make sure the controller actually runs? By bounding the amount of time that physics is allowed to evolve before running the controller again. Before we can talk about time, the model needs to be changed to include some variable, let's call it t, that reflects the progress of time with a differential equation $t' = 1$:

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[\big(\{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1\}; \\ &\qquad \mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}(4 \leq x \leq 5)\, v := -fv\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9.1}$$

Of course, the semantics of hybrid programs included some notion of time already, but it was inaccessible in the program itself because the duration r of differential

equations was not a state variable that the model could read (Definition 3.2). No problem, (9.1) simply added a time variable t that evolves along the differential equation $t' = 1$ just like time itself does. In order to bound the progress of time by 1, the evolution domain includes $\ldots \& t \leq 1$ and declares that the clock variable t evolves with time as $t' = 1$.

Oops, that does not actually quite do it, because the HP in (9.1) restricts the evolution of the system so that it will never ever evolve beyond time 1, no matter how often the loop repeats. It imposes a global bound on the progress of time. That is not what we meant to say! Rather, we wanted the duration of each individual continuous evolution to be at most one second. The trick is to reset the clock t to zero by a discrete assignment $t := 0$ before the continuous evolution starts:

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[\big(t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1\}; \\ &\qquad \mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}(4 \leq x \leq 5)\, v := -fv\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9.2}$$

In order to bound the duration by 1, the evolution domain includes $\ldots \& t \leq 1$ and the variable t is reset to 0 by $t := 0$ right before the differential equation. Hence, t represents a local clock measuring how long the evolution of the differential equation was. Its bound of 1 ensures that physics gives the controller a chance to react at least once per second. The system could stop the continuous evolution more often and earlier, because this model has no lower bound on t. Even if possible, it is inadvisable to constrain the model unnecessarily by lower bounds on the duration.

Before going any further, let's take a step back to notice an annoyance in the way the control in (9.2) was written. It is written in the style in which the original bouncing ball and the event-triggered ping-pong ball were phrased: continuous dynamics followed by control. That has the unfortunate effect that (9.2) lets physics happen *before* control does anything, which is not a very safe start. In other words, the initial condition would have to be modified to assume the initial control choice was fine. That would duplicate part of the control into the assumptions on the initial state. Instead, let's switch the statements from *plant*;*ctrl* to *ctrl*;*plant* to make sure control always happens before physics does anything:

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[\big(\mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}(4 \leq x \leq 5)\, v := -fv; \\ &\qquad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1\}\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9.3}$$

Now that dL formula (9.3) has an upper bound on the time it takes between two subsequent control actions, is it valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

Even though (9.3) ensures a bound on how long it may take at most until the controller inspects the state and reacts, there is still a fundamental issue with (9.3).

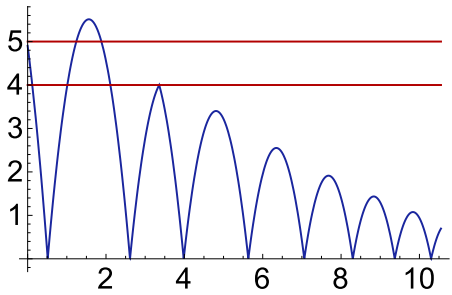


Fig. 9.2 Sample trajectory of a time-triggered ping-pong ball (as position over time), missing the first event

We can try to prove (9.3) and inspect the non-provable cases in the proof to find out what the issue is. Or we can just think about what could go wrong. The controller of (9.3) runs at the latest after one second (hence at least once per second) and then checks whether $4 \leq x \leq 5$. But if $4 \leq x \leq 5$ was not true when the controller ran last, there is no guarantee that this event will be detected reliably when the controller runs next. In fact, the ball might very well have been at $x = 3$ at the last controller run, then evolved continuously to $x = 6$ in a second and missed the event $4 \leq x \leq 5$ that it was supposed to detect (Exercise 9.2); see Fig. 9.2. Worse than that, the ping-pong ball has then not only missed the exciting event $4 \leq x \leq 5$ but already became unsafe.

Similarly, driving a car would be unsafe if you were to only open your eyes once a minute and monitor whether there is a car right in front of you. Too many things could happen in between that should prompt you to brake.

9.2.1 The Impact of Delays on Event Detection

How can this problem with formula (9.3) be solved? How can the CPS model make sure the controller does not miss its time to take action? Waiting until $4 \leq x \leq 5$ holds true is not guaranteed to be the right course of action for the controller.

Before you read on, see if you can find the answer for yourself.

The problem with (9.3) is that its controller is unaware of its own delay. It does not take into account how the ping-pong ball could move further before it gets a chance to react next. If the ball is already close to the ping-pong paddle's intended range of actuation, then the controller had better take action already if it is not sure whether it can still safely wait to take action till next time the time-triggered controller runs.

Note 51 (Delays may miss events) Delays in controller reactions may cause events to be missed that it was supposed to monitor. When that happens, there is a discrepancy between an event-triggered understanding of a CPS and the real time-triggered implementation. Delays may make controllers miss events especially when slow controllers monitor events in relatively small regions for a fast-moving system. This relationship deserves special attention to make sure the impact of delays on a system controller cannot make it unsafe.
It is often a good idea to first understand and verify an event-triggered design of a CPS controller to identify correct responses to the respective events and subsequently refine it to a time-triggered controller to analyze and verify that CPS in light of its reaction time. Discrepancies in this analysis hint at problems that event-triggered designs will likely experience at runtime and they indicate a poor event abstraction. Controllers need to be aware of their own delays to foresee what they might otherwise miss.

The ping-pong controller would be in trouble if $x > 5$ might already hold in its next control cycle after the continuous evolution, which would be outside the operating range of the ping-pong paddle (and already unsafe). Due to the evolution domain constraint, the continuous evolution can take at most 1 time unit, after which the ball will be at position $x + v - \frac{g}{2}$ as previous chapters already showed by solving the differential equation. Choosing gravity $g = 1$ to simplify the math, the controller would be in trouble in the next control cycle after 1 second, which would take the ball to position $x + v - \frac{1}{2} > 5$, if $x > 5\frac{1}{2} - v$ holds now.

9.2.2 Model-Predictive Control Basics

The idea is to make the controller now act based on how it predicts the state might evolve until the next control cycle (this is a very simple example of *model-predictive control* because the controller acts based on what its model predicts). Chap. 8 already discovered for the event-triggered case that the controller only wants to trigger the ping-pong paddle action if the ball is still flying up, not if it is already on its way down. Making (9.3) aware of the future in this way leads to

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[\big(\mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}((x > 5\frac{1}{2} - v) \wedge v \geq 0)\, v := -fv; \\ &\quad\quad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1\}\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9.4}$$

Is conjecture (9.4) about the future-aware controller valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

The controller in formula (9.4) has been designed based on the prediction that the future may evolve for 1 time unit. If an action will no longer be possible in 1 time unit, because the event $x \leq 5$ has passed in that future time instant, then the controller in (9.4) takes action right now already. That is a good start. The issue with that approach, however, is that there is no guarantee at all that the ping-pong ball will fly for exactly 1 time unit before the controller is asked to act again (and the postcondition is checked). The controller in (9.4) checks whether the ping-pong ball would be too far up after one time unit and does not intervene unless that is the case. Yet, what if the ball only flies for $\frac{1}{2}$ a time unit? Clearly, if the ball will be safe after 1 time unit, which is what the controller in (9.4) checks, it will also be safe after just $\frac{1}{2}$ a time unit, right?

Before you read on, see if you can find the answer for yourself.

Wrong! The ball may well be below height 5 again after 1 time unit but still could have been above 5 in between the current point of time and the time that is 1 time unit from now. Then the safety of the controller will be a mere rope of sand, because this incorrect controller will have a false sense of safety after having checked what happens 1 time unit from now, in complete ignorance of whether the behavior was actually safe until then. Such trajectories are shown in Fig. 9.3 from the same initial state and the same controller, just with different sampling periods. What a bad controller design if its behavior depends on the sampling period! But worse than that, such a bouncing ball will not be safe if it has been above 5 between two sampling points. After all, the bouncing ball follows a ballistic trajectory, which first climbs and then falls.

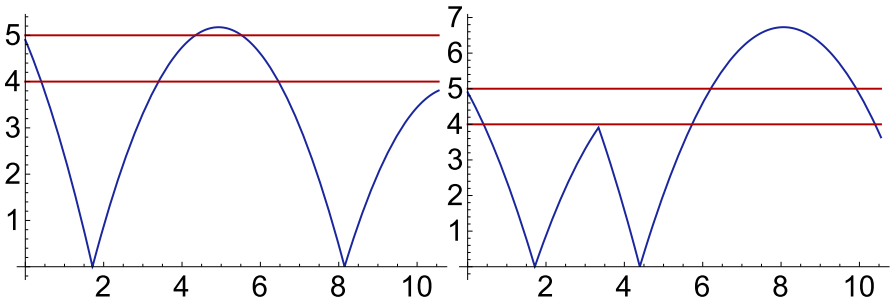


Fig. 9.3 Sample trajectory of a time-triggered ping-pong ball (as position over time), missing different events with different sampling periods

9.2.3 Design-by-Invariant

In order to get to the bottom of this, we need a quantity that tells us what the ball will do at all times, without mentioning the time variable explicitly, because we can hardly have the controller check its safety predictions at all times 0, 0.1, 0.25, 0.5, 0.786, ..., of which there are infinitely many anyhow.

Come to think of it, we were already investigating what we can say about a bouncing ball independently of the time when we were designing loop invariants for its proof in Sect. 7.4:

$$2gx = 2gH - v^2 \land x \geq 0 \land (c = 1 \land g > 0) \tag{7.11*}$$

This formula was proved to be an invariant of the bouncing ball, which means it holds true always while the bouncing ball is bouncing around. Invariants are the most crucial information about the behavior of a system that we can rely on all the time. Since (7.11) is only an invariant of the bouncing dynamics not the ping-pong ball, it, of course, only holds until the ping-pong paddle hits, which changes the control. But until the ping-pong paddle is used, (7.11) summarizes concisely all we need to know about the state of the bouncing ball at all times. Of course, (7.11) is an invariant of the bouncing ball, but it still needs to be true initially. The easiest way to make that happen is to assume (7.11) at the beginning of the ping-pong ball's life.[1]

Because (7.11) only conducted the proof of the bouncing ball invariant (7.11) for the case $c = 1$ to simplify the arithmetic, the ping-pong ball now adopts this assumption as well. To simplify the arithmetic and arguments, let us also adopt the assumption $f = 1$ in addition to $c = 1 \land g = 1$ for the proofs.

Substituting safety-critical height 5 for H in the invariant (7.11) for this instance of parameter choices leads to a condition when the energy exceeds safe height 5:

$$2x > 2 \cdot 5 - v^2 \tag{9.5}$$

as an indicator of the fact that the ball might end up climbing too high, because its energy would allow it to. Adding this condition (9.5) to the controller (9.4) leads to

$$\begin{aligned} &2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 \land 1 = c \land 1 = f \geq 0 \rightarrow \\ &\big[\big(\mathsf{if}(x = 0)\, v := -cv \,\mathsf{else}\, \mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2) \land v \geq 0)\, v := -fv; \\ &\quad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1\}\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9.6}$$

The bouncing ball invariant (7.11) is now also assumed to hold in the initial state.

[1] Note that H is a variable that does not need to coincide with the upper height limit 5 as it did in the case of the bouncing ball, because the ping-pong ball has more control at its fingertips. In fact, the most interesting case is if $H > 5$ in which case the ping-pong ball will only stay safe *because of* its control. One way to think of H is as an indicator of the energy of the ball showing how high it might jump up if not for all its interaction with the ground and the ping-pong paddle.

Is dL formula (9.6) about the time-triggered controller valid? As usual, it is best to use an invariant or a counterexample for justification.

Before you read on, see if you can find the answer for yourself.

Formula (9.6) is "almost valid." But it is still not valid for a very subtle reason. It is great to have the help of proofs to catch those subtle issues. The controller in (9.6) takes action for two different conditions on the height x. However, the ping-pong paddle controller actually only runs in (9.6) if the ball is not at height $x = 0$, otherwise the ground performs the control action of reversing the direction of the ball. Now, if the ball is flat on the floor already ($x = 0$) yet its velocity so incredibly high that it will rush past height 5 in less than 1 time unit, then the ping-pong paddle controller will not even have had a chance to react before it is too late, because it does not execute on the ground according to (9.6); see Fig. 9.4.

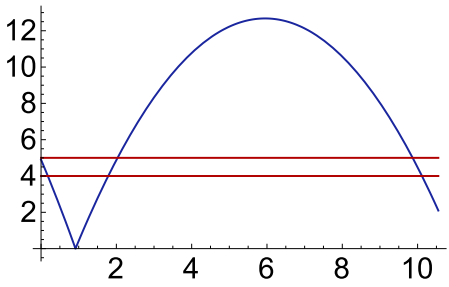


Fig. 9.4 Sample trajectory of a time-triggered ping-pong ball (as position over time), failing to control on the ground

9.2.4 Sequencing and Prioritizing Reactions

Fortunately, these thoughts already indicate how the problem with multiple control actions can be fixed. We turn the nested if-then-else cascade into a sequential composition of two separate if-then statements that will ensure the ping-pong paddle controller runs even if the bouncing ball is still on the ground (Exercise 9.3).

$$\begin{aligned}&2x = 2H - v^2 \land 0 \le x \land x \le 5 \land v \le 0 \land g = 1 \land 1 = c \land 1 = f \rightarrow \\ &\big[\big(\mathsf{if}(x = 0)\, v := -cv \;;\; \mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2) \land v \ge 0)\, v := -fv; \\ &\quad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \ge 0 \land t \le 1\}\big)^*\big](0 \le x \le 5)\end{aligned} \tag{9.7}$$

Now, is formula (9.7) finally valid, please? If so, using which invariant? Otherwise, show a counterexample.

Before you read on, see if you can find the answer for yourself.

Yes, formula (9.7) is valid. What invariant can be used to prove formula (9.7)?
Formula (9.7) is valid, which, for $g = c = f = 1$, can be proved with this invariant:

$$2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5 \tag{9.8}$$

This invariant instantiates the general bouncing ball invariant (7.11) for the present case of parameter choices and augments it with the desired safety constraint $x \leq 5$.

Yet, is the controller in (9.7) useful? That is where the problem lies now. The condition (9.5) that is the second disjunct in the controller of (9.7) checks whether the ping-pong ball could possibly *ever* fly up all the way to height 5. If this is ever true, it might very well be true long before the bouncing ball even approaches the critical control cycle where a ping-pong paddle action needs to be taken. In fact, if (9.5) is ever true, it will also be true at the very beginning. After all, the formula (7.11), from which condition (9.5) derived, is an invariant, so always true for the bouncing ball. What would that mean?

That would cause the controller in (9.7) to take action right away at the mere prospects of the ball ever being able to climb way up high, even if the ping-pong ball is still close to the ground and pretty far away from the last triggering height 5. That would make the ping-pong ball quite safe, since (9.7) is a valid formula. But it would also make it rather conservative and would not allow the ping-pong ball to bounce around nearly as much as it wants to. It would make the bouncing ball lie flat on the ground, because of an overly anxious ping-pong paddle. That is a horrendously acrophobic bouncing ball if it never even starts bouncing around in the first place. And the model would even require the (model) world to end, because there can be no progress beyond the point in time where the ball gets stuck on the ground. How can the controller in (9.7) be modified to resolve this problem?

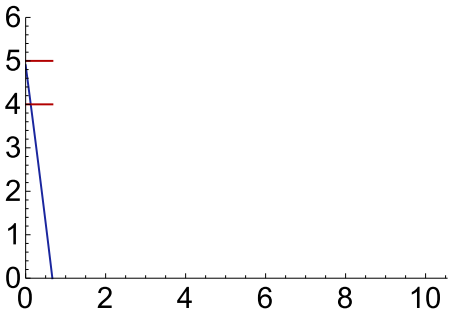


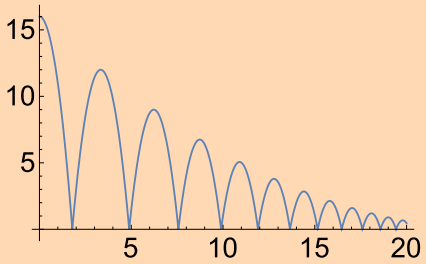
Fig. 9.5 Sample trajectory of a time-triggered ping-pong ball (as position over time), stuck on the ground

Before you read on, see if you can find the answer for yourself.

The idea is to restrict the use of the second if-then disjunct (9.5) in (9.7) to slow velocities, in order to make sure it only applies to the occasions that the first controller disjunct $x > 5\frac{1}{2} - v$ misses, because the ball will have been above height 5 in between. Only with slow velocities will the ball ever move so slowly that it is

Expedition 9.1 (Zeno paradox)

There is something quite surprising about how (9.7) may cause time to freeze. But, come to think of it, time did already freeze in mere bouncing balls!



The duration between two hops on the ground of a bouncing ball keeps on decreasing rapidly. If, for simplicity, the respective durations are $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$, then these durations sum to

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{1-\frac{1}{2}} = 2$$

which shows that the bouncing-ball model will make the (model) world freeze almost to a complete stop, because it can never reach time 2 nor any time after. The bouncing ball model disobeys what is called *divergence of time*, i.e., that the real time keeps diverging to ∞. The reason this model prevents time from progressing beyond 2 is that the bouncing-ball model switches directions on the ground more and more frequently. This may be very natural for bouncing balls, but can cause subtleties and issues in other control systems if they switch infinitely often in finite time.

The name *Zeno paradox* comes from the Greek philosopher Zeno (ca. 490–430 BC) who found a paradox when fast runner Achilles gives the slow Tortoise a head start of 100 meters in a race: In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead – recounted in Aristotle, Physics VI:9, 239b15.

Pragmatic solutions to counteract the Zeno paradox in bouncing balls add a statement to the model that makes the ball stop when the remaining velocity on the ground is too small. For example:

$$\mathsf{if}(x = 0 \wedge -0.1 < v < 0.1)\,(v := 0; \{x' = 0\})$$

This statement switches to a differential equation that does not change position but, unlike the differential equation $x' = v, v' = -g \,\&\, x \geq 0$ for the bouncing ball, can be followed for any duration when $x = 0 \wedge v = 0$.

near its turning point to begin its descent and start falling down again before 1 time unit. And only then can the first condition miss that the ball is able to evolve above 5 within 1 time unit. When is a velocity slow in this respect?

For the ball to turn around and descend, it first needs to reach velocity $v = 0$ by continuity (during the flying phase) on account of the mean-value theorem. In gravity $g = 1$ the ball can reach velocity 0 within 1 time unit exactly when its velocity was $v < 1$ before the differential equation, because the velocity changes according to $v(t) = v - gt$. Consequently, adding a conjunct $v < 1$ to the second disjunct in the controller makes sure that the controller only checks for turnaround when it might actually happen during the next control cycle.

$$\begin{aligned}&2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 \land 1 = c \land 1 = f \rightarrow \\ &\big[\big(\mathsf{if}(x = 0)\, v := -cv; \mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2 \land v < 1) \land v \geq 0)\, v := -fv; \\ &\quad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1\}\big)^*\big](0 \leq x \leq 5)\end{aligned} \tag{9.9}$$

This dL formula is valid and provable with the same invariant (9.8) that was already used to prove (9.7). It has a much more aggressive controller than (9.7), though, so it is more fun for the ping-pong ball to bounce around with it.

Note 52 (Design by invariant) Designing safe controller actions by following the system invariant is a great idea. After having identified an invariant for the bare-bones system (such as the bouncing ball), the remainder of the control actions can be designed safely by ensuring that each of them preserves the invariant. For example, the ping-pong paddle is used if the ball might violate the invariant. Some care is needed to avoid limiting the system unnecessarily. The reaction time determines which control cycle has the last chance to act to keep the invariant maintained. Of course, design-by-invariant does not extend to changing the laws of physics to please our controllers. But once the appropriate invariants have been identified for physics, the design of the controller can follow the objective of always maintaining the safety-critical invariants.

9.2.5 Time-Triggered Verification

The easiest way of proving that dL formula (9.9) is valid is to show that the invariant (9.8) holds after every line of code. Formally, this reasoning by lines corresponds to a number of uses of the generalization proof rule MR from Lemma 7.4 to show that the invariant (9.8) remains true after each line if it was true before. The first statement $\mathsf{if}(x = 0)\, v := -cv$ does not change the truth-value of (9.8), i.e.,

$$2x = 2H - v^2 \land x \geq 0 \land x \leq 5 \rightarrow [\mathsf{if}(x = 0)\, v := -cv](2x = 2H - v^2 \land x \geq 0 \land x \leq 5)$$

is valid, because, when $c = 1$, the statement can only change the sign of v and (9.8) is independent of signs, because the only occurrence of v satisfies $(-v)^2 = v^2$. Similarly, the second statement $\mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2 \land v < 1) \land v \geq 0)\, v := -fv$ does not change the truth-value of (9.8). That is the formula

$$2x = 2H - v^2 \land x \geq 0 \land x \leq 5 \rightarrow [\mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2 \land v < 1) \land v \geq 0)\, v := -fv] (2x = 2H - v^2 \land x \geq 0 \land x \leq 5)$$

is valid, because, at least for $f = 1$, the second statement can also only change the sign of v, which is irrelevant for the truth-value of (9.8). Finally, the relevant parts of (9.8) are a special case of (7.11), which has already been shown to be an invariant for the bouncing-ball differential equation and, thus, continues to be an invariant when adding a clock $t' = 1 \,\&\, t \leq 1$, which does not occur in (9.8). The additional invariant $x \leq 5$ that (9.8) has compared to (7.11) is easily taken care of using the corresponding knowledge about potential height H.

Note 53 (Time-triggered control) One common paradigm for designing controllers is *time-triggered control*, in which controllers run periodically or pseudo-periodically with certain frequencies to inspect the state of the system. Time-triggered systems are closer to implementation than event-triggered control. They can be harder to build, however, because they invariably require the designer to understand the impact of delay on control decisions. That impact is important in reality, however, and, thus, effort invested in understanding the impact of time delays usually pays off in designing a safer system that is robust to bounded time delays.

Partitioning the hybrid program in the verified dL formula (9.9) into the parts that come from physics (typographically marked like physics) and the parts that come from control (typographically marked like control) leads to the following.

Proposition 9.1 (Time-triggered ping-pong is safe). *This* dL *formula is valid:*

$$2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 \land 1 = c \land 1 = f \rightarrow \big[\big(\mathsf{if}(x = 0)\, v := -cv; \mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2 \land v < 1) \land v \geq 0)\, v := -fv; t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1\}\big)^*\big](0 \leq x \leq 5) \tag{9.9*}$$

Part of the differential equation, namely $t' = 1$, comes from the controller, because it corresponds to putting a clock on the controller and running it with at least the sampling frequency 1 (coming from the evolution domain constraint $t \leq 1$).

9.3 Summary

This chapter studied time-triggered control, which, together with event-triggered control from Chap. 8, is an important principle for designing feedback mechanisms in CPS and embedded systems. The chapter illustrated the most important aspects for a running example of a ping-pong ball. Despite or maybe even because of its simplicity, the ping-pong ball was an instructive source for the most important subtleties involved with time-triggered control decisions. Getting time-triggered controllers correct requires predictions about how the system state might evolve over short periods of time (one control cycle). The effects and subtleties of time-triggered actions in control were sufficiently subtle to merit focusing on a simple intuitive case.

Unlike event-triggered control, which assumes continuous sensing, the time-triggered control principle is more realistic by only assuming the availability and processing of sensor data at discrete instants of time (discrete sensing). Time-triggered system models avoid the modeling subtleties that events tend to cause for the detection of events. It is, thus, often much easier to get the models right and implementable for time-triggered systems than it is for event-triggered control. The price is that the burden of event-detection is then brought to the attention of the CPS programmer, whose time-triggered controller will now have to ensure it predicts and detects events early enough before it is too late to react to them. That is what makes time-triggered controllers more difficult to get correct, but is also crucial because important aspects of reliable event detection may otherwise be brushed under the rug, which does not help the final CPS become any safer either.

CPS design often begins by pretending the idealized world of event-triggered control (if the controller is not even safe when events are checked and responded to continuously, it is broken already) and then subsequently morphing the event-triggered controller into a time-triggered controller. This second step then often indicates additional subtleties that were missed in the event-triggered design. The additional insights gained in time-triggered controllers are crucial whenever the system reacts slowly or whenever it reacts quickly but needs a high precision in event detection to remain safe. For example, the reaction time for ground control decisions to reach a rover on Mars are so prohibitively large that they could hardly be ignored. Reaction times in a surgical robotics system that is running at, say, 55 Hz, are still crucial even if the system is moving slowly and reacting quickly, because the required precision of the system is in the sub-millimeter range [1]. But reaction times will have less of an impact for parking a slowly moving car somewhere in an empty football stadium.

Overall, the biggest issues with event-triggered control, besides sometimes being hard to implement, are the subtleties involved in properly modeling event detection without accidentally defying the laws of physics in pursuit of an event. But controlling event-triggered systems is reasonably straightforward as long as the events are chosen well. In contrast, finding a model is relatively easy in time-triggered control, but identifying appropriately safe controller constraints takes a lot more thought, leading, however, to important insights about the system at hand. It is possible to provide the best of both worlds by systematically reducing the safety proof of an

(implementable) time-triggered controller to the (easier) safety proof of an event-triggered controller along with corresponding compatibility conditions [2, 3].

Exercises

9.1 (Time bounds). HP (9.3) imposes an upper bound on the duration of a continuous evolution. Can you impose an upper bound 1 and lower bound 0.5? Is there relevant safety-critical behavior in the system that is then no longer considered?

9.2. Give an initial state for which the controller in (9.3) would skip over the event without noticing it.

9.3. What would happen if the controller in (9.7) used the ping-pong paddle while the ball is still on the ground? To what physical phenomenon does that correspond?

9.4. The formula (9.9) with the time-triggered controller with reaction time at most 1 time unit is valid. Yet, if a ball is let loose ever so slightly above ground with a very fast negative velocity, couldn't it possibly bounce back and exceed the safe height 5 faster than the reaction time of 1 time unit? Does that mean the formula ought to have been falsifiable? No! Identify why and give a physical interpretation.

9.5. The controller in (9.9) runs at least once a second. How can you change the model and controller so that it runs at least twice a second? What changes can you make to the controller to reflect that increased frequency? How do you need to change (9.9) if the controller only runs reliably at least once every two seconds? Which of those changes are safety-critical, which are not?

9.6. What happens if we misread the binding precedences and think the condition $v < 1$ is added to both disjuncts in the controller in (9.9)?

$$\begin{aligned}&2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 \land 1 = c \land 1 = f \rightarrow\\ &\big[\big(\mathsf{if}(x = 0)\, v := -cv; \mathsf{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2) \land v < 1 \land v \geq 0)\, v := -fv;\\ &\quad t := 0; \{x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1\}\big)^*\big](0 \leq x \leq 5)\end{aligned}$$

Is the resulting formula still valid? Find an invariant or counterexample.

9.7. Conduct a sequent proof proving the validity of dL formula (9.9). Is it easier to follow a direct proof or is it easier to use the generalization rule MR for the proof?

9.8. The event-triggered controller we designed in Chap. 8 monitored the event $4 \leq x \leq 5$. The time-triggered controller in Sect. 9.2, however, ultimately only took the upper bound 5 into account. How and under what circumstances can you modify the controller so that it really only reacts to the event $4 \leq x \leq 5$ rather than under all circumstances where the ball is in danger of exceeding 5?

9.9. Devise a controller that reacts if the height changes by 1 when comparing the height before the continuous evolution to the height after. Can you make it safe? Can you implement it? Is it an event-triggered or a time-triggered controller? How does it compare to the controllers developed in this chapter?

9.10. The ping-pong ball proof relied on the parameter assumptions $g = c = f = 1$ for mere convenience of the resulting arithmetic. Develop a time-triggered model, controller, invariant, and proof for the general ping-pong ball without these unnecessarily strong simplifying assumptions.

9.11. Show that the ping-pong ball (9.9) can also be proved safe using just the invariant $0 \leq x \leq 5$ (possibly including assumptions on constants such as $g > 0$). Which assumptions on the initial state does this proof crucially depend on?

9.12 (*). Design a variation of the time-triggered controller for the ping-pong ball that is allowed to use the ping-pong paddle within height $4 \leq x \leq 5$ but has a relaxed safety condition that accepts $0 \leq x \leq 2 \cdot 5$. Make sure to only force the use of the ping-pong paddle when necessary. Find an invariant and conduct a proof.

9.13 (2D ping-pong time). Design and verify the safety of a ping-pong controller that goes sideways with horizontal motion like in ordinary ping-pong matches. What is the impact of reaction time?

9.14 (Robot chase). You are in control of a robot tailing another one in hot pursuit. You can accelerate ($a := A$), brake ($a := -b$), or coast ($a := 0$). But so can the robot you're following! Your job is to fill in the blanks with test conditions that make the robots not crash.

$$
\begin{array}{l}
x \leq y \wedge v = 0 \wedge A \geq 0 \wedge b > 0 \rightarrow \\
\quad \big[\big((c := A \cup c := -b \cup c := 0); \\
\qquad (?\underline{\qquad\qquad}; a := A \cup ?\underline{\qquad\qquad}; a := -b \cup ?\underline{\qquad\qquad}; a := 0); \\
\qquad t := 0; \{x' = v, v' = a, y' = w, w' = c, t' = 1 \,\&\, v \geq 0 \wedge w \geq 0 \wedge t \leq \varepsilon\}\big)^*\big] \\
\quad x \leq y
\end{array}
$$

9.15 (Zeno's paradox of Achilles and the Tortoise). Hybrid systems make transparent the two different world models with which Zeno described the race of the fast runner Achilles against the slow Tortoise (Expedition 9.1). Achilles is at position a running with velocity v. The Tortoise is at position t crawling with velocity $w < v$. The model of successive motion uses separate differential equations, where Achilles first moves for duration s till he reaches the position t where the Tortoise was, which already moved on with its smaller velocity w for the same duration:

$$s := 0; \big(\{a' = v, s' = 1 \,\&\, a \leq t\}; ?a = t; \{t' = w, s' = -1 \,\&\, s \geq 0\}; ?s = 0\big)^*$$

Compare this to simultaneous motion in a combined differential equation system:

$$s := 0; \{a' = v, t' = w, s' = 1\}$$

Show that Achilles a will never reach Tortoise t in the first model despite $v > w$ if $a < t$ holds initially. For the second model prove that postcondition $a = t$ will eventually be true (with the help of a diamond modality). Then contrast both models with what happens when another Greek philosopher stumbles upon the race track, distracting Achilles with questions about other paradoxical models of motion.

References

[1] Yanni Kouskoulas, David W. Renshaw, André Platzer, and Peter Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In: *HSCC*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, 263–272. DOI: 10.1145/2461328.2461369.

[2] Sarah M. Loos. Differential Refinement Logic. PhD thesis. Computer Science Department, School of Computer Science, Carnegie Mellon University, 2016.

[3] Sarah M. Loos and André Platzer. Differential refinement logic. In: *LICS*. Ed. by Martin Grohe, Eric Koskinen, and Natarajan Shankar. New York: ACM, 2016, 505–514. DOI: 10.1145/2933575.2934555.