André Platzer

Lecture Notes on Foundations of Cyber-Physical Systems

15-424/624/824 Foundations of Cyber-Physical Systems

Chapter 4
Safety & Contracts

Synposis This chapter provides a lightweight introduction to safety specification techniques for cyber-physical systems. It briefly discusses how program contracts generalize to CPS for declaring expectations on the initial states together with guarantees for the possible final states of a CPS model. Since assumptions and guarantees can be quite subtle for CPS applications, it is important to capture them early during a CPS design. This chapter introduces *differential dynamic logic*, a logic for specifying and verifying hybrid systems, which provides a formal underpinning for a precise meaning of CPS contracts. In subsequent chapters, differential dynamic logic plays a central role for rigorous verification as well. This chapter develops the running example of Quantum the bouncing ball, which is a hopelessly impoverished CPS but still features many of the important dynamical aspects of CPS in a very intuitive setting.

4.1 Introduction

In the previous chapters, we have studied models of cyber-physical systems and discovered hybrid programs as their programming language [22, 23, 26, 30]. Its most unique features are differential equations and nondeterminism alongside the usual classical control structures and discrete assignments. Together, these features provide powerful and flexible ways of modeling even very challenging systems and very complex control principles. This chapter will start studying ways of making sure that the resulting behavior, however flexible and powerful it may be, also meets the required safety and correctness standards.

In case you have already experienced contracts in conventional discrete programming languages, you will have observed how they make properties and input requirements of programs explicit. You will probably have seen how contracts can be checked dynamically at runtime, which, if they fail, will alert you right away to flaws in the design of the programs. In that case, you have experienced first hand that it is much easier to find and fix problems in programs starting from the first con-

tract that failed in the middle of the program, rather than from the mere observation about the symptoms that ultimately surface when the final output is not as expected. In particular, unless you check the output dynamically with a contract or manually every time, you may not even notice that something is wrong.

Another aspect of contracts that you may or may not yet have had the opportunity to observe is that they can be used in proofs, which show that *every* program run will satisfy the contracts. Every time the requirements hold for the input, the output will promise to meet its guarantees. Unlike in dynamic checking, the scope of correctness arguments with proofs extends far beyond the test cases that have been tried, however clever the tests may have been chosen. After all, testing can only show the presence of bugs, never quite their absence [32, 40]. Both uses of contracts, dynamic checking and rigorous proofs, are very helpful to check whether a system does what we intend it to, as has been argued repeatedly in the literature [5, 14, 18, 19, 34, 36, 39].

The principles of contracts help cyber-physical systems [3, 22, 23, 28] as well. Yet, their use in proving may, arguably, be more important than their use in dynamic checking. The reason has to do with the physical impact of CPS and the non-negotiability of the laws of physics. The reader is advised to imagine a situation where a self-driving car is propelling him or her down the street. Suppose the car's control software is covered with contracts all over, but all of them are exclusively for dynamic checking, none have been proved. If that self-driving car speeds up to 100mph on a 55mph highway and drives up very close to a car in front of it, then dynamically checking the contract "keep at least a distance of 1 meter to the car in front" no longer helps. If that contract fails, the car's software would know that it made a mistake, but it has become too late to do anything about it, because the brakes of the car cannot slow the car down quickly enough. The car would be "trapped in its own physics", in the sense that it has run out of all safe control options and can only brace itself for impact. While there are effective ways of making use of dynamic contract checking in CPS [20], the design of those contracts requires proof to ensure that they respond early enough and safety is always maintained.

For those reasons, this textbook will focus on the role of proofs as correctness arguments for CPS contracts much more than on their use in dynamical checking. Because of the physical consequences of malfunctions, correctness requirements on CPS are also more stringent. The subtle nuances in their behavior may require significantly more challenging arguments than, e.g., array-bounds checking for classical programs. For those reasons, we will approach CPS proofs with a fair amount of rigor. But such rigorous reasoning is already a story for a later chapter.

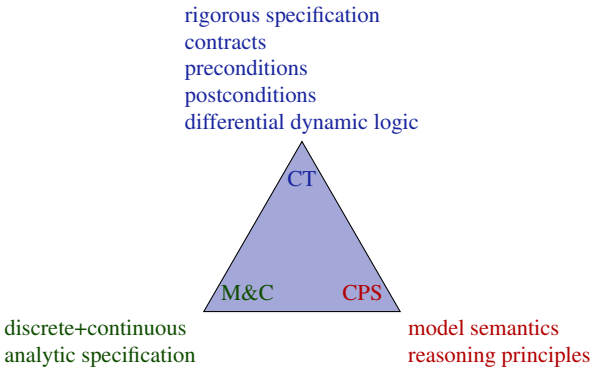
The focus of this chapter will first be to understand CPS contracts themselves. As a useful modeling and specification exercise, we will develop a model of a bouncing ball and identify all requirements for it to be safe. Along the way, however, this chapter develops an intuitive understanding for the role of requirements and contracts in CPS as well as important ways of formalizing CPS properties and their analyzes. The material in this chapter provides a more intuitive gradual introduction into correctness specification techniques for CPS [21–23, 26, 30].

The most important learning goals of this chapter are:

Modeling and Control: We deepen our understanding of the core principles behind CPS by internalizing discrete and continuous aspects of CPS in logical formulas, which is a crucial stepping stone toward analytic reasoning principles.

Computational Thinking: We develop an example that is equally simple and instructive to learn how to identify specifications and critical properties of CPS. Even if the example we look at, the bouncing ball, is a hopelessly impoverished CPS, it still formidably conveys the subtleties involved with hybrid systems models, which are crucial for understanding CPS. This chapter is devoted to contracts in the form of pre- and post-conditions for CPS models. We will begin to rigorously specify our requirements and expectations on CPS models, which is critical to getting CPS right. In order to enable mathematically rigorous and unambiguous specifications, this chapter introduces *differential dynamic logic* dL [22, 23, 26, 30] as the specification and verification language for CPS that we will be using throughout this textbook.

CPS Skills: We will begin to deepen our understanding of the semantics of CPS models by relating it to their reasoning principles. A full study of this alignment will only be covered in the next chapter, though.



4.2 A Gradual Introduction to CPS Contracts

This section provides a gradual and informal introduction into contracts for cyber-physical systems. Its focus is on an intuitive development of the need for contracts, which provides the motivation for a subsequent rigorous development of a logic for CPS contracts in which every aspect has an unambiguously well-defined meaning. This section also introduces the running example of Quantum the bouncing ball, which will be with us as a simple intuitive yet surprisingly representative example throughout the book.

4.2.1 The Adventures of Quantum the Bouncing Ball

A model for accelerated motion along a straight line with a choice of increasing acceleration or braking was considered in Sect. 3.2. That model did perform interesting control choices and we could continue to study it in this chapter. In order to sharpen our intuition about CPS, we will, however, prefer to study a very simple but also very intuitive system instead.

Once upon a time, there was a little bouncing ball called *Quantum*. Day in, day out, Quantum had nothing else to do but bounce up and down the street until he was tired of doing even that, which, in fact, rarely happened, because bouncing was such a joy for him (Fig. 8.1). Quantum the bouncing ball was not much of a CPS, because Quantum does not actually have any interesting decisions to make. At least, Quantum was quite content without having to face any decisions until he discovers in Chap. 8 how empowering, subtle, and intriguing they can be. For one thing, Quantum did not even bring his computer, os he already lacked one simple indicator of qualifying as a cyber-physical system.

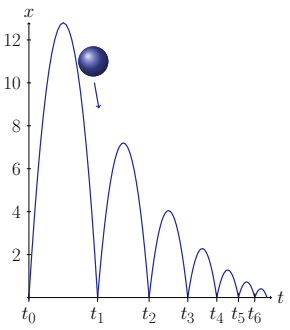


Fig. 4.1 Sample trajectory of a bouncing ball (plotted as height over time)

But Quantum nevertheless formed a perfectly reasonable hybrid system, because, after a closer look, the system turns out to involve discrete and continuous dynamics. The continuous dynamics is caused by gravity, which is pulling the ball down and makes it fall down from the sky in the first place. The discrete dynamics comes from the singular discrete event of what happens when the ball hits the ground and bounces back up. There are a number of ways of modeling the ball and its impact on the ground with physics. They include a whole range of different, and either less or more realistic, physical effects such as gravity, aerodynamic resistance, the elastic deformation on the ground, and so on and so on. But the little bouncing ball, Quantum, didn't study enough physics to know anything about those effects. And so Quantum had to go about understanding the world in easier terms. Quantum was a clever bouncing ball, though, so it had also experienced the phenomenon of sudden change and was trying to use that to his advantage.

When looking for a very simple model of what the bouncing ball does, it is easier to describe as a hybrid system. The ball at height x is falling subject to gravity g:

$$x'' = -g$$

When it hits the ground, which is assumed at height $x = 0$, the ball bounces back and jumps back up in the air. Yet, as every child knows, the ball tends to come back up a little less high than before. Given enough time to bounce around, it will ultimately even lie flat on the ground forever. Until it is picked up again and thrown high up in the air. Quantum was no stranger to this common experience on the physical experience of bouncing.

So Quantum went ahead to model the impact on the ground as a discrete phenomenon and sought ways of describing what happens to make the ball jump back up. One attempt of understanding this could be to make the ball jump back up rather suddenly by increasing its height x by, say, 10 when the ball hits the ground $x = 0$:

$$\begin{aligned} &x'' = -g; \\ &\mathsf{if}(x = 0)\, x := x + 10 \end{aligned} \tag{4.1}$$

This HP first follows the differential equation in the first line continuously for some time and then, after the sequential composition (;), performs the discrete computation in the second line to increase x by 10 if it is on the ground $x = 0$. Such a model may be useful to describe other systems, but would be rather at odds with our physical experience with bouncing balls, because the ball is indeed slowly climbing back up rather than suddenly starting out way up in the air again.

Quantum ponders about what happens when he hits the ground. Quantum does not suddenly get teleported to a new position above ground like HP (4.1) would suggest. Instead, the ball suddenly changes its direction but not position. A moment ago, Quantum used to fall down with a negative velocity (i.e. one that is pointing down into the ground) and then, all of a sudden, climbs back up with a positive velocity (pointing up into the sky). In order to be able to write such a model, the velocity v will be made explicit in the bouncing ball's differential equation system:

$$\begin{aligned} &\{x' = v, v' = -g\}; \\ &\mathsf{if}(x = 0)\, v := -v \end{aligned} \tag{4.2}$$

Of course, something also happens after the bouncing ball reversed its direction because it hit the ground. Physics continues until it hits the ground again:

$$\begin{aligned} &\{x' = v, v' = -g\}; \\ &\mathsf{if}(x = 0)\, v := -v; \\ &\{x' = v, v' = -g\}; \\ &\mathsf{if}(x = 0)\, v := -v \end{aligned} \tag{4.3}$$

Then, of course, physics moves on again, so the model actually involves a repetition:

$$\begin{aligned} &\big(\{x' = v, v' = -g\}; \\ &\quad \mathsf{if}(x = 0)\, v := -v\big)^* \end{aligned} \tag{4.4}$$

It is good that repetitions (*) in HP are nondeterministic, because Quantum has no way of knowing how many iterations of the control loop he will take. Yet, Quantum is now rather surprised. For if he follows that HP (4.4), it seems as if he should always be able to climb back up to his initial height again. Excited about that possibility, Quantum tries and tries again, but he never succeeds to bounce back up as high as he was before. So there must be something wrong with the model in (4.4), Quantum concludes and sets out to fix (4.4).

Having observed himself rather carefully when bouncing around, Quantum concludes that he feels just a little bit slower when bouncing back up than he used to be when falling down. Indeed, Quantum feels less energetic on his way up. So his velocity must not only flip direction from down to up at a bounce, but also seems to shrink in magnitude. Quantum swiftly calls the corresponding damping factor c and quickly comes up with a better model of himself:

$$\begin{array}{l}\big(\{x' = v, v' = -g\};\\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^*\end{array} \tag{4.5}$$

Yet, running that model in clever ways, Quantum observes that model (4.5) could make him fall through the cracks in the ground. Terrified at that thought, Quantum quickly sets the physics right, lest he falls through the cracks in space before he had a chance to fix his own physics. The issue with (4.5) is that its differential equation isn't told when to stop, so it could evolve for too long a time below ground and just fail the subsequent test $x = 0$ as shown in Fig. 4.2. Yet, Quantum luckily remembers

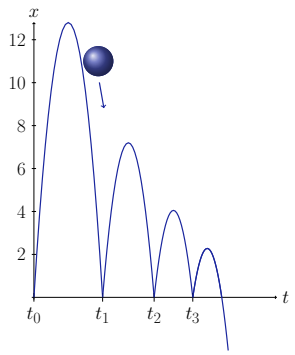


Fig. 4.2 Sample trajectory of a bouncing ball (plotted as height over time) with a crack in the floor

from Chap. 3 that this purpose is quite exactly what evolution domains were meant for. Above ground is where he wants to remain, and so $x \geq 0$ is what Quantum asks dear physics to obey, since the floor underneath Quantum is of rather sturdy built. Unlike in poor Alice's case [2], the floor comes without rabbit holes to fall through:

$$\begin{array}{l}\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^*\end{array} \tag{4.6}$$

Now, indeed, physics will have to stop evolving before gravity has made our little bouncing ball Quantum fall through the ground. Yet, physics could still choose to stop evolving while the ball is still high up in the sky. In that case, the ball will not yet be on the ground and line 2 of (4.6) would have no effect because $x \neq 0$ still. This is not a catastrophe, however, because the loop in (4.6) could simply repeat, which would allow physics to continue to evolve along the same differential equation just a little bit further. Thankfully, the evolution domain constraint $\ldots \& x \geq 0$ prevents all ill-fated attempts of following the differential equation below ground.

Being quite happy with model (4.6), the bouncing ball Quantum goes on to explore whether the model does what he expects it to do. Of course, had Quantum read this book already, he would have marched right into a rigorous analysis of the model. Since Quantum is still a CPS rookie, he takes a detour along visualization road, and first shoots a couple of pictures of what happens when simulating model (4.6). Thanks to a really good simulator, these simulations all came out looking characteristically similar to Fig. 8.1.

4.2.2 How Quantum Discovered a Crack in the Fabric of Time

After a little while of idly simulating is very own personal model, Quantum decides to take out his temporal magnifying glasses and zoom in real close to see what happens when his model (4.6) bounces on the ground ($x = 0$). At that point in time, the differential equation is forced to stop due to the evolution domain $x \geq 0$, so the continuous evolution stops and a discrete action happens that inspects the height and, if $x = 0$, discretely changes the velocity to $-cv$ instantly in no time.

At the continuous point in time of the first bounce—Quantum recorded the time t_1—the ball observes a succession of different states. First at continuous time t_1, Quantum has position $x = 0$ and velocity $v = -5$. But then, after the discrete assignment of (4.6) ran, still at real time t_1, it has position $x = 0$ and velocity $v = 4$. This temporal chaos cannot possibly go on like that, thought Quantum, and decided to give an extra natural number index $j \in \mathbb{N}$ to distinguish the two successive occurrences of continuous time t_1. So, for the sake of illustration, he called $(t_1, 0)$ the first point in time where Quantum was in state $x = 0, v = -5$, and then called $(t_1, 1)$ the second point in time where he was in state $x = 0, v = 4$.

In fact, Quantum's temporal magnifying glasses worked so well that he suddenly discovered he had accidentally invented an extra dimension for time: the discrete time step $i \in \mathbb{N}$ in addition to the continuous time coordinate $t \in \mathbb{R}$. Quantum plots the continuous $\mathbb{R}$-valued time coordinate in the t axis of Fig. 4.3 while separating the $\mathbb{N}$-valued discrete step count of the hybrid time into the j axis and leaving the x axis for position. In Fig. 4.3, Quantum now observed the first simulation of model (4.6) with his temporal magnifiers activated to fully appreciate its hybrid nature in its full blossom. And, indeed, if Quantum looks at the hybrid time simulation from Fig. 4.3 and turns his temporal magnifiers off again, the extra dimension of discrete steps j vanishes again, leaving behind only the shadow of the execution in the x over

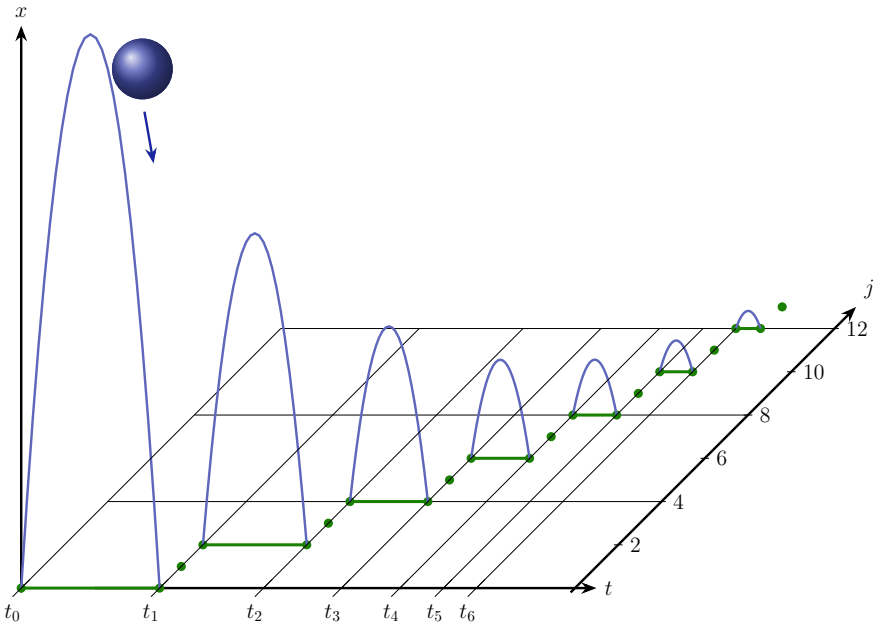


Fig. 4.3 Sample trajectory of a bouncing ball plotted as position x over its hybrid time domain with discrete time step j and continuous time t

t face, which agrees with the layman's simulation shown in Fig. 8.1. And even the projection of the hybrid time simulation from Fig. 4.3 to the j over t time face leads to a curious illustration, shown in Fig. 4.4, of what the temporal magnifying glasses revealed about how hybrid time has evolved in this particular simulation.

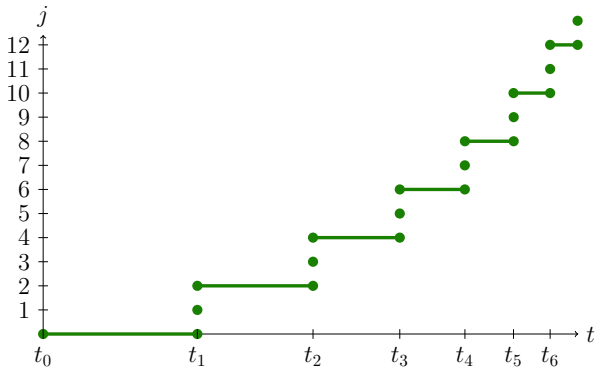


Fig. 4.4 Hybrid time domain for the sample trajectory of a bouncing ball with discrete time step j and continuous time t

Armed with the additional intuition about the operations of HP (4.6) that these sample executions in Fig. 8.1 and its hybrid version Fig. 4.3 provide, Quantum now feels prepared to ask the deeper questions. Will his model (4.6) *always* do the right thing? Or was he just lucky in the particular simulation shown in Fig. 8.1? What even is the right behavior for a proper bouncing ball? What are the important properties? And for what purpose? How could they be specified unambiguously? And, ultimately, how could Quantum convince himself about these properties being true before he bounces around idly and in potential risk of harm?

4.2.3 How Quantum Learned to Deflate

While half of Quantum's mind is already in hot pursuit of the pressing questions of correctness and personal safety he raised in the previous section, the other half is still wondering what went wrong with his model if it did not even know how to lie still any more. Scrutinizing each and every simulation (Figs. 8.1 and 4.3) of the bouncing ball's model from every angle of time again, he learns that every which way balls bounce according to HP (4.6), they do not ever seem to stop bouncing. Bewildered about this distinct possibility, Quantum gave it a try and bounced around like he had never bounced before. After a while of unsuccessful attempts, he lay down to think, realizing that something must be rather counterfactual about the model if it seriously predicted he would always bounce, when, in reality, Quantum ultimately runs out of steam and just lies flat on his back as in Fig. 4.5.

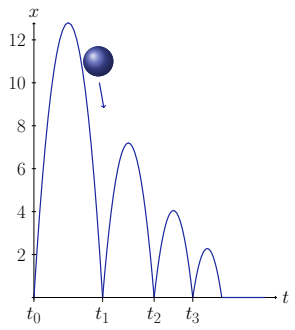


Fig. 4.5 Sample trajectory of a bouncing ball (plotted as height over time) that ultimately lies down flat

If Quantum wants to describe reality, the model (4.6) needs some fixing. Halfway through his sophisticated development of better models with increasingly high-fidelity mixes of elastic and plastic deformation of balls at a bounce, friction, and the role of energy loss, Quantum suddenly has a clever idea. Of course, all these models of deformations and frictions and energies would be needed for a model of highly precise physics. Yet, if Quantum is merely trying to describe the qualitative behavior of a bouncing ball, he might as well try to leverage the power of abstraction that he read about in Chap. 3. On a big ideas level, when Quantum bounces on

the ground, he either bounces back up with reduced velocity $v := -cv$ or he just lies flat with no velocity at all $v := 0$. Figuring out exactly which case happens when would put Quantum back into the mode of describing increasingly precise physics and measuring all kinds of specific coefficients and parameters in those models. But just describing the fact that one of the two options can happen when on the ground is quite easy with a nondeterministic choice ($\cup$):

$$\begin{aligned}&\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\\ &\quad \mathsf{if}(x = 0)\,(v := -cv \cup v := 0)\big)^*\end{aligned} \tag{4.7}$$

This new and improved HP now allows with reasonable accuracy all behavior that Quantum observes when trying to bounce around. The model also allows some extra behavior that he never quite got physics to do for him, such as never stopping to bounce around, so (4.7) is an overapproximation. Comparing notes with his high-fidelity physics models, though, Quantum cannot help but appreciate the relative simplicity of (4.7) and would much rather seek to analyze the simpler hybrid systems model (4.7) than to try an analysis of a more precise but also significantly more difficult physical model. Quantum now saw first hand how abstraction can create simplicity.

Speaking of simplicity. Even if Quantum now appreciates HP (4.7) as the better model compared to the simpler HP (4.6) on account of giving the physics some options to have the ball lie flat after a bounce, Quantum still prefers to first investigate the simpler model (4.6) for the pressing correctness questions that now have Quantum's full attention. It is also a distinct possibility that Quantum simply took a peek at Exercise 4.15, in which the more involved model (4.7) will be considered.

4.2.4 Postcondition Contracts for CPS

Hybrid programs α are interesting models for CPS. They describe the behavior of a CPS, ultimately captured by their semantics $[\![\alpha]\!]$, which is a reachability relation on states (Chap. 3). Yet, reliable development of CPS also needs a way of ensuring that this behavior will be as expected. For one thing, we want the behavior of a CPS to always satisfy certain crucial safety properties. A robot, for example, should never do anything unsafe[1] like running over a human being.

[1] Safety of robots has, of course, been aptly defined by Asimov's Three Laws of Robotics [1]:

① A robot may not injure a human being or, through inaction, allow a human being to come to harm.

② A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.

③ A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Sadly, their exact rendition in logic or anything similarly precise still remains quite a challenge due to language ambiguities and similar minor nuisances that kept scientists busy for the good deal of a century since. The Three Laws of Robotics are not the answer. They are the inspiration!

Even if Quantum, the little bouncing ball, may be less safety-critical, he is still quite interested in his own safety. Quantum still wants to make sure that he couldn't ever fall through the cracks in the ground. And even though he would love to jump all the way up to the moon, Quantum turns out to be rather terrified of big heights. He would never want to jump any higher than he was in the very beginning. So, when H denotes the initial height, Quantum really wants to know whether his height will always stay within $0 \leq x \leq H$ when following HP (4.6).

Scared of what otherwise might happen to him if $0 \leq x \leq H$ should ever be violated, Quantum decides to make his goals for the HP (4.6) explicit. Fortunately, Quantum excelled in basic programming courses, where contracts have been used to make behavioral expectations for programs explicit. Even though Quantum clearly no longer deals with plain conventional programs, but rather a hybrid program, Quantum still decides to put an **ensures**(F) contract in front of HP (4.6) to express that all runs of that HP are expected to lead only to states in which logical formula F is true. Quantum even uses two postconditions, once for each of his expectations. For the time being, Quantum temporarily uses the following notation to indicate the two expected postconditions for its bouncing ball HP:

$$\begin{array}{l} \textbf{ensures}(0 \leq x) \\ \textbf{ensures}(x \leq H) \\ \big(\{x' = v, v' = -g \,\&\, x \geq 0\}; \\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^* \end{array} \tag{4.8}$$

Subsequent sections will quickly abandon this **ensures**(F) notation in favor of a more logical approach. But for now, Quantum is quite happy with documenting what his model is expected to achieve.

4.2.5 Precondition Contracts for CPS

Having read up a lot about conventional program contracts, Quantum immediately begins to wonder whether the **ensures**() contracts in HP (4.8) would, in fact, always be true after running that HP. After all, acrophobic Quantum would really like to rely on this contract never failing. In fact, he prefers to see that logical contract met before he dares trying another careless bounce ever again.

Quantum's deliberations eventually lead him to conclude that it would very well depend on the initial values that the bouncing ball starts out with, whether the **ensures**() contract in (4.8) works out. Quantum thought of H as the initial height, but the HP (4.8) cannot know that. Indeed, the contracts would be rather hard to fulfill if $H = -5$, because $0 \leq x$ and $x \leq H$ could not possibly both be true then.

So, Quantum decides he should demand a **requires**$(x = H)$ contract with the precondition $x = H$ to say that the height, x, of the bouncing ball is initially H.

Since that still does not ensure that $0 \leq x$ has a chance of holding, Quantum requires $0 \leq H$ to hold initially as well, leading to:

$$\begin{array}{l} \textbf{requires}(x = H) \\ \textbf{requires}(0 \leq H) \\ \textbf{ensures}(0 \leq x) \\ \textbf{ensures}(x \leq H) \\ \big(\{x' = v, v' = -g \,\&\, x \geq 0\}; \\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^* \end{array} \tag{4.9}$$

Expedition 4.1 (Invariant contracts for CPS)

In addition to preconditions and postconditions, loop invariants play a prominent role in contracts for conventional imperative programs. Preconditions state what is expected to hold before the program runs. Postconditions state what is guaranteed to hold after the program runs. And loop invariants indicate what is true every time the loop body executes, so before and after every run of the loop body. In C-style programs, for example, invariants are associated with loops:

```
i = 0;
while (i < 10)
  // loop_invariant(0 <= i && i <= 10)
  {
    i++;
  }
```

Such loop invariants will also play an equally important role in CPS (Chap. 7), but they first require additional developments to become meaningful.

4.3 Logical Formulas for Hybrid Programs

CPS contracts play a very useful role in the development of CPS programs or, in fact, any other CPS models. Using them as part of their design right from the very beginning is a good idea, probably much more crucial than it is when developing conventional programs, because CPS have more stringent requirements on safety.

Yet, we do not only want to program CPS, we also want to and will have to understand thoroughly what CPS programs and their contracts mean, and how we convince ourselves that the CPS contracts are respected by the CPS program. This is where mere contracts are at a disadvantage compared to the full features of logic.

Note 19 (Logic is for specification and reasoning) *Logic allows not only the specification of a whole CPS program, but also an analytic inspection of its parts as well as argumentative relations between contracts and program parts.*

Logic has originally been invented for precise statements, justifications, and ways of systematizing rational human thought and mathematical reasoning [6, 8–13, 16, 37, 38]. Logic saw influential generalizations to enable precise statements and reasoning about conventional discrete programs [5, 14, 34], and other aspects, including modes of truth such as necessity and possibility [15] or temporal relations of truth [33, 35].

What cyber-physical systems need, though, is a logic for precise statements and reasoning about their dynamical systems, so they need logics of dynamical systems [26], of which the most fundamental representative is *differential dynamic logic* (dL) [21–23, 26, 27, 30], the logic of hybrid systems. Differential dynamic logic allows direct logical statements about hybrid programs and, thus, serves as the logic of CPS programs in Parts I and II of this textbook for both specification and verification purposes. Additional multi-dynamical systems aspects beyond hybrid systems are discussed elsewhere [24–26, 29, 31], some of which will be picked up in later parts of this textbook.

The most unique feature of differential dynamic logic for our purposes is that it allows us to refer to hybrid systems. Chap. 2 introduced first-order logic of real arithmetic, which was used to describe evolution domain constraints of differential equations, and made it possible to refer to conjunctions or disjunctions of comparisons of (polynomial) terms as well as quantifiers over real-valued variables.

Note 20 (Limits of first-order logic for CPS) *First-order logic of real arithmetic is a crucial basis for describing what is true and false about CPS, because it allows us to refer to real-valued quantities like positions and velocities and their arithmetic relations. Yet, that is not quite enough, because first-order logic describes what is true in a single state of a system. It has no way of referring to what will be true in future states of a CPS, nor of describing the relationship of the initial state of the CPS to the final state of the CPS. Without such a capability, it is impossible to refer to what preconditions were true before the CPS started and how this relates to what postconditions are true afterwards.*

Recall from Sect. 3.3.2 that this relation, $[\![\alpha]\!] \subseteq \mathcal{S} \times \mathcal{S}$, is what ultimately constitutes the semantics of HP α. It defines which new state $\nu \in \mathcal{S}$ is reachable from which initial state $\omega \in \mathcal{S}$ in HP α, in which case we write $(\omega, \nu) \in [\![\alpha]\!]$.

Note 21 (Differential dynamic logic principle) *Differential dynamic logic (denoted dL) extends first-order logic of real arithmetic with operators that refer to the future states of a CPS, that is to the states that are reachable by running*

a given HP. The logic dL provides a modal operator $[\alpha]$, parametrized by HP α, that refers to all states reachable by this HP α according to the reachability relation $[\![\alpha]\!]$ of its semantics. This modal operator can be placed in front of any dL formula P. The dL formula

$$[\alpha]P$$

expresses that all *states reachable by HP α satisfy formula P.*

The logic dL also provides another modal operator $\langle\alpha\rangle$, parametrized by HP α, that can be placed in front of any dL formula P. The dL formula

$$\langle\alpha\rangle P$$

expresses that there is at least one *state reachable by HP α for which P holds. The modalities $[\alpha]$ and $\langle\alpha\rangle$can be used to express necessary or possible properties of the transition behavior of α, since they refer to all or some runs of α. The formula $[\alpha]P$ is pronounced "α box P" and $\langle\alpha\rangle P$ is "α diamond P."*

An **ensures**(E) postcondition for a HP α can be expressed directly as a logical formula in differential dynamic logic:

$$[\alpha]E$$

So, the first CPS postcondition **ensures**$(0 \leq x)$ for the bouncing ball HP in (4.8) can be stated as a dL formula:

$$[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*]\, 0 \leq x \tag{4.10}$$

The second CPS postcondition **ensures**$(x \leq H)$ for the bouncing ball HP in (4.8) can be stated as a dL formula as well:

$$[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*]\, x \leq H \tag{4.11}$$

The logic dL allows all other logical operators from first-order logic, including conjunction ($\wedge$). So, the two dL formulas (4.10) and (4.11) can be stated together as a single dL formula:

$$\begin{aligned}&[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*]\, 0 \leq x \\ \wedge\ &[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*]\, x \leq H\end{aligned} \tag{4.12}$$

Stepping back, we could have combined the two postconditions **ensures**$(0 \leq x)$ and **ensures**$(x \leq H)$ into a single postcondition **ensures**$(0 \leq x \wedge x \leq H)$ using a conjunction in the postcondition instead. The translation of that into dL would have gotten us an alternative way of combining both statements about the lower and upper bound on the height of the bouncing ball into a single dL formula:

$$[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*]\, (0 \leq x \wedge x \leq H) \tag{4.13}$$

Which way of representing what we expect bouncing balls to do is better? Like (4.12) or like (4.13)? Are they equivalent? Or do they express different things?

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

There is a very simple argument within the logic dL that shows that dL formulas (4.12) and (4.13) are equivalent. It even shows that the same equivalence holds not just for these particular formulas but for any dL formulas of the same form:

$$[\alpha]P \wedge [\alpha]Q \text{ is equivalent to } [\alpha](P \wedge Q) \tag{4.14}$$

This equivalence will be investigated in more detail in a later chapter, but it is useful to observe now already in order to sharpen our intuition about dL.

Having said that, do we believe dL formula (4.12) should be valid so true in all states? Should (4.13) be valid? Well,they should certainly either agree to both be valid or agree to both not be valid since they are equivalent by equivalence (4.14). But is (4.12) valid now or is it not? Before we study this question in any further detail, the first question should be what it means for a modal formula $[\alpha]P$ to be true. What is its semantics? Better yet, what exactly is its syntax in the first place?

4.4 Differential Dynamic Logic

Based on the above informal introduction, this section defines *differential dynamic logic* [21–23, 26, 27, 30], which plays a central role as an unambiguous notation and our basis for rigorous reasoning techniques for CPS throughout this book. Differential dynamic logic uses the terms from Sect. 2.6.2 and hybrid programs from Sect. 3.3.1.

4.4.1 Syntax of Differential Dynamic Logic

The formulas of differential dynamic logic are defined like the formulas of first-order logic of real arithmetic (Sect. 2.6.3) with the additional capability of using modal operators $[\alpha]$ and $\langle\alpha\rangle$ for any hybrid program α. The dL formula $[\alpha]P$ expresses that all states after all runs of HP α satisfy dL formula P. The dL formula $\langle\alpha\rangle P$ expresses that there is a run of HP α that leads to a state in which dL formula P is true.

Definition 4.1 (dL formula). The *formulas of differential dynamic logic* (dL) are defined by the following grammar (where P, Q are dL formulas, $e, \tilde{e}$ are (polynomial) terms, x is a variable, and α is an HP):

$$P,Q ::= e = \tilde{e} \mid e \geq \tilde{e} \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid \forall x P \mid \exists x P \mid [\alpha]P \mid \langle\alpha\rangle P$$

Operators $>, \leq, <, \leftrightarrow$ are definable as usual, e.g., $P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$.

Of course, all first-order real arithmetic formulas from Chap. 2 are also dL formulas and will mean exactly the same. Occasionally, we will use the backwards implication $P \leftarrow Q$ which is just alternative notation for the converse forward implication $Q \rightarrow P$. The dL formula $[x:=5]x > 0$ expresses that x is always positive after assigning 5 to it, which is quite trivially true. The formula $[x:=x+1]x > 0$ expresses that x is always positive after incrementing it by one in a discrete change, which is only *true* in some states but *false* in others whose value of x is too small. With an extra implication, the dL formula $x \geq 0 \rightarrow [x:=x+1]x > 0$ is valid, so true in all states, though, because x will certainly be positive after an increment if it was nonnegative initially. This implication expresses that in all states satisfying the left-hand side assumption $x \geq 0$, the right-hand side $[x:=x+1]x > 0$ is true, which says that after incrementing x, it will have become positive. The implication is trivially true in all states falsifying the assumption $x \geq 0$.

The program $x:=x+1$ can only be run in exactly one way. More interesting things happen for other HPs. The formula $[x:=0;(x:=x+1)^*]x \geq 0$ is valid, since incrementing x any number of times after assigning 0 to x will still yield a nonnegative number. The conjunction $[x:=x+1]x > 0 \wedge [x:=x-1]x < 0$ is *true* in an initial state where x is 0.5, but is not valid, because it is *false* when x starts out at -10.

Likewise, the dL formula $[x' = 2]x > 0$ is not valid, because it is *false* in initial states with negative x values. But $x > 0 \rightarrow [x' = 2]x > 0$ is valid, since x will always increase along the differential equation $x' = 2$. Similarly, $[x:=1;x' = 2]x > 0$ is valid, because x remains positive along ODE $x' = 2$ after first assigning 1 to x.

Quantifiers and modalities can be mixed as well. For example, the dL formula $x > 0 \rightarrow \exists d\,[x' = d]x > 0$ is valid, since if x starts positive, there is a value for d, e.g., 2, which will always keep x positive along $x' = d$. The formula $\exists x \exists d\,[x' = d]x > 0$ is valid, since there is an initial value for x, namely 1 and a value for d, namely 2, such that x stays positive always after any continuous evolution along $x' = d$ starting from that initial value. Even a conjunction $\exists x \exists d\,[x' = d]x > 0 \wedge \exists x \exists d\,[x' = d]x < 0$ is valid, because there indeed is an initial value for x and slope d such that x always stays positive along $x' = d$ for those choices, but there also is another initial value for x, namely -1, and another d, namely -2, such that x always stays negative along $x' = d$. Universal quantifiers and modalities combine as well. For example, $\forall x\,[x:=x^2;x' = 2]x \geq 0$ is valid, because for all values of x, after assigning the (nonnegative) square of x to x, following the differential equation $x' = 2$ for any amount of time will keep x nonnegative.

The box modality $[\alpha]$ in the dL formula $[\alpha]P$ expresses that its *postcondition* P is true after all runs of HP α. By contrast, the diamond modality $\langle\alpha\rangle$ in the dL formula $\langle\alpha\rangle P$ expresses that its postcondition P is true after at least one run of HP α. Even if the diamond modality is not that important in the earlier parts of this book, some examples are already discussed here.

For example, dL formula $\langle x' = 2\rangle x > 0$ is valid, since whatever initial value x has, it will ultimately be positive at some point after just following the differential

equation $x' = 2$ for long enough. The dL formula $\langle x' = d \rangle x > 0$ is not valid, but it is at least true in an initial state whose d value is 2. In particular, $\exists d \langle x' = d \rangle x > 0$ is valid, because there is a (positive) choice of d for which x will eventually be positive after following the ODE $x' = d$ long enough. Even the following conjunction is valid $\exists d \langle x' = d \rangle x > 0 \land \exists x \exists d [x' = d] x < 0$, because the first conjunct is true by the previous argument and the second conjunct is true since a possibly different choice for the initial value of x will always keep x negative along ODE $x' = d$ with an appropriate (and different) choice of d such as 2.

Modalities can also be nested, because any dL formula can be used as a postcondition. For example, dL formula $x > 0 \rightarrow [x' = 2]\langle x' = -2 \rangle x < 0$ is valid, because if x starts positive, then no matter how long one follows the differential equation $x' = 2$, there is a way of subsequently following the differential equation $x' = -2$ for some amount of time such that x becomes negative again. In fact, for the same reason, even the following dL formula is valid:

$$x > 0 \rightarrow [x' = 2](x > 0 \land \langle x' = -2 \rangle x = 0)$$

Beware that equality signs can occur in differential dynamic logic formulas and their hybrid programs in different rôles now:

Expression	Rôle
$x := e$	Discrete assignment in HP assigning value of e to variable x
$x' = f(x)$	Differential equation in HP for continuous evolution of variable x
$x = e$	Equality comparison in dL formula that can be true or false
$?x = e$	Testing for equality comparison in HP, only continue if true

4.4.2 Semantics of Differential Dynamic Logic

For dL formulas that are also formulas of first-order real arithmetic (i.e. formulas without modalities), the semantics of dL formulas is the same as that of first-order real arithmetic. The semantics in Chap. 2 inductively defined the satisfaction relation $\omega \models P$ which formula P is true in which state ω, and then collected the set of states $[\![P]\!]$ in which P is true. Now, we define the semantics of dL formulas right away by simultaneously defining the set of states $[\![P]\!]$ in which formula P is true. Both styles of definitions are equivalent (Exercise 4.12), but the latter is more convenient.

The semantics of modalities $[\alpha]$ and $\langle \alpha \rangle$ quantifies over all (for the box modality $[\alpha]$) or some (for the diamond modality $\langle \alpha \rangle$) of the (final) states reachable by following HP α, respectively.

Definition 4.2 (dL semantics). The *semantics* of a dL formula P is the set of states $[\![P]\!] \subseteq \mathcal{S}$ in which P is true, and is defined inductively as follows:

Expedition 4.2 (Operator precedence for differential dynamic logic)

To save parentheses, the notational conventions have unary operators (including[a] $\neg$, quantifiers $\forall x, \exists x$, modalities $[\alpha], \langle\alpha\rangle$ as well as HP operator *) bind stronger than binary operators. We let $\wedge$ bind stronger than $\vee$, which binds stronger than $\rightarrow, \leftrightarrow$, and let ; bind stronger than $\cup$. Arithmetic operators $+, -, \cdot$ associate to the left. All logical and program operators associate to the right.

These precedences imply that quantifiers and modal operators bind strong, i.e. their scope only extends to the formula immediately after. So, $[\alpha]P \wedge Q \equiv ([\alpha]P) \wedge Q$ and $\forall x P \wedge Q \equiv (\forall x P) \wedge Q$ and $\forall x P \rightarrow Q \equiv (\forall x P) \rightarrow Q$. They also imply $\alpha;\beta \cup \gamma \equiv (\alpha;\beta) \cup \gamma$ and $\alpha \cup \beta;\gamma \equiv \alpha \cup (\beta;\gamma)$ and $\alpha;\beta^* \equiv \alpha;(\beta^*)$.

All logical and program operators associate to the right, so most crucially $P \rightarrow Q \rightarrow R \equiv P \rightarrow (Q \rightarrow R)$. To avoid confusion, we do not adopt precedence conventions between $\rightarrow, \leftrightarrow$ but expect explicit parentheses. So $P \rightarrow Q \leftrightarrow R$ would be considered illegal and explicit parentheses are required to distinguish $P \rightarrow (Q \leftrightarrow R)$ from $(P \rightarrow Q) \leftrightarrow R$. Likewise $P \leftrightarrow Q \rightarrow R$ is illegal and explicit parentheses are required to distinguish $P \leftrightarrow (Q \rightarrow R)$ from $(P \leftrightarrow Q) \rightarrow R$.

[a] It is debatable whether quantifiers are unary operators: $\forall x$ is a unary operator on formulas but $\forall$ is an operator with mixed arguments (a variable and a formula). In a higher-order context with λ-abstractions $\lambda x.P$ for the function that maps x to P, the operator $\forall$ can also understood by understanding $\forall x P$ as an operator on functions: $\forall(\lambda x.P)$. Similar cautionary remarks apply to the understanding of modalities as unary operators. The primary reason for adopting this understanding is that it mnemonically simplifies the precedence rules.

- $[\![e = \tilde{e}]\!] = \{\omega : \omega[\![e]\!] = \omega[\![\tilde{e}]\!]\}$
  That is, an equation is true in the set of states ω where the terms on both sides evaluate to the same number according to Definition 2.4.
- $[\![e \geq \tilde{e}]\!] = \{\omega : \omega[\![e]\!] \geq \omega[\![\tilde{e}]\!]\}$
  That is, a greater-or-equals inequality is true in states ω where the term on the left evaluates to a number that is greater or equal to that on the right.
- $[\![\neg P]\!] = ([\![P]\!])^{\complement} = \mathscr{S} \setminus [\![P]\!]$
  That is, a negated formula $\neg P$ is true in the complement of the set of states in which the formula P itself is true.
- $[\![P \wedge Q]\!] = [\![P]\!] \cap [\![Q]\!]$
  That is, a conjunction is true in the intersection of the states where both conjuncts are true.
- $[\![P \vee Q]\!] = [\![P]\!] \cup [\![Q]\!]$
  That is, a disjunction is true in the union of the set of states where either of its disjuncts is true.
- $[\![P \rightarrow Q]\!] = [\![P]\!]^{\complement} \cup [\![Q]\!]$
  That is, an implication is true in the states where its left-hand side is false or the states where its right-hand side true.

- $[\![P \leftrightarrow Q]\!] = ([\![P]\!] \cap [\![Q]\!]) \cup ([\![P]\!]^\complement \cap [\![Q]\!]^\complement)$
  That is, a biimplication is true in the states where both sides are true or both sides are false.
- $[\![\forall x P]\!] = \{\omega : \nu \in [\![P]\!]$ for all states ν that agree with ω except on $x\}$
  That is, a universally quantified formula $\forall x P$ is true in a state iff its kernel P is also true in all variations of the state that have other real values for x.
- $[\![\exists x P]\!] = \{\omega : \nu \in [\![P]\!]$ for some state ν that agrees with ω except on $x\}$
  That is, an existentially quantified formula $\exists x P$ is true in a state iff its kernel P is true in some variation of the state that has a potentially different real value for x.
- $[\![[\alpha]P]\!] = \{\omega : \nu \in [\![P]\!]$ for all states ν such that $(\omega, \nu) \in [\![\alpha]\!]\}$
  That is, a box modal formula $[\alpha]P$ is true in state ω iff its postcondition P is true in all states ν that are reachable by running α from ω.
- $[\![\langle\alpha\rangle P]\!] = [\![\alpha]\!] \circ [\![P]\!] = \{\omega : \nu \in [\![P]\!]$ for some state ν with $(\omega, \nu) \in [\![\alpha]\!]\}$
  That is, diamond modal formula $\langle\alpha\rangle P$ is true in state ω iff its postcondition P is true in at least one state ν that is reachable by running α from ω.

If $\omega \in [\![P]\!]$, then we say that P is true at ω. The literature sometimes also uses the *satisfaction relation* notation $\omega \models P$ synonymously for $\omega \in [\![P]\!]$. A formula P is *valid*, written $\models P$, iff $[\![P]\!] = \mathscr{S}$, so $\omega \in [\![P]\!]$ for all states ω. A formula P is a *consequence* of a set of formulas Γ, written $\Gamma \models P$, iff, for each state ω: ($\omega \in [\![Q]\!]$ for all $Q \in \Gamma$) implies that $\omega \in [\![P]\!]$.

The semantics of modal formulas $[\alpha]P$ and $\langle\alpha\rangle P$ in differential dynamic logic is illustrated in Fig. 4.6, showing how the truth of P at (all or some) states ν_i reachable by α relates to the truth of $[\alpha]P$ or $\langle\alpha\rangle P$ at state ω.

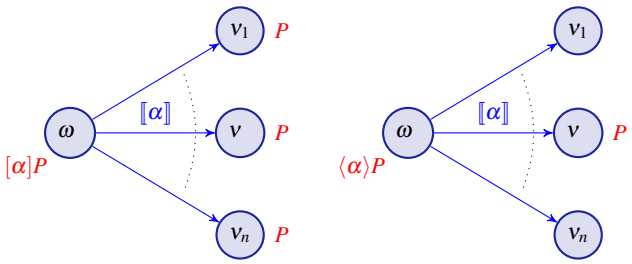


Fig. 4.6 Transition semantics of modalities in dL formulas

4.5 CPS Contracts in Logic

Now that we know what truth and validity are, let's go back to the previous question. Is dL formula (4.12) valid? Is (4.13) valid? Actually, let's first ask if they are

Expedition 4.3 (Set-valued dL semantics $[\![\cdot]\!] : \text{Fml} \to \wp(\mathcal{S})$)

The semantics of a term e directly defines the real-valued function $[\![e]\!] : \mathcal{S} \to \mathbb{R}$ from states to the real value that the term evaluates to in that state (Expedition 2.2 on p. 44). Similarly, Definition 4.2 directly defines inductively, for each dL formula P, the set of states, written $[\![P]\!] \subseteq \mathcal{S}$, in which P is true:

$$
\begin{array}{lcl}
[\![e \geq \tilde{e}]\!] & = & \{\omega : \omega[\![e]\!] \geq \omega[\![\tilde{e}]\!]\} \\
[\![P \wedge Q]\!] & = & [\![P]\!] \cap [\![Q]\!] \\
[\![P \vee Q]\!] & = & [\![P]\!] \cup [\![Q]\!] \\
[\![\neg P]\!] & = & [\![P]\!]^{\complement} = \mathcal{S} \setminus [\![P]\!] \\
[\![\langle\alpha\rangle P]\!] & = & [\![\alpha]\!] \circ [\![P]\!] = \{\omega : \nu \in [\![P]\!] \text{ for some state } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\} \\
[\![[\alpha]P]\!] & = & [\![\neg[\alpha]\neg P]\!] = \{\omega : \nu \in [\![P]\!] \text{ for all states } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\} \\
[\![\exists x P]\!] & = & \{\omega : \nu \in [\![P]\!] \text{ for some state } \nu \text{ that agrees with } \omega \text{ except on } x\} \\
[\![\forall x P]\!] & = & \{\omega : \nu \in [\![P]\!] \text{ for all states } \nu \text{ that agree with } \omega \text{ except on } x\}
\end{array}
$$

When Fml is the set of dL formulas, the semantic brackets for formulas define an operator $[\![\cdot]\!] : \text{Fml} \to \wp(\mathcal{S})$ that defines the meaning $[\![P]\!]$ for each dL formula $P \in \text{Fml}$, which, in turn, defines the set of states $[\![P]\!] \subseteq \mathcal{S}$ in which P is true. The *powerset* $\wp(\mathcal{S})$ is the set of all subsets of the set of states $\mathcal{S}$.

equivalent, i.e. whther the dL formula

$$(4.12) \leftrightarrow (4.13)$$

is valid. Expanding the abbreviations this is the question whether the following dL formula is valid:

$$
\begin{array}{l}
\Big([(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv)^*]\, 0 \leq x \\
\quad \wedge\ [(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv)^*]\, x \leq H\Big) \\
\leftrightarrow\ [(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv)^*]\, (0 \leq x \wedge x \leq H)
\end{array}
\tag{4.15}
$$

Exercise 4.1 gives you an opportunity to convince yourself that the equivalence (4.12) $\leftrightarrow$ (4.13) is indeed valid.[2] So if (4.12) is valid, then (4.13) is valid as well (Exercise 4.2). But is (4.12) valid?

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

[2] This equivalence also foreshadows the fact that CPS provide ample opportunity for questions how multiple system models relate. The dL formula (4.15) relates three different properties of three occurrences of one and the same hybrid program, for example. Over the course of the semester, the need to relate different properties of different CPS will arise even if it may lie dormant for the moment. You are advised to already take notice that this is possible, because dL can form any arbitrary combination and nesting of all its logical operators.

Certainly, (4.12) is not true in a state ω where $\omega(x) < 0$, because from that initial state, zero repetitions of the loop (which is allowed by nondeterministic repetition, Exercise 4.4) lead to the same state ω in which $0 \leq x$ is still false. The initial state is a possible final state for any HP of the form α^*, because it could repeat 0 times. Thus, (4.12) only has a chance of being valid in initial states that satisfy further assumptions, including $0 \leq x$ and $x \leq H$. That is what the preconditions were meant for in Sect. 4.2.5. How can we express a precondition contract in a dL formula?

Preconditions serve a very different role than postconditions do. Postconditions of HP α are expected to be true after every run of α, which is difficult to express in first-order logic (to say the least), but straightforward using the modalities of dL. Do we also need any extra logical operator to express preconditions?

The meaning of precondition **requires**(A) of HP α is that A is assumed to hold before the HP starts. *If* A holds when α starts, then its postcondition **ensures**(B) holds after all runs of HP α. What if A does not hold when the HP starts?

If precondition A does not hold initially, then all bets are off, because the person who started the HP did not obey the requirements that need to be met before the HP can be started safely. The effects of ignoring precondition A are about as useful and predictable as what happens when ignoring the operating requirements "operate in a dry environment only" for a robot when it is submerged into the deep sea. If you are lucky, it will come out unharmed, but the chances are that its electronics will suffer considerably. The CPS contract **requires**(A) **ensures**(B) for a HP α promises that B will always hold after running α if A was true initially when α started. Thus, the meaning of a precondition can be expressed easily using an implication

$$A \rightarrow [\alpha]B \tag{4.16}$$

because an implication is valid if, in every state in which the left-hand side is true, the right-hand side is also true. The implication (4.16) is valid ($\models A \rightarrow [\alpha]B$), if, indeed, for every state ω in which precondition A holds ($\omega \in [\![A]\!]$), it is the case that all runs of HP α lead to states ν (with $(\omega, \nu) \in [\![\alpha]\!]$) in which postcondition B holds (so $\nu \in [\![B]\!]$). By the nature of implication, the dL formula (4.16) does not say what happens in states ω in which the precondition A does not hold (so $\omega \notin [\![A]\!]$).

How does formula (4.16) talk about the runs of a HP and postcondition B again? Recall that the dL formula $[\alpha]B$ is true in exactly those states in which all runs of HP α lead only to states in which postcondition B is true. The implication in (4.16), thus, ensures that this holds in all (initial) states that satisfy precondition A.

Note 22 (Contracts to dL Formulas) *Consider HP α with a CPS contract using a single* **requires**(A) *precondition and single* **ensures**(B) *postcondition:*

requires(A)
ensures(B)
α

This CPS contract can be expressed directly as a logical formula in dL*:*

$$A \to [\alpha]B$$

dL *formulas of this shape are very common and correspond to* Hoare triples *[14] but for hybrid systems instead of conventional programs.*

CPS contracts with multiple preconditions and multiple postconditions can directly be expressed as a dL formula as well (Exercise 4.7).

Recall HP (4.25), which is shown here in a slightly simplified form:

$$\begin{aligned} &\textbf{requires}(0 \leq x \land x = H) \\ &\textbf{ensures}(0 \leq x \land x \leq H) \\ &\big(\{x' = v, v' = -g \,\&\, x \geq 0\}; \\ &\quad \mathsf{if}(x = 0)\, v := -cv\big)^* \end{aligned} \tag{4.17}$$

The dL formula expressing that the CPS contract for HP (4.17) holds is:

$$0 \leq x \land x = H \to \big[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\big)^*\big]\,(0 \leq x \land x \leq H) \tag{4.18}$$

So to find out whether (4.17) satisfies its CPS contract, we ask whether the dL formula (4.18) is valid.

In order to find out whether such a formula is valid, i.e. true in all states, we need some operational way that allows us to tell whether it is valid, because mere inspection of the semantics alone is not a particularly scalable way of approaching validity question.

4.6 Identifying Requirements of a CPS

Before trying to prove any formulas to be valid, it is a pretty good idea to check whether all required assumptions have been found that are necessary for the formula to hold. Otherwise, the proof will fail and we need to start over after having identified the missing requirements from the failed proof attempt. So let us scrutinize dL formula (4.18) and ponder whether there are any circumstances under which it is not true. Even though the bouncing ball is a rather impoverished CPS (it noticeably suffers from a lack of control), its immediate physical intuition still makes the ball a particularly insightful example for illustrating how critical it is to identify the right requirements. Besides, unlike for heavy duty CPS, we trust you have had ample opportunities to become familiar with the behavior of bouncing balls.

Maybe the first thing to notice is that the HP mentions g, which is meant to represent the standard gravity constant, but the formula (4.18) never says that. Certainly, if gravity were negative ($g < 0$), bouncing balls would function rather differently in a quite astonishing way. They would suddenly be floating balls disappearing into the sky and would lose all the joy of bouncing around; see Fig. 4.7.

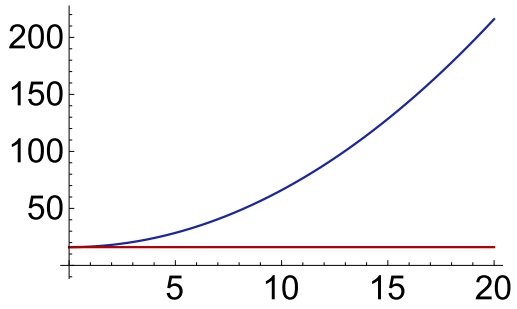


Fig. 4.7 Sample trajectory of a bouncing ball in an anti-gravity field with $g < 0$

So let's modify (4.18) to assume $g = 9.81$:

$$0 \leq x \wedge x = H \wedge g = 9.81 \rightarrow \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \wedge x \leq H) \quad (4.19)$$

Let's undo unnecessarily strict requirements right away, though. What would the bouncing ball do if it were set loose on the moon instead of on Earth? Would it still fall? Things are much lighter on the moon. Yet they still fall down, which what gravity is about, just with a different constant (1.6 on the moon and 25.9 on Jupiter). Besides, none of those constants was particularly precise. Earth's gravity is more like 9.8067. The behavior of the bouncing ball depends on the value of that parameter g. But its qualitative behavior and whether it obeys (4.18) does not.

Note 23 (Parameters) *A common feature of CPS is that their behavior is subject to parameters, which can have quite a non-negligible impact. Yet, it is very hard to determine precise values for all parameters by measurements. When a particular concrete value for a parameter has been assumed to prove a property of a CPS, it is not clear whether that property holds for the true system, which may in reality have a slightly different parameter value.*

Instead of concrete numerical values for a parameter, our analysis can proceed just fine by treating the parameter as a symbolic parameter*, i.e. a variable such as g, which is not assumed to hold a specific numerical value like 9.81. Instead, we would only assume certain constraints about the parameter, say $g > 0$ without choosing a specific value. If we then analyze the CPS with this symbolic parameter g, all analysis results will continue to hold for any concrete choice of g respecting its constraints (here $g > 0$). That results in a stronger statement about the system, which is less fragile, because it does not break down just because the true g is $\approx$9.8067 rather than the previously assumed $g = 9.81$. Those more general statements with symbolic parameters can even be easier to prove than statements about systems with specific magic numbers chosen for their parameters, because their assumptions are explicit.*

In light of these thoughts, we could assume $9 < g < 10$ to be the gravity constant for Earth. Yet, we can also just consider all bouncing balls on all planets in the solar system or elsewhere at once by assuming only $g > 0$ instead of $g = 9.81$ as in (4.19), since this is the only aspect of gravity that the usual behavior of a bouncing ball depends on:

$$0 \leq x \land x = H \land g > 0 \rightarrow \\ \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \land x \leq H) \quad (4.20)$$

Do we expect dL formula (4.20) to be valid, i.e. true in all states? What could go wrong? The insight from modifying (4.18) to (4.19) and finally to (4.20) started with the observation that (4.18) did not include any assumptions about its parameter g. It is worth noting that (4.20) also does not assume anything about c. Bouncing balls clearly would not work as expected if $c > 1$, because such anti-damping would cause the bouncing ball to jump back up higher and higher and higher and ultimately as high up as the moon, clearly falsifying (4.20); see Fig. 4.8.

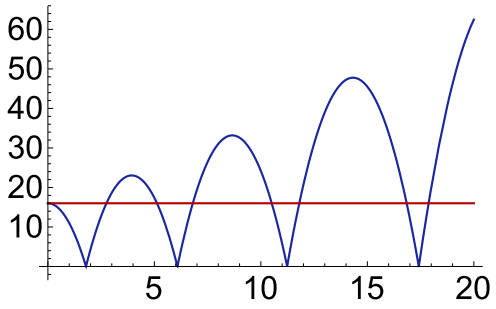


Fig. 4.8 Sample trajectory of a bouncing ball with anti-damping $c > 1$

Being a damping factor, we also expect $c \geq 0$ (despite Exercise 4.14). Yet, (4.20) only has a chance of being true when assuming that c is not too big:

$$0 \leq x \land x = H \land g > 0 \land 1 \geq c \geq 0 \rightarrow \\ \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \land x \leq H) \quad (4.21)$$

Is (4.21) valid now? Or does its truth depend on more assumptions that have not been identified yet? Now, all parameters (H, g, c) have some assumptions in (4.21). Is there some requirement we forgot about? Or did we find them all?

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

What about velocity variable v? Why is there no assumption about it yet? Should there be one? Unlike g and c, velocity v changes over time. What is its initial value allowed to be? What could go wrong?

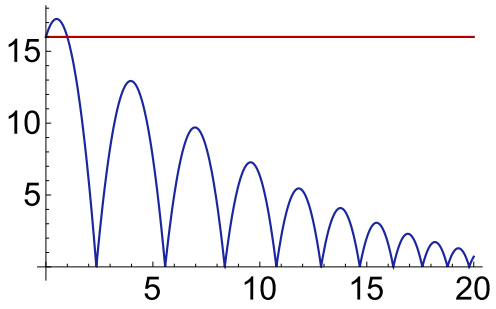


Fig. 4.9 Sample trajectory of a bouncing ball climbing with upwards initial velocity $v > 0$

Indeed, the initial velocity v of the bouncing ball could be positive ($v > 0$), which would make the bouncing ball climb initially, clearly exceeding its initial height H; see Fig. 4.9. This would correspond to the bouncing ball being thrown high up in the air in the beginning, so that its initial velocity v is upwards from its initial height $x = H$. Consequently, (4.21) has to be modified to assume $v \leq 0$ holds initially:

$$0 \leq x \land x = H \land v \leq 0 \land g > 0 \land 1 \geq c \geq 0 \rightarrow \\ \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \land x \leq H) \quad (4.22)$$

Now there's finally assumptions about all parameters and variables of (4.22). That does not mean that we found the right assumptions, yet, but is still a good sanity check. Before wasting cycles on trying to prove or otherwise justify (4.22), let's try once more whether we can find an initial state ω that satisfies all assumptions $v \leq 0 \land 0 \leq x \land x = H \land g > 0 \land 1 \geq c \geq 0$ in the assumptions on the left-hand side of the implication in (4.22) so that ω nevertheless does not satisfy the right-hand side of the implication in (4.22). Such an initial state ω falsifies (4.22) and would, thus, represent a *counterexample*.

Is there still a counterexample to (4.22)? Or have we successfully identified all assumptions so that it is now valid?

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

Formula (4.22) still has a problem. Even if the initial state satisfies all requirements in the antecedent of (4.22), the bouncing ball might still jump higher than it ought to, i.e. higher than its initial height H. That happens if the bouncing ball initially has a very large downwards velocity, so if v is a lot smaller than 0 (sometimes written $v \ll 0$). If v is a little smaller than 0, then the damping c will eat up enough of the ball's kinetic energy so that it cannot jump back up higher than it was initially (H). But if v is a lot smaller than 0, then it starts falling down with so much kinetic energy that the damping on the ground does not slow it down enough, so the ball will come bouncing back higher than it was originally like when dribbling a basket ball; see Fig. 4.10. Under which circumstance this happens depends on the relationship of the initial velocity and height to the damping coefficient.

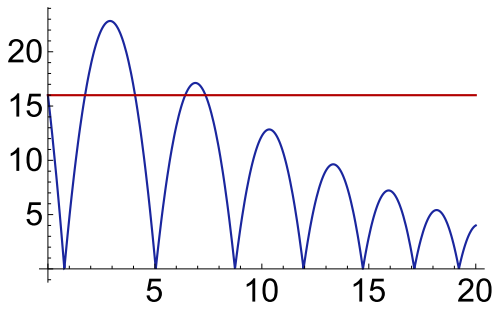


Fig. 4.10 Sample trajectory of a bouncing ball dribbling with fast initial velocity $v < 0$

We could explore this relationship in more detail. But it is easier to infer it by conducting a proof. So we modify (4.22) to simply assume $v = 0$ initially:

$$0 \leq x \land x = H \land v = 0 \land g > 0 \land 1 \geq c \geq 0 \rightarrow \\ \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ \mathsf{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \land x \leq H) \quad (4.23)$$

Is dL formula (4.23) valid now? Or does it still have a counterexample?

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

It seems like all required assumptions have been identified to make the dL formula (4.23) valid so that the bouncing ball described in (4.23) satisfies the post-condition $0 \leq x \leq H$. But after so many failed starts and missing assumptions and requirements for the bouncing ball, it is a good idea to prove (4.23) once and for all beyond any doubt.

In order to be able to prove dL formula (4.23), however, we need to investigate how proving works in CPS. How can dL formulas be proved? And, since first-order formulas are dL formulas as well, one part of the question will be: how can first-order formulas be proved? How can real arithmetic be proved? How can requirements for the safety of CPS be identified systematically? All these questions will be answered in this textbook, but not all of them already in this chapter.

In order to make sure we only need proof techniques for a minimal set of operators of dL, let's simplify (4.23) by getting rid of its if-then-else (Exercise 4.16):

$$0 \leq x \land x = H \land v = 0 \land g > 0 \land 1 \geq c \geq 0 \rightarrow \\ \left[\left(\{x' = v, v' = -g \,\&\, x \geq 0\};\ (?x = 0; v := -cv \cup ?x \neq 0)\right)^*\right](0 \leq x \land x \leq H) \quad (4.24)$$

Having added these crucial assumptions to the dL formula, Quantum quickly rephrases contract (4.17) by incorporating what we learned for dL formula (4.24):

$$\begin{array}{l}\textbf{requires}(0 \leq x \land x = H \land v = 0)\\ \textbf{requires}(g > 0 \land 1 \geq c \geq 0)\\ \textbf{ensures}(0 \leq x \land x \leq H)\\ \big(\{x' = v, v' = -g \,\&\, x \geq 0\};\\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^*\end{array} \tag{4.25}$$

Observing the non-negligible difference between the original conjecture (4.19) and the revised and improved conjecture (4.24), leads us to often adopt the principle of Cartesian Doubt from Expedition 4.4.

Expedition 4.4 (Principle of Cartesian Doubt)

In 1641, René Descartes suggested an attitude of systematic doubt where he would be skeptical about the truth of all beliefs until he found a reason that the beliefs were justified [4]. This influential principle is now known as *Cartesian Doubt* or skepticism.

We will have perfect justifications: proofs. But until we have found proof, it is helpful to adopt the principle of Cartesian Doubt in a weak and pragmatic form. Before setting out on the journey to prove a conjecture, we first scrutinize it to see if we can find a counterexample that would make it false. Such a counterexample will not only save us a lot of misguided effort in trying to prove a false conjecture, but also helps us identify missing assumptions in conjectures and justifies our assumptions to be necessary. If, without making assumption A, a counterexample to a conjecture exists, then A is necessary.

4.7 Summary

This chapter introduced differential dynamic logic (dL), whose operators and their informal meaning is summarized in Table 4.1.

The appendix of this chapter also features first reasoning aspects for CPS. But reasoning for CPS will be investigated systematically in subsequent chapters, one operator at a time, which establishes separate reasoning principles for each operator, rather than proceeding in the more ad-hoc style of this appendix along one example. For future chapters, we should keep the bouncing ball example and its surprising subtleties in mind, though.

Table 4.1 Operators and (informal) meaning in differential dynamic logic (dL)

dL	Operator	Meaning
$e = \tilde{e}$	equals	true iff values of e and $\tilde{e}$ are equal
$e \geq \tilde{e}$	equals	true iff value of e greater-or-equal to $\tilde{e}$
$\neg P$	negation / not	true if P is false
$P \wedge Q$	conjunction / and	true if both P and Q are true
$P \vee Q$	disjunction / or	true if P is true or if Q is true
$P \rightarrow Q$	implication / implies	true if P is false or Q is true
$P \leftrightarrow Q$	bi-implication / equivalent	true if P and Q are both true or both false
$\forall x P$	universal quantifier / for all	true if P is true for all values of variable x
$\exists x P$	existential quantifier / exists	true if P is true for some values of variable x
$[\alpha]P$	$[\cdot]$ modality / box	true if P is true after all runs of HP α
$\langle\alpha\rangle P$	$\langle\cdot\rangle$ modality / diamond	true if P is true after at least one run of HP α

4.8 Appendix

This appendix already features first reasoning aspects for CPS even if a fully systematic account to CPS reasoning will be pursued from scratch in more elegant ways in subsequent chapters. Especially for readers who have seen the Floyd-Hoare calculus for conventional programs [5, 14] or prefer the start with a concrete example, this appendix can be a useful stepping stone for reaching that level of generality. This appendix begins a semiformal study of the bouncing ball, which is an optional but useful preparation for the next chapter.

4.8.1 Intermediate Conditions for a Proof of Sequential Compositions

Before proceeding any further with ways of proving dL formulas, let's simplify (4.24) grotesquely by removing the loop:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\ \left[\{x' = v, v' = -g \,\&\, x \geq 0\}; (?x = 0; v := -cv \cup ?x \neq 0)\right](0 \leq x \wedge x \leq H) \quad (4.26)$$

Removing the loop clearly changes the behavior of the bouncing ball. It no longer bounces particularly well. All it can do now is fall and, if it reaches the floor, have its velocity reverted without actually ever climbing back up. So if we manage to prove (4.26), we certainly have not shown the actual dL formula (4.24). But it's a start, because the behavior modeled in (4.26) is a part of the behavior of (4.24). So it is useful (and easier) to understand the loop-free HP (4.26) first.

The dL formula (4.26) has a number of assumptions $0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0$ that can be used during the proof. It claims that the postcondition $0 \leq x \wedge x \leq H$ holds after all runs of the HP in the $[\cdot]$ modality. The top-level operator

inside the modality of (4.26) is a sequential composition (;), for which we need to find a proof argument.

The HP in (4.26) first follows a differential equation first and then a discrete program ($?x = 0; v := -cv \cup ?x \neq 0$). This leads to different intermediate states after the differential equation and before the discrete program.

Note 24 (Intermediate states of sequential compositions) *The first HP α in a sequential compositions $\alpha;\beta$ may reach a whole range of states, which represent intermediate states for the sequential composition $\alpha;\beta$, i.e. states that are final states for α and initial states for β. The intermediate states of $\alpha;\beta$ are the states μ in the semantics $[\![\alpha;\beta]\!]$ from Chap. 3:*

$$[\![\alpha;\beta]\!] = [\![\alpha]\!] \circ [\![\beta]\!] = \{(\omega,\nu) : (\omega,\mu) \in [\![\alpha]\!], (\mu,\nu) \in [\![\beta]\!]\}$$

Can we find a way of summarizing what all intermediate states between the differential equation and the discrete program of (4.26) have in common? They differ by how long the CPS has followed the differential equation.

If the system has followed the differential equation of (4.26) for time t, then the resulting velocity $v(t)$ at time t and height $x(t)$ at time t will be

$$v(t) = -gt, \quad x(t) = H - \frac{g}{2}t^2 \tag{4.27}$$

This answer can be found by integrating or solving the differential equations (Example 2.4 on p. 32). This knowledge (4.27) is useful but it is not (directly) clear how to use it to describe what all intermediate states have in common, because the time t in (4.27) is not available as a variable in the HP (4.26).[3] Can the intermediate states be described by a relation of the variables that (unlike t) are actually in the system? That is, an arithmetic formula relating variables x, v, g, H?

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

One way of producing a relation from (4.27) is to get the units aligned and get rid of time t. Time drops out of the "equation" when squaring the identity for velocity:

$$v(t)^2 = g^2t^2, \quad x(t) = H - \frac{g}{2}t^2$$

and multiplying the identity for position by $2g$:

$$v(t)^2 = g^2t^2, \quad 2gx(t) = 2gH - 2\frac{g^2}{2}t^2$$

Then substituting the first equation into the second yields

[3] Following these thoughts a bit further reveals how (4.27) can actually be used perfectly well to describe intermediate states when changing the HP (4.26) a little bit. But working with solutions is still not the way that gets us to the goal the quickest, usually, because of their difficult arithmetic.

$$2gx(t) = 2gH - v(t)^2$$

This equation does not depend on time t, so we expect it to hold after all runs of the differential equation irrespective of t:

$$2gx = 2gH - v^2 \tag{4.28}$$

We conjecture the intermediate condition (4.28) to hold in the intermediate state of the sequential composition in (4.26). In order to prove (4.26) we can decompose our reasoning into two parts. The first part will prove that the intermediate condition (4.28) holds after all runs of the first differential equation. The second part will assume (4.28) to hold and prove that all runs of the discrete program in (4.26) from any state satisfying (4.28) satisfy the postcondition $0 \leq x \wedge x \leq H$.

Note 25 (Intermediate conditions as contracts for sequential composition) *For a HP that is a sequential composition $\alpha;\beta$ an* intermediate condition *is a formula that characterizes the intermediate states in between HP α and β. That is, for a* dL *formula*

$$A \rightarrow [\alpha;\beta]B$$

an intermediate condition is a formula E such that the following dL *formulas are valid:*

$$A \rightarrow [\alpha]E \qquad \text{and} \qquad E \rightarrow [\beta]B$$

The first dL *formula expresses that intermediate condition E characterizes the intermediate states accurately, i.e. E actually holds after all runs of HP α from states satisfying A. The second* dL *formula says that the intermediate condition E characterizes intermediate states well enough, i.e. E is all we need to know about a state to conclude that all runs of β end up in B. That is, from all states satisfying E (including those that result by running α from a state satisfying A), B holds after all runs of β.*

For proving (4.26), we conjecture that (4.28) is an intermediate condition, which requires us to prove the following two dL formulas:

$$\begin{aligned} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 &\rightarrow [x' = v, v' = -g \,\&\, x \geq 0] 2gx = 2gH - v^2 \\ 2gx = 2gH - v^2 &\rightarrow [?x = 0; v := -cv \cup ?x \neq 0]\,(0 \leq x \wedge x \leq H) \end{aligned} \tag{4.29}$$

Let's focus on the second formula in (4.29). Do we expect to be able to prove it? Do we expect it to be valid?

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

The second formula of (4.29) claims that $0 \leq x$ holds after all runs of the hybrid program $?x = 0; v := -cv \cup ?x \neq 0$ from all states that satisfy $2gx = 2gH - v^2$. That is a bit much to hope for, however, because $0 \leq x$ is not even ensured in the pre-

condition of this second formula. So the second formula of (4.29) is not valid. How can this problem be resolved? By adding $0 \leq x$ into the intermediate condition, thus, requiring us to prove two formulas:

$$\begin{aligned} 0 \leq x \land x = H \land v = 0 \land g > 0 \land 1 \geq c \geq 0 \rightarrow \\ [x' = v, v' = -g \,\&\, x \geq 0](2gx = 2gH - v^2 \land x \geq 0) \\ 2gx = 2gH - v^2 \land x \geq 0 \rightarrow [?x = 0; v := -cv \cup ?x \neq 0]\,(0 \leq x \land x \leq H) \end{aligned} \tag{4.30}$$

Proving the first formula in (4.30) requires us to handle differential equations, which we will get to in Chap. 5. The second formula in (4.30) is the one whose proof is discussed first.

4.8.2 A Proof of Choice

The second formula in (4.30) has a nondeterministic choice ($\cup$) as the top-level operator in its $[\cdot]$ modality. How can we prove a formula of the form

$$A \rightarrow [\alpha \cup \beta]B \tag{4.31}$$

Recalling its semantics from Chap. 3,

$$[\![\alpha \cup \beta]\!] = [\![\alpha]\!] \cup [\![\beta]\!]$$

HP $\alpha \cup \beta$ has two possible behaviors. It could run as HP α does or as β does. And it is chosen nondeterministically which of the two behaviors happens. Since the behavior of $\alpha \cup \beta$ could be either α or β, proving (5.5) requires proving B to hold after α and after β. More precisely, (5.5) assumes A to hold initially, otherwise (5.5) is vacuously true. Thus, proving (5.5) allows us to assume A and requires us to prove that B holds after all runs of α (which is permitted behavior for $\alpha \cup \beta$) and to prove that, assuming A holds initially, that B holds after all runs of β (which is also permitted behavior of $\alpha \cup \beta$).

Note 26 (Proving choices) *For a HP that is a nondeterministic choice $\alpha \cup \beta$, we can prove*

$$A \to [\alpha \cup \beta]B$$

by proving the following dL formulas:

$$A \to [\alpha]B \qquad \textit{and} \qquad A \to [\beta]B$$

Using these thoughts on the second formula of (4.30), we could prove that formula if we would manage to prove both of the following dL formulas:

$$\begin{aligned} 2gx = 2gH - v^2 \land x \geq 0 \to [?x = 0; v := -cv]\,(0 \leq x \land x \leq H) \\ 2gx = 2gH - v^2 \land x \geq 0 \to [?x \neq 0]\,(0 \leq x \land x \leq H) \end{aligned} \tag{4.32}$$

4.8.3 A Proof of Tests

Consider the second formula of (5.8). Proving it requires us to understand how to handle a test $?Q$ in a modality $[?Q]$. The semantics of a test $?Q$ from Chap. 3

$$[\![?Q]\!] = \{(\omega, \omega) \;:\; \omega \in [\![Q]\!]\} \tag{4.33}$$

says that a test $?Q$ completes successfully without changing the state in any state ω in which Q holds (i.e. $\omega \in [\![Q]\!]$) and fails to run in all other states (i.e. where $\omega \notin [\![Q]\!]$). How can we prove a formula with a test:

$$A \to [?Q]B \tag{4.34}$$

This formula expresses that, from all initial states that satisfy A, all runs of $?Q$ reach states satisfying B. When is there such a run of $?Q$ at all? There is a run from state ω if and only if Q holds in ω. So the only cases to worry about are initial states that satisfy Q, otherwise, the HP in (4.34) cannot execute at all and fails miserably so that the run is discarded. Hence, we get to assume Q holds, as HP $?Q$ does not otherwise execute. In all states that the HP $?Q$ reaches from states satisfying A, (4.34) conjectures that B holds. By (4.33), the final states that $?Q$ reaches are the same as the initial state (as long as they satisfy Q so that HP $?Q$ can be executed at all). That is, postcondition B needs to hold in all states from which $?Q$ runs (i.e. that satisfy Q) and that satisfy the precondition A. So (4.34) can be proved by proving

$$A \land Q \to B$$

Note 27 (Proving tests) *For a HP that is a test ?Q, we can prove*

$$A \to [?Q]B$$

by proving the following dL *formula:*

$$A \wedge Q \to B$$

Using this for the second formula of (5.8), Note 27 reduces proving the second formula of (5.8)

$$2gx = 2gH - v^2 \wedge x \geq 0 \to [?x \neq 0]\,(0 \leq x \wedge x \leq H)$$

to proving

$$2gx = 2gH - v^2 \wedge x \geq 0 \wedge x \neq 0 \to 0 \leq x \wedge x \leq H \tag{4.35}$$

Now we are left with arithmetic that we need to prove. Proofs for arithmetic and propositional logical operators such as $\wedge$ and $\to$ will be considered in a later chapter. For now, we notice that the formula $0 \leq x$ in the right-hand side of $\to$ is justified by assumption $x \geq 0$ if we flip the inequality around. Yet, $x \leq H$ does not exactly have a justification in (4.35), because we lost the assumptions about H somewhere.

How could that happen? We used to know $x \leq H$ in (4.26). We still knew about it in the first formula of (4.30). But we somehow let it disappear from the second formula of (4.30), because we chose an intermediate condition that was too weak.

This is a common problem in trying to prove properties of CPS or of any other mathematical statements. One of our intermediate steps might have been too weak, so that our attempt of proving it fails and we need to revisit how we got there. For sequential compositions, this is actually a nonissue as soon as we move on (in the next chapter) to a proof technique that is more useful than the intermediate conditions from Note 25. But similar difficulties can arise in other parts of proof attempts.

In this case, the fact that we lost $x \leq H$ can be fixed by including it in the intermediate conditions, because it can be shown to hold after the differential equation. Other crucial assumptions have also suddenly disappeared in our reasoning. An extra assumption $1 \geq c \geq 0$, for example, is crucially needed to justify the first formula of (5.8). It is much easier to see why that particular assumption can be added to the intermediate contract without changing the argument much. The reason is that c never ever changes during the system run so if $1 \geq c$ was true initially, it is still true.

Note 28 (Changing assumptions in a proof) *It is very difficult to come up with bug-free code. Just thinking about your assumptions really hard does not ensure correctness. But we can gain confidence that our system does what we want it to by proving that certain properties are satisfied.*

Assumptions and arguments in a hybrid program frequently need to be changed during the search for a proof of safety. It is easy to make subtle mistakes in informal arguments such as "I need to know C here and I would know C if I had included it here or there, so now I hope the argument holds". This is one of many reasons why we are better off if our CPS proofs are rigorous, because we would rather not end up in trouble because of a subtle flaw in a correctness argument. The rigorous, formal proof calculus for differential dynamic logic (dL) that we develop in Chaps. 5 and 6 will help us avoid the pitfalls of informal arguments. The theorem prover KeYmaera X [7] implements a proof calculus for dL, which supports such mathematical rigor.

A related observation from our informal arguments in this chapter is that we desperately need a way to keep an argument consistent as a single argument justifying one conjecture. Quite the contrary to the informal loose threads of argumentation we have pursued in this chapter for the sake of developing an intuition. Consequently, we will investigate what holds all arguments together and what constitutes an actual proof in subsequent chapters. A proof in which the relationship of premises to conclusions via proof steps is rigorous.

Moreover, there are two loose ends in our arguments. For one, the differential equation in (4.30) is still waiting for an argument that could help us prove it. Also, the assignment in (5.8) still needs to be handled and its sequential composition needs an intermediate contract (Exercise 4.17). Both will be pursued in the next chapter, where we move to a systematic and rigorous reasoning style for CPS.

Exercises

4.1. Show that (4.15) is valid. It is okay to focus only on this case, even though the argument is more general, because the following dL formula is valid for any hybrid program α:

$$[\alpha]F \wedge [\alpha]G \leftrightarrow [\alpha](F \wedge G)$$

4.2 (Equivalence). Let A, B be dL formulas. Suppose $A \leftrightarrow B$ is valid and A is valid and show that B is then valid. Suppose $A \leftrightarrow B$ is valid and you replace an occurrence of A in another formula P with B to obtain formula Q. Are P and Q then equivalent, i.e. is $P \leftrightarrow Q$ valid?

4.3. Let A, B be dL formulas. Suppose $A \leftrightarrow B$ is true in state ω and A is true in state ω. That is, $\omega \in [\![A \leftrightarrow B]\!]$ and $\omega \in [\![A]\!]$. Is B true in state ω? Prove or disprove. Is B valid? Prove or disprove.

4.4. Let α be an HP. Let ω be a state with $\omega \notin [\![P]\!]$. Does $\omega \notin [\![[\alpha^*]P]\!]$ have to hold? Prove or disprove.

4.5. Let α be an HP. Let ω be a state with $\omega \in [\![P]\!]$. Does $\omega \in [\![[\alpha^*]P]\!]$ have to hold? Prove or disprove.

4.6. Let α be an HP. Let ω be a state with $\omega \in [\![P]\!]$. Does $\omega \in [\![\langle \alpha^* \rangle P]\!]$ have to hold? Prove or disprove.

4.7 (Multiple pre/postconditions). Suppose you have a HP α with a CPS contract using multiple preconditions $A_1, \ldots, A_n$ and multiple postconditions $B_1, \ldots, B_m$:

$$
\begin{array}{l}
\textbf{requires}(A_1) \\
\textbf{requires}(A_2) \\
\vdots \\
\textbf{requires}(A_n) \\
\textbf{ensures}(B_1) \\
\textbf{ensures}(B_2) \\
\vdots \\
\textbf{ensures}(B_m) \\
\alpha
\end{array}
$$

How can this CPS contract be expressed in a dL formula? If there are multiple alternatives on how to express it, discuss advantages and disadvantages of each option.

4.8. For each of the following dL formulas, determine if they are valid, satisfiable, *and/or* unsatisfiable:

1. $[?x \geq 0]x \geq 0$.
2. $[?x \geq 0]x \leq 0$.
3. $[?x \geq 0]x < 0$.
4. $[?true]true$.
5. $[?true]false$.
6. $[?false]true$.
7. $[?false]false$.
8. $[x' = 1 \,\&\, true]true$.
9. $[x' = 1 \,\&\, true]false$.
10. $[x' = 1 \,\&\, false]true$.
11. $[x' = 1 \,\&\, false]false$.
12. $[(x' = 1 \,\&\, true)^*]true$.
13. $[(x' = 1 \,\&\, true)^*]false$.
14. $[(x' = 1 \,\&\, false)^*]true$.
15. $[(x' = 1 \,\&\, false)^*]false$.

4.9. For each of the following dL formulas, determine if they are valid, satisfiable, *and/or* unsatisfiable:

1. $x > 0 \rightarrow [x' = 1]x > 0$
2. $x > 0 \rightarrow [x' = -1]x < 0$
3. $x > 0 \rightarrow [x' = -1]x \geq 0$

4. $x>0 \to [(x:=x+1)^*]x>0$
5. $x>0 \to [(x:=x+1)^*]x>1$
6. $[x:=x^2+1;x'=1]x>0$.
7. $[(x:=x^2+1;x'=1)^*]x>0$.
8. $[(x:=x+1;x'=-1)^*;?x>0;x'=2]x>0$
9. $x=0 \to [x'=1;x'=-2)]x<0$.

4.10. For each of the following dL formulas, determine if they are valid, satisfiable, *and/or* unsatisfiable:

1. $\langle x'=-1\rangle x<0$
2. $x>0 \wedge \langle x'=1\rangle x<0$
3. $x>0 \wedge \langle x'=-1\rangle x<0$
4. $x>0 \to \langle x'=1\rangle x>0$
5. $[(x:=x+1)^*]\langle x'=-1\rangle x<0$.
6. $x>0 \to [x'=2]\big(x>0 \wedge [x'=1]x>0 \wedge \langle x'=-2\rangle x=0\big)$
7. $\langle x'=2\rangle[x'=-1]\langle x'=5\rangle x>0$
8. $\forall x\, \langle x'=-1\rangle x<0$
9. $\forall x\, [x'=1]x\geq 0$
10. $\exists x\, [x'=-1]x<0$
11. $\forall x \exists d\, (x\geq 0 \to [x'=d]x\geq 0)$
12. $\forall x\, (x\geq 0 \to \exists d\, [x'=d]x\geq 0)$
13. $[x'=1]\big(x\geq 0 \to [x'=2]x\geq 0\big)$
14. $[x'=1]x\geq 0 \to [x'=2]x\geq 0$
15. $[x'=2]x\geq 0 \to [x'=1]x\geq 0$
16. $\langle x'=2\rangle x\geq 0 \to [x'=1]x\geq 0$

4.11. For each $j,k \in \{\text{satisfiable}, \text{unsatisfiable}, \text{valid}\}$ answer whether there is a formula that is j but not k. Also answer for each such j,k whether there is a formula that is j but its negation is not k. Briefly justify each answer.

4.12 (Set-valued semantics). There are at least two styles of giving a meaning to a logical formula. One way is, as in Definition 4.2, to inductively define a satisfaction relation $\models$ that holds between a state ω and a dL formula P, written $\omega \models P$, whenever the formula P is true in the state ω. Its definition will include, among other cases, the following:

$$\begin{array}{l}\omega \models P \wedge Q \text{ iff } \omega \models P \text{ and } \omega \models Q\\ \omega \models \langle\alpha\rangle P \text{ iff } \nu \models P \text{ for some state } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\\ \omega \models [\alpha]P \text{ iff } \nu \models P \text{ for all states } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\end{array}$$

The other way is to inductively define, for each dL formula P, the set of states, written $[\![P]\!]$, in which P is true. Its definition will include, among other cases, the following:

$$\begin{array}{lcl}
[\![e \geq \tilde{e}]\!] & = & \{\omega : \omega[\![e]\!] \geq \omega[\![\tilde{e}]\!]\} \\
[\![P \wedge Q]\!] & = & [\![P]\!] \cap [\![Q]\!] \\
[\![\neg P]\!] & = & [\![P]\!]^{\complement} = \mathscr{S} \setminus [\![P]\!] \\
[\![\langle\alpha\rangle P]\!] & = & [\![\alpha]\!] \circ [\![P]\!] = \{\omega : \nu \in [\![P]\!] \text{ for some state } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\} \\
[\![[\alpha]P]\!] & = & [\![\neg[\alpha]\neg P]\!] = \{\omega : \nu \in [\![P]\!] \text{ for all states } \nu \text{ such that } (\omega,\nu) \in [\![\alpha]\!]\} \\
[\![\exists x P]\!] & = & \{\omega : \nu \in [\![P]\!] \text{ for some state } \nu \text{ that agrees with } \omega \text{ except on } x\} \\
[\![\forall x P]\!] & = & \{\omega : \nu \in [\![P]\!] \text{ for all states } \nu \text{ that agree with } \omega \text{ except on } x\}
\end{array}$$

Prove that both styles of defining the semantics are equivalent. That is $\omega \models P$ iff $\omega \in [\![P]\!]$ for all states ω and all dL formulas P.

4.13 (Rediscover Quantum). Help Quantum the bouncing ball with a clean-slate approach. Pull up a blank sheet of paper and double-check whether you can help Quantum identify all the requirements that imply the following formula:

$$\big[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\\ \quad \mathsf{if}(x = 0)\, v := -cv\big)^*\big](0 \leq x \leq H)$$

What are the requirements for the following formula to be true in which the order of the sequential composition is swapped so that the discrete step comes first?

$$\big[\big(\mathsf{if}(x = 0)\, v := -cv;\\ \quad \{x' = v, v' = -g \,\&\, x \geq 0\}\big)^*\big](0 \leq x \leq H)$$

4.14. What would happen with the bouncing ball if $c < 0$? Consider a variation of the arguments in Sect. 4.6 where instead of the assumption in (4.21), you assume $c < 0$. Is the formula valid? What happens with a bouncing ball of damping $c = 1$?

4.15 (Deflatable Quantum). Help Quantum the bouncing ball identify all requirements that imply the following formula, in which the bouncing ball might deflate and lie flat using the model from Sect. 4.2.3:

$$\big[\big(\{x' = v, v' = -g \,\&\, x \geq 0\};\\ \quad \mathsf{if}(x = 0)\, (v := -cv \cup v := 0)\big)^*\big](0 \leq x \leq H)$$

4.16. We went from (4.23) to (4.24) by removing an if-then-else. Explain how this works and justify why it is okay to do this transformation. It is okay to focus only on this case, even though the argument is more general.

4.17 (*). Find an intermediate condition for proving the first formula in (5.8). The proof of the resulting formulas is complicated significantly by the fact that assignments have not yet been discussed in this chapter. Can you find a way of proving the resulting formulas before the next chapter develops how to handle assignments?

4.18 (**). Sect. 4.8.1 used a mix of a systematic and ad-hoc approach for producing an intermediate condition that was based on solving and combining differential equations. Can you think of a more systematic rephrasing?

4.19 (**). Note 25 in Sect. 4.8.1 gave a way of showing a property of a sequential composition

$$A \to [\alpha;\beta]B$$

by identifying an intermediate condition E and showing

$$A \to [\alpha]E \qquad \text{and} \qquad E \to [\beta]B$$

Can you already see a way of exploiting the operators of differential dynamic logic to show the same formula without having to be creative by inventing an intermediate condition E?

4.20 (**). How could formula (4.30) be proved using its differential equation?

References

1. Asimov, I. Runaround. *Astounding Science Fiction* **March** (1942).
2. Carroll, L. *Alice's Adventures in Wonderland* (Macmillan, 1865).
3. Derler, P., Lee, E. A., Tripakis, S. & Törngren, M. *Cyber-physical system design contracts* in *ICCPS* (eds Lu, C., Kumar, P. R. & Stoleru, R.) (ACM, 2013), 109–118.
4. Descartes, R. *Meditationes de prima philosophia, in qua Dei existentia et animae immortalitas demonstratur* (1641).
5. Floyd, R. W. *Assigning meanings to programs* in *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics* (ed Schwartz, J. T.) **19** (Providence, 1967), 19–32.
6. Frege, G. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* (Verlag von Louis Nebert, 1879).
7. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M. & Platzer, A. *KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems* in *CADE* (eds Felty, A. & Middeldorp, A.) **9195** (Springer, 2015), 527–538. doi:10.1007/978-3-319-21401-6_36.
8. Gentzen, G. Untersuchungen über das logische Schließen. I. *Math. Zeit.* **39,** 176–210 (1935).
9. Gentzen, G. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Math. Ann.* **112,** 493–565 (1936).
10. Gödel, K. *Über die Vollständigkeit des Logikkalküls* PhD thesis (Universität Wien, 1929).
11. Gödel, K. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys.* **38,** 173–198 (1931).
12. Gödel, V. K. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* **12,** 280–287 (1958).
13. Hilbert, D. & Ackermann, W. *Grundzüge der theoretischen Logik* (Springer, 1928).

14. Hoare, C. A. R. An Axiomatic Basis for Computer Programming. *Commun. ACM* **12,** 576–580 (1969).
15. Kripke, S. A. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica* **16,** 83–94 (1963).
16. Leibniz, G. W. *Generales inquisitiones de analysi notionum et veritatum* (1686).
17. *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012* (IEEE, 2012).
18. Logozzo, F. *Practical Verification for the Working Programmer with CodeContracts and Abstract Interpretation - (Invited Talk)* in *VMCAI* (eds Jhala, R. & Schmidt, D. A.) **6538** (Springer, 2011), 19–22. doi:10.1007/978-3-642-18275-4_3.
19. Meyer, B. Applying "Design by Contract". *Computer* **25,** 40–51 (Oct. 1992).
20. Mitsch, S. & Platzer, A. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models. *Form. Methods Syst. Des.* **49.** Special issue of selected papers from RV'14, 33–74 (2016).
21. Platzer, A. *Differential Dynamic Logic for Verifying Parametric Hybrid Systems.* in *TABLEAUX* (ed Olivetti, N.) **4548** (Springer, 2007), 216–232. doi:10.1007/978-3-540-73099-6_17.
22. Platzer, A. Differential Dynamic Logic for Hybrid Systems. *J. Autom. Reas.* **41,** 143–189 (2008).
23. Platzer, A. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics* doi:10.1007/978-3-642-14509-4 (Springer, Heidelberg, 2010).
24. Platzer, A. *Stochastic Differential Dynamic Logic for Stochastic Hybrid Programs* in *CADE* (eds Bjørner, N. & Sofronie-Stokkermans, V.) **6803** (Springer, 2011), 431–445. doi:10.1007/978-3-642-22438-6_34.
25. Platzer, A. A Complete Axiomatization of Quantified Differential Dynamic Logic for Distributed Hybrid Systems. *Log. Meth. Comput. Sci.* **8.** Special issue for selected papers from CSL'10, 1–44 (2012).
26. Platzer, A. *Logics of Dynamical Systems* in *LICS* (IEEE, 2012), 13–24. doi:10.1109/LICS.2012.13.
27. Platzer, A. *The Complete Proof Theory of Hybrid Systems* in *LICS* (IEEE, 2012), 541–550. doi:10.1109/LICS.2012.64.
28. Platzer, A. *Teaching CPS Foundations With Contracts* in *CPS-Ed* (2013), 7–10.
29. Platzer, A. Differential Game Logic. *ACM Trans. Comput. Log.* **17,** 1:1–1:51 (2015).
30. Platzer, A. A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *J. Autom. Reas.* doi:10.1007/s10817-016-9385-1 (2016).
31. Platzer, A. *Logic & Proofs for Cyber-Physical Systems* in *IJCAR* (eds Olivetti, N. & Tiwari, A.) **9706** (Springer, 2016), 15–21. doi:10.1007/978-3-319-40229-1_3.

32. Platzer, A. & Clarke, E. M. *The Image Computation Problem in Hybrid Systems Model Checking.* in *HSCC* (eds Bemporad, A., Bicchi, A. & Buttazzo, G. C.) **4416** (Springer, 2007), 473–486. doi:10.1007/978-3-540-71493-4_37.
33. Pnueli, A. *The Temporal Logic of Programs* in *FOCS* (IEEE, 1977), 46–57.
34. Pratt, V. R. *Semantical Considerations on Floyd-Hoare Logic* in *17th Annual Symposium on Foundations of Computer Science, 25-27 October 1976, Houston, Texas, USA* (IEEE, 1976), 109–121.
35. Prior, A. *Time and Modality* (Clarendon Press, 1957).
36. Scott, D. S. Logic and Programming Languages. *Commun. ACM* **20,** 634–641 (1977).
37. Skolem, T. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Videnskapsselskapet Skrifter, I. Matematisk-naturvidenskabelig Klasse* **6,** 1–36 (1920).
38. Whitehead, A. N. & Russell, B. *Principia Mathematica* (Cambridge Univ. Press, 1910).
39. Xu, D. N., Jones, S. L. P. & Claessen, K. *Static contract checking for Haskell* in *POPL* (eds Shao, Z. & Pierce, B. C.) (ACM, 2009), 41–52. doi:10.1145/1480881.1480889.
40. Zuliani, P., Platzer, A. & Clarke, E. M. Bayesian Statistical Model Checking with Application to Simulink/Stateflow Verification. *Form. Methods Syst. Des.* **43,** 338–367 (2013).