



# Chapter 4

## Safety & Contracts

**Synopsis** This chapter provides a lightweight introduction to safety specification techniques for cyber-physical systems. It discusses how program contracts generalize to CPS by declaring expectations on the initial states together with guarantees for all 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 the precise meaning of CPS contracts. In subsequent chapters, differential dynamic logic plays a central rôle in rigorous verification of CPSs as well. This chapter also 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 perfectly intuitive setting.

### 4.1 Introduction

In the previous chapters, we have studied models of cyber-physical systems and the use of hybrid programs as their programming language [21, 22, 25, 30]. The distinguishing feature of hybrid programs 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 the study of ways of making sure that the resulting behavior, however flexible and powerful it may be, also meets the required safety and correctness standards. Powerful and flexible CPSs would not be very useful if they failed to meet certain crucial safety demands.

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, and, if they fail, this alerts you right away to flaws

in the design of the program. In that case, you have experienced first hand that it is much easier to find and fix problems in programs starting from the first contract that failed in the middle of the program, rather than from the mere observation about the symptoms that ultimately surface when the final output of the program 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 have had the opportunity to observe is that they can be used in proofs, which show that *every* program run will satisfy the contracts, as opposed to just the ones you have tried. 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 cleverly 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, 17, 18, 34, 36, 39].

The principles of contracts help cyber-physical systems [3, 21, 22, 27] 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 as well as 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 100 mph on a 55 mph 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 will know that it made a mistake, but it has become too late to do anything about it, because the brakes of the car cannot possibly slow the car down quickly enough. The car is "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 also in CPS [19], the design of those contracts requires proof to ensure that they respond early enough and that safety is always maintained.

For those reasons, this textbook will focus on the rôle of proofs as correctness arguments for CPS contracts much more than on their use in dynamical checking. Due to the physical consequences of malfunctions, correctness requirements on CPS are also more stringent. Subtle nuances in their behavior may require significantly more challenging arguments than, e.g., mere 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 just 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 of the rôle of requirements and contracts in CPS as well as important ways of formalizing CPS properties and their

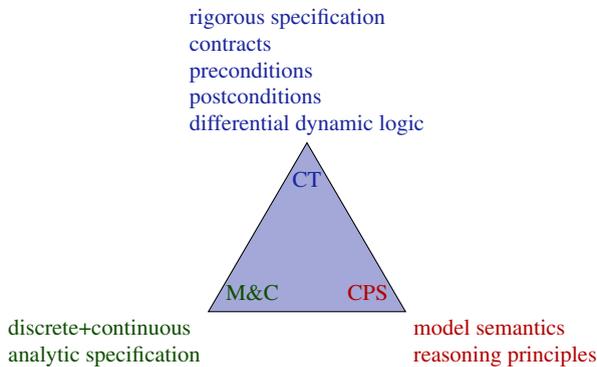
analysis. The material in this chapter provides an intuitive gradual introduction to correctness specification techniques for CPS [20–22, 25, 30].

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 enable us to make precise statements about the expected behavior of a CPS. This forms 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 conveys the formidable subtleties involved in hybrid systems models, which are crucial for understanding CPS. This chapter is devoted to contracts in the form of pre- and postconditions for CPS models. We will begin to rigorously specify our requirements and expectations for 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 [21, 22, 25, 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 tentatively relating it to their reasoning principles. This alignment will only be fully covered in the next chapter, though.



## 4.2 A Gradual Introduction to CPS Contracts

This section provides a gradual and informal introduction to contracts for cyber-physical systems. Its focus is on an intuitive development of the need for contracts,

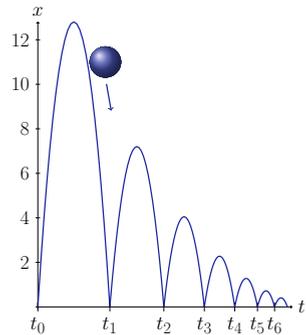
which provides the motivation for a subsequent rigorous development of a logic for CPS in which every aspect has an unambiguously well-defined meaning. This section also introduces the running example of Quantum the bouncing ball, which will be with us as a simple intuitive yet surprisingly representative example throughout the book.

A model of 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. Developing this example will be yet another welcome modeling exercise.

### 4.2.1 The Adventures of Quantum the Bouncing Ball

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. 4.1). Quantum the bouncing ball really was not much of a CPS, because bouncing balls do not actually have any interesting decisions to make. At least, Quantum was quite content without having to face any decisions, although, he will discover in Chap. 8 how empowering, subtle, and intriguing decisions can be. For one thing, Quantum did not even bring his computer, so 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 forms a perfectly reasonable hybrid system, because, after a closer look, the system turns out to involve both 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 forth. 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 little bouncing ball, though, so he 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 it as a hybrid system. The ball at height  $x$  is falling subject to gravity  $g$ :

$$x'' = -g$$

That is, the height  $x$  changes with the second time-derivative  $-g$  along the differential equation  $x'' = -g$  with constant gravity factor  $g$ . 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 effects 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 to understand 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  $x = 0$ :

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

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 presently 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 in reality slowly climbs 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 as 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 toward the ground) and then, all of a sudden, he 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:

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

Now the differential equation system  $\{x' = v, v' = -g\}$  expresses that the time-derivative of height  $x$  is the vertical velocity  $v$  whose time-derivative is  $-g$ . Of course, something keeps happening after the bouncing ball reverses its direction

because it hits the ground. Physics continues until it hits the ground again:

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

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

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

It is good that repetitions (\*) in HP are nondeterministic, because Quantum has no way of knowing ahead of time how many iterations of the control loop he will take. Yet, Quantum is now rather surprised. For if he follows 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 really succeeds to bounce back up quite as high as he was before. So there must have been something wrong with the model in (4.4), Quantum concludes, and sets out to fix (4.4).

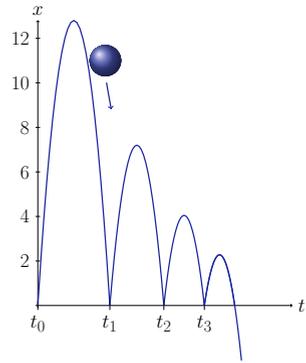
Having observed himself rather carefully when bouncing around for a while, Quantum concludes that he feels just a little bit slower when bouncing back up than he used to when falling down. Indeed, Quantum feels less energetic on his way up. So his velocity must not only flip direction from down to up at a bounce on the ground, 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:

$$\begin{aligned}
 & (\{x' = v, v' = -g\}; \\
 & \text{if}(x = 0) v := -cv)^*
 \end{aligned} \tag{4.5}$$

Now, if the ball is on the ground, its velocity will flip but its magnitude will shrink by damping factor  $c$ . 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 has a chance to fix his very own model of 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 from Chap. 2 that this purpose is exactly what evolution domains were meant for. Above ground is where he wants to remain, and so the evolution domain constraint  $x \geq 0$  is what Quantum asks dear physics to kindly obey, since the floor underneath Quantum is of rather sturdy build. Unlike in poor Alice's case [2], the floor comes without rabbit holes to fall through:

$$\begin{aligned}
 & (\{x' = v, v' = -g \ \& \ x \geq 0\}; \\
 & \text{if}(x = 0) v := -cv)^*
 \end{aligned} \tag{4.6}$$

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



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 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 exactly 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  $\dots \&x \geq 0$  prevents all ill-fated attempts to follow the differential equation below ground, because those would not satisfy  $x \geq 0$  all the time.

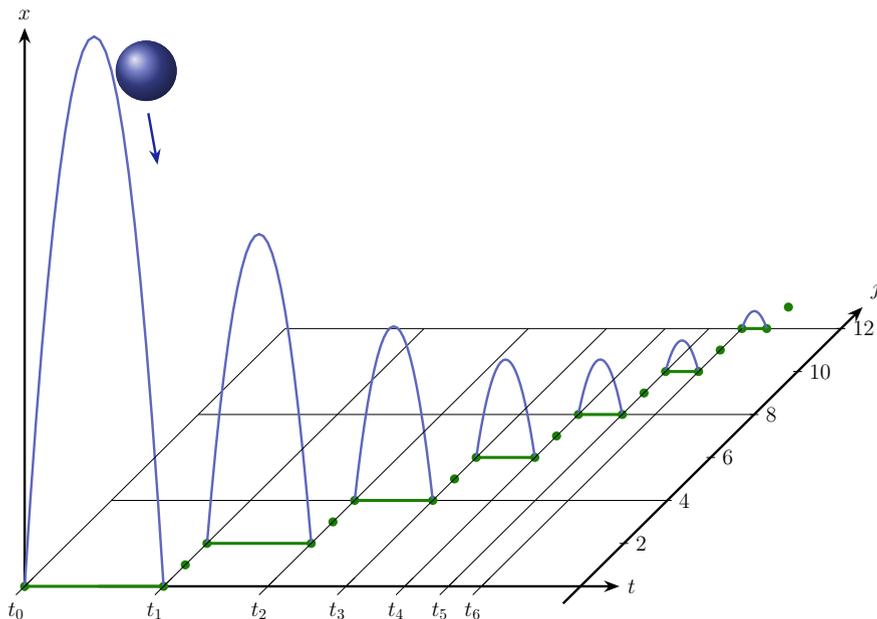
Being quite happy with model (4.6), the bouncing ball Quantum goes on to explore whether the model does what he expects it to do. Of course, had Quantum read this book already, he would have marched right into a rigorous analysis of the model. Since Quantum is still a CPS rookie, he takes a detour along 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 come out looking characteristically similar to Fig. 4.1.

### 4.2.2 *How Quantum Discovered a Crack in the Fabric of Time*

After a little while idly simulating his very own personal model, Quantum decides to take out his temporal magnifying glasses and zoom in really close to see what actually 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 records 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) runs, yet 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

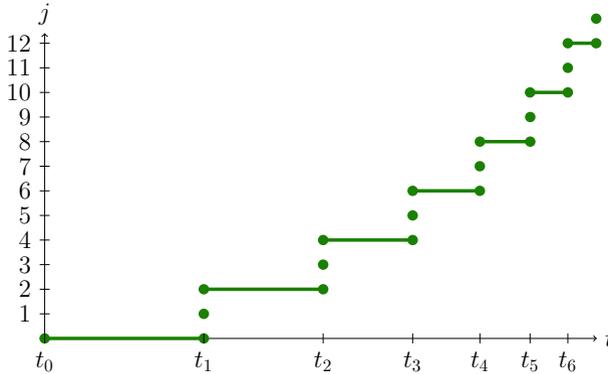
to give an extra natural number index  $j \in \mathbb{N}$  to distinguish the two successive occurrences of continuous time  $t_1$ . So, for the sake of illustration, he called  $(t_1, 0)$  the first point in time where Quantum was in state  $x = 0, v = -5$ , and then went on to call  $(t_1, 1)$  the second point in time where he was in state  $x = 0, v = 4$ .



**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  $t$

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 plotted 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 pale shadow of the execution in the  $x$  over  $t$  face, which agrees with the layman’s simulation shown in Fig. 4.1. 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.

Armed with the additional intuition about the operations of HP (4.6) that these sample executions in Fig. 4.1 and its hybrid version Fig. 4.3 provide, Quantum now



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

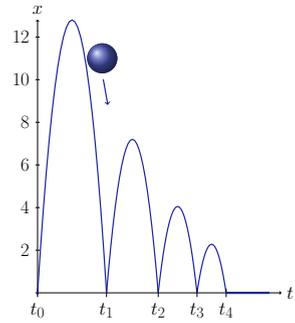
feels prepared to ask the deeper questions. Will his model (4.6) *always* do the right thing? Or was he just lucky in the particular simulation shown in Fig. 4.1 and magnified in Fig. 4.3? What even is the right behavior for a proper bouncing ball? What are its most important properties? And for what purpose even? How could they be specified unambiguously? And, ultimately, how can Quantum possibly convince himself that these properties are true before he ever bounces around again idly and at 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 the model does not even know how to lie still any more. Scrutinizing each and every simulation (Figs. 4.1 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 number 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 always runs out of steam and just lies flat on his back as in Fig. 4.5.

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 rôle of energy loss, Quantum suddenly has a pretty clever idea. Of course, all these models of deformations and frictions and energies and whatnot would be needed for a model of highly precise physics. Yet, if Quantum is merely trying to describe the

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



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 just with a nondeterministic choice ( $\cup$ ):

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

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 actual physics to do for him, such as never stopping bouncing 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 try an analysis of a more precise but also significantly more difficult physical model. Quantum now sees first hand how abstraction can create simplicity and will be sure to keep that in mind in all future endeavors.

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 an option to have the ball lie flat after some 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 already took a peek at Exercise 4.16, in which the more involved model (4.7) will be considered.

### 4.2.4 Postcondition Contracts for CPS

Having developed an elegant hybrid systems model of bouncing balls, Quantum subsequently proceeds to state his expectations about its behavior. Hybrid programs  $\alpha$  are useful models for CPS. They describe with a program the behavior of a CPS, ultimately captured by their semantics  $\llbracket \alpha \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$ , 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 such as running over a human being.

#### Expedition 4.1 (Three Laws of Robotics)

Safety of robots has been aptly defined by Isaac 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 a 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 than a proper CPS, he is still quite interested in his own safety. Quantum 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. Come to think of it, he would never want to jump any higher than he was in the very beginning. So, when  $H$  denotes initial height, Quantum 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 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, one for each of his expectations. For the time being, Quantum temporarily uses the following notation to indicate the two

expected postconditions that the bouncing ball HP is supposed to ensure:

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

Subsequent sections will quickly abandon this  $\mathbf{ensures}(F)$  notation in favor of a more elegant and 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  $\mathbf{ensures}()$  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 even dares to try another careless bounce ever again.

Quantum's deliberations eventually lead him to conclude that whether  $\mathbf{ensures}()$  contract in (4.8) works out will depend on the initial values that the bouncing ball starts out with. Quantum thinks of  $H$  as the initial height, but 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  $\mathbf{requires}(x = H)$  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

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

With this informal introduction to the need for and purpose of contracts in cyber-physical systems, it is about time for the development to become more general again. We will, thus, develop a systematic approach to CPS contracts before returning to a further study of whether the contracts expressed in Quantum's HP (4.9) really hold.

**Expedition 4.2 (Invariant contracts for CPS)**

In addition to preconditions and postconditions, loop invariants play a prominent rôle in contracts for conventional imperative programs, because they constitute the major logical mode for understanding loops. 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 a 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++;
  }
```

Dijkstra's algorithm for computing the greatest common divisor of  $a$  and  $b$  needs a loop invariant and a precondition, as  $\text{gcd}(5, 0)$  would not terminate:

```
// requires(x!=0 && y!=0)
x=a; y=b; u=b; v=a;
while (x!=y)
  // loop_invariant(2*a*b == u*x + v*y)
  {
    if (x>y) {
      x=x-y; v=v+u;
    } else {
      y=y-x; u=u+v;
    }
  }
```

Such loop invariants will also play an equally important rôle 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 rôle 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 CPSs have more stringent requirements on safety.

However, we do not only want to program CPSs, 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 was 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, instead, a logic for precise statements and reasoning about their dynamical systems. So CPSs need logics of dynamical systems [25, 29], of which the most fundamental representative is *differential dynamic logic* (dL) [20–22, 25, 26, 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, and still forms the basis for Parts III and IV. Additional multi-dynamical systems aspects beyond hybrid systems are discussed elsewhere [23–25, 28, 29, 31], some of which will be picked up in Part III of this textbook.

The most important feature of differential dynamic logic for our purposes is that it allows us to refer to hybrid systems. Chapter 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 with 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 CPSs, because it allows us to refer to real-valued quantities such as 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 the relation  $\llbracket \alpha \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$  is what ultimately constitutes the semantics of HP  $\alpha$ . It defines which new state  $\nu \in \mathcal{S}$  is reachable from which initial state  $\omega \in \mathcal{S}$  in HP  $\alpha$ , in which case we write  $(\omega, \nu) \in \llbracket \alpha \rrbracket$ .

**Note 21 (Differential dynamic logic principle)** Differential dynamic logic, which is 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  $\llbracket \alpha \rrbracket$ , parametrized by HP  $\alpha$ , which refers to all states reachable by this HP  $\alpha$  according to the reachability relation  $\llbracket \alpha \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$  of its semantics. For any HP  $\alpha$ , this modal

operator  $[\alpha]$  can be placed in front of any **dL** formula  $P$ . The resulting **dL** formula

$$[\alpha]P$$

expresses that *all* states reachable by HP  $\alpha$  satisfy formula  $P$ .

The logic **dL** also provides another modal operator  $\langle \alpha \rangle$ , parametrized by HP  $\alpha$ , 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  $\alpha$  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  $\alpha$ , since they refer to all or some runs of  $\alpha$ . The formula  $[\alpha]P$  is pronounced “ $\alpha$  box  $P$ ” and  $\langle \alpha \rangle P$  is “ $\alpha$  diamond  $P$ .”

With the help of **dL**’s modalities, an **ensures**( $E$ ) postcondition for an HP  $\alpha$  can be expressed directly as a logical formula in differential dynamic logic:

$$[\alpha]E$$

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

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

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:

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

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 consisting of the logical conjunction of (4.10) and (4.11):

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

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:

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

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:

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

The equivalence of  $[\alpha]P \wedge [\alpha]Q$  and  $[\alpha](P \wedge Q)$  can in turn be expressed as a logical formula with the equivalence operator ( $\leftrightarrow$ ) giving a formula that is true in all states:

$$[\alpha]P \wedge [\alpha]Q \leftrightarrow [\alpha](P \wedge Q)$$

This equivalence will be investigated in more detail in a later chapter, but it is useful to observe it now already in order to sharpen our intuition about **dL** and anticipate possible use cases for its flexibility.

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 (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 gradual motivation and informal introduction in this chapter, this section now defines *differential dynamic logic* [20–22, 25, 26, 30], which plays a central rôle 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 on p. 42 and the hybrid programs from Sect. 3.3.1 on p. 76.

### 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  $\alpha$ . The **dL** formula  $[\alpha]P$  expresses that all states after all runs of HP  $\alpha$  satisfy **dL** formula  $P$ . The **dL** formula  $\langle\alpha\rangle P$  expresses that there is a run of HP  $\alpha$  that leads to a state in which **dL** formula  $P$  is true. In the *modal formulas*  $[\alpha]P$  and  $\langle\alpha\rangle P$ , the formula  $P$  is called the *postcondition*.

**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  $\alpha$  is an HP):

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

Operators  $>, \leq, <, \leftrightarrow$  are definable, 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 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, because the new value 5 that  $x$  assumes is, indeed, positive. 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, in turn, says that after incrementing  $x$ , it will have become positive. The implication is trivially true in all states falsifying the assumption  $x \geq 0$ , because implications are *true* if their left-hand side is *false*.

The program  $x := x + 1$  can only be run in exactly one way, so quantifying over all runs of  $x := x + 1$  in the box modality  $[x := x + 1]$  is indistinguishable from quantifying over one run of  $x := x + 1$  in the diamond modality  $\langle x := x + 1 \rangle$ . 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, then there is a value for variable  $d$ , 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  $x$ , e.g., 1 and a value for  $d$ , e.g., 2, such that  $x$  stays positive always after any continuous evolution along  $x' = d$  starting from that initial value of  $x$ . 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 an initial value for  $x$ , e.g.,  $-1$ , and a  $d$ , e.g.,  $-2$ , such that  $x$  always stays negative along  $x' = d$ . In this case, different values have to be chosen for  $x$  and  $d$  in the two conjuncts to make the whole formula evaluate to *true*, but that is perfectly allowed for different quantifiers in different places. 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 differential equation  $x' = 2$  for any amount of time keeps  $x$  nonnegative.

The box modality  $[\alpha]$  in the dL formula  $[\alpha]P$  expresses that its postcondition  $P$  is true after *all* runs of HP  $\alpha$ . 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  $\alpha$ . Even if the diamond modality is not yet 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 or otherwise positive. In fact,  $\langle x' = d \rangle x > 0$  is equivalent to  $x > 0 \vee d > 0$ . In particular,  $\exists d \langle x' = d \rangle x > 0$  is valid, because there is a choice of  $d$  for which  $x$  will eventually be positive after following the ODE  $x' = d$  long enough. Even the following conjunction is valid:  $\exists d \langle x' = d \rangle x > 0 \wedge \exists d \langle x' = d \rangle x < 0$ , 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  $d$  will ultimately get  $x$  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. All it takes is sufficient patience. In fact, for the same reason, even the following dL formula is valid:

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

It expresses that, from any positive initial value of  $x$ , any duration of following  $x' = 2$  will lead to a state in which  $x$  is still positive but for which it is also still possible to follow  $x' = -2$  for some (larger) amount of time to make the final value of  $x$  zero.

Before you get yourself confused, 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 new 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 <i>true</i> or <i>false</i>
$?x = e$	Testing $x$ for equality comparison in an HP, only continue if <i>true</i>

#### 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

**Expedition 4.3 (Operator precedence for differential dynamic logic)**

To save parentheses, the notational conventions have unary operators (including<sup>a</sup>  $\neg$ , quantifiers  $\forall x, \exists x$ , modalities  $[\alpha], \langle \alpha \rangle$  as well as HP operator  $*$ ) bind more strongly than binary operators. We let  $\wedge$  bind more strongly than  $\vee$ , which binds more strongly than  $\rightarrow, \leftrightarrow$ , and let  $;$  bind more strongly 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 strongly, 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 imply  $\alpha; \beta \cup \gamma \equiv (\alpha; \beta) \cup \gamma$  and  $\alpha \cup \beta; \gamma \equiv \alpha \cup (\beta; \gamma)$  and  $\alpha; \beta^* \equiv \alpha; (\beta^*)$  like in regular expressions. All logical and program operators associate to the right, 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  $P \rightarrow Q \leftrightarrow R$  is 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  $(P \leftrightarrow Q) \rightarrow R$ .

<sup>a</sup> It is debatable whether quantifiers are unary operators:  $\forall x$  is a unary operator on formulas but  $\forall$  is an operator with mixed arguments (one variable and one formula). In higher-order logic with  $\lambda$ -abstractions  $\lambda x.P$  for the function that maps  $x$  to  $P$ , the operator  $\forall$  can be understood by considering  $\forall x P$  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 convention is that it mnemonically simplifies the precedence rules.

real arithmetic. The semantics in Chap. 2 inductively defined the satisfaction relation  $\omega \models P$ , which holds iff formula  $P$  is true in state  $\omega$ , and then collected the set of states  $\llbracket P \rrbracket$  in which  $P$  is true. Now, we define the semantics of dL formulas right away by simultaneously defining the set of states  $\llbracket P \rrbracket$  in which formula  $P$  is true. The two styles of definition are equivalent (Exercise 4.13), but the latter is more convenient here.

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

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

1.  $\llbracket e = \tilde{e} \rrbracket = \{ \omega : \omega[e] = \omega[\tilde{e}] \}$

That is, an equation is true in the set of states  $\omega$  in which the terms on the two sides evaluate to the same real number according to Definition 2.4.

2.  $\llbracket e \geq \tilde{e} \rrbracket = \{ \omega : \omega[e] \geq \omega[\tilde{e}] \}$

That is, a greater-or-equals inequality is true in states  $\omega$  where the term on

the left evaluates to a number that is greater than or equal to that on the right.

$$3. \llbracket \neg P \rrbracket = (\llbracket P \rrbracket)^c = \mathcal{S} \setminus \llbracket P \rrbracket$$

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. So  $\neg P$  is true iff  $P$  is false.

$$4. \llbracket P \wedge Q \rrbracket = \llbracket P \rrbracket \cap \llbracket Q \rrbracket$$

That is, a conjunction is true in the intersection of the states where both conjuncts are true. So  $P \wedge Q$  is true iff  $P$  and  $Q$  are true.

$$5. \llbracket P \vee Q \rrbracket = \llbracket P \rrbracket \cup \llbracket Q \rrbracket$$

That is, a disjunction is true in the union of the set of states where either of its disjuncts is true. So  $P \vee Q$  is true iff  $P$  or  $Q$  (or both) are true.

$$6. \llbracket P \rightarrow Q \rrbracket = \llbracket P \rrbracket^c \cup \llbracket Q \rrbracket$$

That is, an implication is true in the states where its left-hand side is false or its right-hand side is true. So  $P \rightarrow Q$  is true iff  $P$  is false or  $Q$  is true.

$$7. \llbracket P \leftrightarrow Q \rrbracket = (\llbracket P \rrbracket \cap \llbracket Q \rrbracket) \cup (\llbracket P \rrbracket^c \cap \llbracket Q \rrbracket^c)$$

That is, a bi-implication is true in the states where both sides are true or both sides are false. So  $P \leftrightarrow Q$  is true iff  $P, Q$  are both false or both true.

$$8. \llbracket \forall x P \rrbracket = \{ \omega : v \in \llbracket P \rrbracket \text{ for all states } v \text{ that agree with } \omega \text{ except on } x \}$$

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  $x$ .

$$9. \llbracket \exists x P \rrbracket = \{ \omega : v \in \llbracket P \rrbracket \text{ for some state } v \text{ that agrees with } \omega \text{ except on } x \}$$

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  $x$ .

$$10. \llbracket [\alpha] P \rrbracket = \{ \omega : v \in \llbracket P \rrbracket \text{ for all states } v \text{ such that } (\omega, v) \in \llbracket \alpha \rrbracket \}$$

That is, a box modal formula  $[\alpha]P$  is true in state  $\omega$  iff its postcondition  $P$  is true in all states  $v$  that are reachable by running  $\alpha$  from  $\omega$ .

$$11. \llbracket \langle \alpha \rangle P \rrbracket = \llbracket \alpha \rrbracket \circ \llbracket P \rrbracket = \{ \omega : v \in \llbracket P \rrbracket \text{ for some state } v \text{ with } (\omega, v) \in \llbracket \alpha \rrbracket \}$$

That is, diamond modal formula  $\langle \alpha \rangle P$  is true in state  $\omega$  iff its postcondition  $P$  is true in at least one state  $v$  that is reachable by running  $\alpha$  from  $\omega$ .

If  $\omega \in \llbracket P \rrbracket$ , then we say that  $P$  is true in state  $\omega$ . The literature sometimes also uses the *satisfaction relation* notation  $\omega \models P$  synonymously for  $\omega \in \llbracket P \rrbracket$ . A formula  $P$  is *valid*, written  $\models P$ , iff it is true in all states, i.e.,  $\llbracket P \rrbracket = \mathcal{S}$ , so  $\omega \in \llbracket P \rrbracket$  for all states  $\omega$ . A formula  $P$  is a *consequence* of a set of formulas  $\Gamma$ , written  $\Gamma \models P$ , iff, for each state  $\omega$ : If  $(\omega \in \llbracket Q \rrbracket)$  for all  $Q \in \Gamma$  then  $\omega \in \llbracket P \rrbracket$ .

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  $v_i$  reachable by  $\alpha$  from initial state  $\omega$  relates to the truth of  $[\alpha]P$  or  $\langle \alpha \rangle P$  at state  $\omega$ .

In a state where HP  $\alpha$  cannot run, the formula  $[\alpha]P$  is (vacuously) true, because all states reached after following  $\alpha$  satisfy the formula  $P$  simply because there are no such states. For example,  $[?x \geq 5]false$  is exactly true in all states in which the value of  $x$  is less than 5, because the HP  $?x \geq 5$  cannot execute successfully at all then, because it fails the test. This is to be contrasted with the formula  $\langle ?x \geq 5 \rangle false$ ,

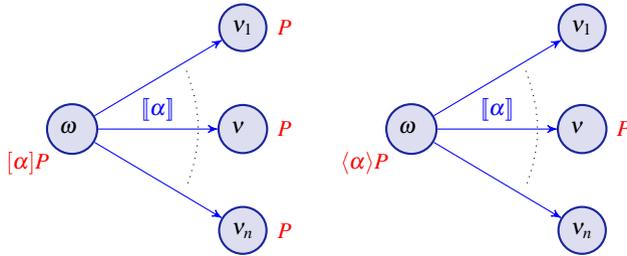


Fig. 4.6 Transition semantics of modalities in dL formulas

**Expedition 4.4 (Set-valued dL semantics  $\llbracket \cdot \rrbracket : \text{Fml} \rightarrow \wp(\mathcal{S})$ )**

The semantics of a term  $e$  directly defines the real-valued function  $\llbracket e \rrbracket : \mathcal{S} \rightarrow \mathbb{R}$  from states to the real value that the term evaluates to in that state (Expedition 2.2 on p. 47). Similarly, Definition 4.2 directly defines inductively, for each dL formula  $P$ , the set of states, written  $\llbracket P \rrbracket \subseteq \mathcal{S}$ , in which  $P$  is true:

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

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

which is not true in any state, because it claims the existence of an execution of HP  $?x \geq 5$  to a state where *false* holds true, which never holds. Contrast this with the formula  $\langle ?x \geq 5 \rangle x < 7$ , which is true exactly in all states satisfying  $x \geq 5$  and  $x < 7$ .

## 4.5 CPS Contracts in Logic

Now that we know what truth and validity are in differential dynamic logic, let's go back to the previous question. Is dL formula (4.12) valid? Is (4.13) valid? Actually, let's first ask whether they are equivalent, i.e., whether the dL formula

(4.12)  $\leftrightarrow$  (4.13) is valid. Expanding the abbreviations this is the question of whether the following dL formula is valid:

$$\begin{aligned} & \left( \left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x=0) \ v := -cv \right)^* \right] 0 \leq x \right. \\ & \left. \wedge \left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x=0) \ v := -cv \right)^* \right] x \leq H \right) \quad (4.15) \\ \leftrightarrow & \left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x=0) \ v := -cv \right)^* \right] (0 \leq x \wedge x \leq H) \end{aligned}$$

Exercise 4.1 gives you an opportunity to convince yourself that the equivalence (4.12)  $\leftrightarrow$  (4.13) is indeed valid.<sup>1</sup> 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.

Certainly, (4.12) is not true in a state  $\omega$  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  $\omega$  in which  $0 \leq x$  is still false. The initial state is a possible final state for any HP of the form  $\alpha^*$ , because it can repeat 0 times. Thus, (4.12) only has a chance of being true 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 rôle than postconditions do. Postconditions of HP  $\alpha$  are expected to be true after every run of  $\alpha$ , 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 additional logical operator to express preconditions?

The meaning of precondition **requires**( $A$ ) of HP  $\alpha$  is that  $A$  is assumed to hold before the HP starts. *If*  $A$  holds when  $\alpha$  starts, *then* its postcondition **ensures**( $B$ ) holds after all runs of HP  $\alpha$ . 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 in 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 an HP  $\alpha$  promises that  $B$  will always hold after running  $\alpha$  if  $A$  was true initially when  $\alpha$  started. Thus, the meaning of a precondition can be expressed easily using an implication

$$A \rightarrow [\alpha]B \quad (4.16)$$

<sup>1</sup> This equivalence also foreshadows the fact that CPS provides ample opportunity for questions about how multiple system models relate to one another. The dL formula (4.15) relates three different properties of three occurrences of one and the same hybrid program, for example. The need to relate different properties of different CPSs will arise frequently throughout this book 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.

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  $\omega$  in which precondition  $A$  holds ( $\omega \in \llbracket A \rrbracket$ ), it is the case that all runs of HP  $\alpha$  lead to states  $\nu$  (with  $(\omega, \nu) \in \llbracket \alpha \rrbracket$ ) in which postcondition  $B$  holds (so  $\nu \in \llbracket B \rrbracket$ ). By the nature of implication, the dL formula (4.16) does not say what happens in states  $\omega$  in which the precondition  $A$  does not hold (so  $\omega \notin \llbracket A \rrbracket$ ).

How does formula (4.16) talk about the runs of an HP and postcondition  $B$  again? Recall that the dL formula  $[\alpha]B$  is true in exactly those states in which all runs of HP  $\alpha$  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  $\alpha$  with a CPS contract using a single **requires**( $A$ ) precondition and single **ensures**( $B$ ) postcondition:

$$\begin{array}{l} \mathbf{requires}(A) \\ \mathbf{ensures}(B) \\ \alpha \end{array}$$

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

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

dL formulas of this shape are very common and correspond to *Hoare triples* [14] but for hybrid systems instead of conventional programs. Hoare triples, in turn, are modeled after Aristotle's syllogisms.

CPS contracts with multiple preconditions and multiple postconditions can directly be expressed as dL formulas as well (Exercise 4.5).

Recall HP (4.9), which is shown here with both preconditions combined into one joint precondition and both postconditions combined into one postcondition:

$$\begin{array}{l} \mathbf{requires}(0 \leq x \wedge x = H) \\ \mathbf{ensures}(0 \leq x \wedge x \leq H) \\ (\{x' = v, v' = -g \ \& \ x \geq 0\}; \\ \text{if}(x = 0) v := -cv)^* \end{array} \quad (4.17)$$

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

$$0 \leq x \wedge x = H \rightarrow [(\{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x = 0) v := -cv)^*] (0 \leq x \wedge x \leq H) \quad (4.18)$$

So to find out whether (4.17) satisfies its CPS contract, we need to ask whether the corresponding dL formula (4.18) is valid. In other words, dL gives CPS contracts an unambiguous meaning.

We will need some operational way that allows us to tell whether such a dL formula is valid, i.e., true in all states, because mere inspection of the semantics

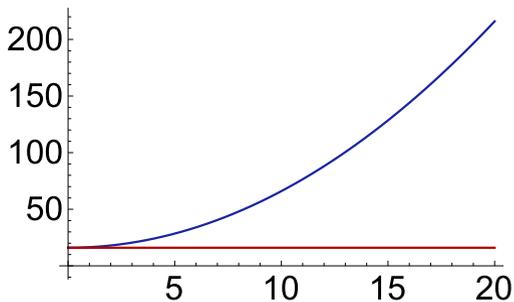
alone is not a particularly scalable way of approaching validity questions. Such an operational way of determining the validity of **DL** formulas by proof will be pursued in the next chapter.

### 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 will 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 before.

Maybe the first thing to notice is that the HP mentions  $g$ , which is meant to represent the standard gravitational constant, but the formula (4.18) never actually says that. Certainly, if gravity were negative ( $g < 0$ ), bouncing balls would function rather differently in quite an astonishing way. They would suddenly become 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  $g = 9.81$ :

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

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 is what gravity is all 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 CPSs is that their behavior is subject to parameters, which can have quite a non-negligible impact. It is very hard to determine precise values for all parameters by measurements. When one particular concrete numeric 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 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  $g > 0$ ). This 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  $g = 9.81$ . 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.

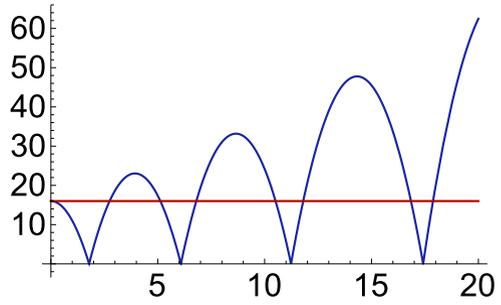
In light of these thoughts, we might assume  $9 < g < 10$  to be the gravitational constant for Earth. Yet, we can also just consider all bouncing balls on all planets in the solar system or elsewhere at once by assuming only  $g > 0$  instead of  $g = 9.81$  as in (4.19), since this is the only aspect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

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

Do we expect dL formula (4.20) to be valid, i.e., true in all states? What could possibly 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.

Being a damping factor, we also expect  $c \geq 0$  (despite Exercise 4.15). Yet, (4.20) really only has a chance of being true when we assume that  $c$  is not too big:

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



$$0 \leq x \wedge x = H \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow$$

$$\left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x = 0) v := -cv \right)^* \right] (0 \leq x \wedge x \leq H) \quad (4.21)$$

Is (4.21) valid now? Or does its truth still depend on assumptions that have not been identified yet? 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.

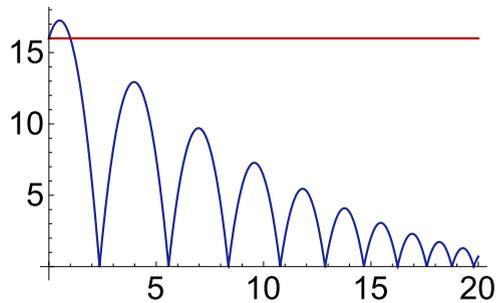
Now, all parameters  $(H, g, c)$  have some assumptions in (4.21), which is a good thing. But what about velocity variable  $v$ ? Why is there no assumption about it yet? Should there be one? Unlike  $g$  and  $c$ , velocity  $v$  changes over time. What is its initial value allowed to be? What could go wrong?

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:

$$0 \leq x \wedge x = H \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow$$

$$\left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x = 0) v := -cv \right)^* \right] (0 \leq x \wedge x \leq H) \quad (4.22)$$

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

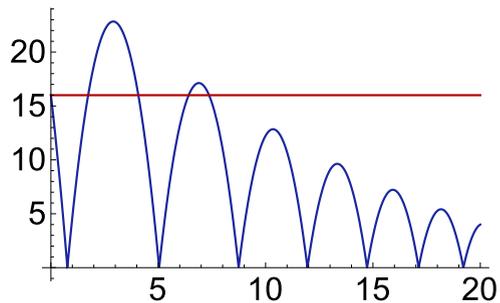


Now there finally are assumptions about all parameters and variables of (4.22). That does not mean that we found the right assumptions! But it is still a good sanity check. Before wasting cycles on trying to prove or otherwise justify (4.22), let's see once more whether we can find an initial state 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

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

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 that 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. It certainly is a good idea to prove **dL** formulas about more subtle CPS models, too.

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 4.17):

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

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):

$$\begin{aligned} &\mathbf{requires}(0 \leq x \wedge x = H \wedge v = 0) \\ &\mathbf{requires}(g > 0 \wedge 1 \geq c \geq 0) \\ &\mathbf{ensures}(0 \leq x \wedge x \leq H) \\ &(\{x' = v, v' = -g \& x \geq 0\}; \\ &\quad \mathbf{if}(x = 0) v := -cv)^* \end{aligned} \quad (4.25)$$

Observing the non-negligible difference between the original conjecture (4.19) and the revised and improved conjecture (4.24) leads us to adopt the principle of Cartesian Doubt from Expedition 4.5.

## 4.7 Summary

This chapter introduced differential dynamic logic (**dL**), whose operators and their informal meaning is summarized in Table 4.1. Their precise semantics was reported in Sect. 4.4.2 on p. 112. The most important aspect of differential dynamic logic is its ability to directly refer to possible or necessary reachability properties of hybrid programs. The fact that it is possible for hybrid program  $\alpha$  to reach a state where formula  $P$  is true is directly expressed by the **dL** formula  $\langle \alpha \rangle P$ . That HP  $\alpha$  necessarily only leads to (final) states where formula  $P$  is true is expressed by  $[\alpha]P$ . Differential dynamic logic is closed under all operators, so HP modalities, proposi-

**Expedition 4.5 (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 whether 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.

tional connectives, and quantifiers can be nested arbitrarily often, leading to a pretty flexible specification language for CPS. Beyond its capability to capture the meaning of CPS contracts rigorously, subsequent chapters will develop the specification logic dL into a verification logic that can be used to prove dL specifications.

For future chapters, we should also keep the bouncing ball example and its surprising subtleties in mind.

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

**4.8 Appendix**

This appendix already features some first reasoning aspects of CPS even if a fully systematic account of CPS reasoning will be pursued from scratch in more elegant

ways in subsequent chapters, one operator at a time. Especially for readers who have seen the Floyd-Hoare calculus for conventional programs [5, 14] or who prefer to 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:

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

Removing the loop clearly changes the behavior of the bouncing ball quite radically. It no longer bounces particularly well. All it can do now is fall and, if it reaches the floor, have its velocity reversed 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 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  $\alpha$  in a sequential composition  $\alpha; \beta$  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  $\alpha$  and initial states for  $\beta$ . The intermediate states of  $\alpha; \beta$  are the states  $\mu$  in the semantics  $\llbracket \alpha; \beta \rrbracket$  from Chap. 3:

$$\llbracket \alpha; \beta \rrbracket = \llbracket \alpha \rrbracket \circ \llbracket \beta \rrbracket = \{(\omega, \nu) : (\omega, \mu) \in \llbracket \alpha \rrbracket, (\mu, \nu) \in \llbracket \beta \rrbracket \text{ for some } \mu\}$$

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

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

This answer can be found by integrating or solving the differential equations (Example 2.4 on p. 34). 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).<sup>2</sup> 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

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

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

$$2gx = 2gH - v^2 \quad (4.28)$$

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, thus, decompose our reasoning into two parts. The first part will prove that the intermediate condition (4.28) indeed 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 an HP that is a sequential composition  $\alpha; \beta$  an *intermediate condition* is a formula that characterizes the intermediate states in between HPs  $\alpha$  and  $\beta$ . That is, for a dL formula

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

<sup>2</sup> Following these thoughts a bit further reveals how (4.27) can actually be used perfectly well to describe intermediate states when changing 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.

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

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

The first dL formula expresses that intermediate condition  $E$  characterizes the intermediate states accurately, i.e.,  $E$  actually holds after all runs of HP  $\alpha$  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  $\beta$  end up in  $B$ . That is, from all states satisfying  $E$  (including those that result by running  $\alpha$  from a state satisfying  $A$ ),  $B$  holds after all runs of  $\beta$ .

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

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

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 precondition of this second formula. So the second formula of (4.29) is not valid. How can this problem be resolved? By adding  $0 \leq x$  to the intermediate condition (4.28), thus, requiring us to prove these two formulas:

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

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

$$A \rightarrow [\alpha \cup \beta]B \quad (4.31)$$

Recalling its semantics from Chap. 3,

$$\llbracket \alpha \cup \beta \rrbracket = \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket$$

HP  $\alpha \cup \beta$  has two possible behaviors. It can run as HP  $\alpha$  does or as HP  $\beta$  does. And it is chosen nondeterministically which of the two behaviors happens. Since the behavior of  $\alpha \cup \beta$  can be either  $\alpha$  or  $\beta$ , proving (4.31) requires proving  $B$  to hold after  $\alpha$  and after  $\beta$ . More precisely, (4.31) assumes  $A$  to hold initially, otherwise (4.31) is vacuously true. Thus, proving (4.31) allows us to assume  $A$  and requires us to prove that  $B$  holds after all runs of  $\alpha$  (which is permitted behavior for  $\alpha \cup \beta$ ) and to prove that, assuming  $A$  holds initially,  $B$  holds after all runs of  $\beta$  (which is also permitted behavior of  $\alpha \cup \beta$ ).

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

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

by proving the following dL formulas:

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

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

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

### 4.8.3 A Proof of Tests

Consider the second formula of (4.32). 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

$$\llbracket ?Q \rrbracket = \{(\omega, \omega) : \omega \in \llbracket Q \rrbracket\} \quad (4.33)$$

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

$$A \rightarrow [?Q]B \quad (4.34)$$

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 of  $?Q$  from state  $\omega$  if and only if  $Q$  holds in  $\omega$ . So the only cases to worry about are initial states that satisfy  $Q$ ; otherwise, the HP in (4.34) cannot execute 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

$$A \wedge Q \rightarrow B$$

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

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

by proving the following dL formula:

$$A \wedge Q \rightarrow B$$

Using this for the second formula of (4.32), Note 27 reduces proving the second formula of (4.32)

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

to proving

$$2gx = 2gH - v^2 \wedge x \geq 0 \wedge x \neq 0 \rightarrow 0 \leq x \wedge x \leq H \quad (4.35)$$

Now we are left with arithmetic that we need to prove. Proofs for arithmetic and propositional logical operators such as  $\wedge$  and  $\rightarrow$  will be considered in a later chapter. For now, we notice that the formula  $0 \leq x$  in the right-hand side of  $\rightarrow$  is justified by assumption  $x \geq 0$  if we flip the inequality around. Yet,  $x \leq H$  does not follow from the left-hand side of (4.35), because we lost our 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 CPSs or of any other mathematical statements. One of our intermediate steps might have been too weak, so that our attempt to prove the property 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 (4.32). 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 intuition. Consequently, we will investigate what holds all arguments together and what constitutes an actual proof in Chap. 6, 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 can help us prove it. Also, the assignment in (4.32) still needs to be handled and its sequential composition needs an intermediate contract (Exercise 4.18). Both will be pursued in the next chapter, where we move to a much more systematic and rigorous reasoning style for CPS.

## Exercises

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

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

**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, too. 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? Why?

**4.3.** Let  $A, B$  be dL formulas. Suppose  $A \leftrightarrow B$  is true in state  $\omega$  and  $A$  is true in state  $\omega$ . That is,  $\omega \in \llbracket A \leftrightarrow B \rrbracket$  and  $\omega \in \llbracket A \rrbracket$ . Is  $B$  true in state  $\omega$ ? Prove or disprove. Is  $B$  valid? Prove or disprove.

**4.4.** Let  $\alpha$  be an HP and let  $\omega$  be a state. Prove or disprove each of the following cases:

1. If  $\omega \notin \llbracket P \rrbracket$  then does  $\omega \notin \llbracket [\alpha^*]P \rrbracket$  have to hold?
2. If  $\omega \notin \llbracket P \rrbracket$  then does  $\omega \notin \llbracket \langle \alpha^* \rangle P \rrbracket$  have to hold?
3. If  $\omega \in \llbracket P \rrbracket$  then does  $\omega \in \llbracket [\alpha^*]P \rrbracket$  have to hold?
4. If  $\omega \in \llbracket P \rrbracket$  then does  $\omega \in \llbracket \langle \alpha^* \rangle P \rrbracket$  have to hold?

**4.5 (Multiple pre/postconditions).** Suppose you have an HP  $\alpha$  with a CPS contract using multiple preconditions  $A_1, \dots, A_n$  and multiple postconditions  $B_1, \dots, B_m$ :

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

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

**4.6 (Late contracts).** dL formula (4.18) represents the canonical way of turning the precondition of a contract into an implication and putting the postcondition after the modality. There are other ways of capturing contract (4.17) as a dL formula. The following formula initially only assumes  $x = H$  but has an implication as a postcondition. Is it also a correct logical rendition of contract (4.17)?

$$x = H \rightarrow \left[ \left( \{x' = v, v' = -g \ \& \ x \geq 0\}; \text{if}(x = 0) v := -cv \right)^* \right] (0 \leq H \rightarrow 0 \leq x \wedge x \leq H) \quad (4.36)$$

**4.7 (Systematically late contracts).** This question compares canonical and late phrasings of precondition/postcondition contracts by shifting formulas from preconditions to postconditions. For simplicity assume that  $x$  is the only variable that  $\alpha$  modifies. Are the following two dL formulas equivalent?

$$\begin{array}{l} A(x) \rightarrow [\alpha]B(x) \\ x = x_0 \rightarrow [\alpha](A(x_0) \rightarrow B(x)) \end{array}$$

For example, is (4.36) logically equivalent to (4.18)?

**4.8.** For each of the following dL formulas, determine whether 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.  $[(x' = 1 \& false)^*]false$ .
16.  $x \geq 0 \rightarrow [x' = v, v' = a]x \geq 0$ .
17.  $x > 0 \rightarrow [x' = x^2]x > 0$ .
18.  $x > 0 \rightarrow [x' = y]x > 0$ .
19.  $x > 0 \rightarrow [x' = x]x > 0$ .
20.  $x > 0 \rightarrow [x' = -x]x > 0$ .

**4.9.** For each of the following dL formulas, determine whether 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.  $x > 0 \rightarrow [(x := x + 1)^*]x > 0$
5.  $x > 0 \rightarrow [(x := x + 1)^*]x > 1$
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.  $x = 0 \rightarrow [x' = 1; x' = -2]x < 0$ .
10.  $x \geq 0 \wedge v \geq 0 \rightarrow [x' = v, v' = 2]x \geq 0$ .

**4.10.** For each of the following dL formulas, determine whether 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.  $x > 0 \rightarrow \langle x' = 1 \rangle x > 0$
5.  $[(x := x + 1)^*] \langle x' = -1 \rangle x < 0$ .
6.  $x > 0 \rightarrow [x' = 2](x > 0 \wedge [x' = 1]x > 0 \wedge \langle x' = -2 \rangle x = 0)$
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.  $\forall x \exists d (x \geq 0 \rightarrow [x' = d] x \geq 0)$
12.  $\forall x (x \geq 0 \rightarrow \exists d [x' = d] x \geq 0)$
13.  $[x' = 1] (x \geq 0 \rightarrow [x' = 2] x \geq 0)$
14.  $[x' = 1] x \geq 0 \rightarrow [x' = 2] x \geq 0$
15.  $[x' = 2] x \geq 0 \rightarrow [x' = 1] x \geq 0$
16.  $\langle x' = 2 \rangle x \geq 0 \rightarrow [x' = 1] x \geq 0$

**4.11.** For each  $j, k \in \{\text{satisfiable, unsatisfiable, valid}\}$  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.** Replace  $\alpha$  with a concrete HP that makes the following dL formulas valid or explain why such an HP does not exist. For an extra challenge do not use assignments in  $\alpha$ .

$$\begin{aligned}
& [\alpha] \text{false} \\
& [\alpha^*] \text{false} \\
& [\alpha] x > 0 \leftrightarrow \langle \alpha \rangle x > 0 \\
& [\alpha] x > 0 \leftrightarrow [\alpha] x > 1 \\
& [\alpha] x > 0 \leftrightarrow \neg[\alpha \cup \alpha] x > 0 \\
& [\alpha] x = 1 \wedge [\alpha] x = 2
\end{aligned}$$

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

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

The other way is to directly inductively define, for each dL formula  $P$ , the set of states, written  $\llbracket P \rrbracket$ , in which  $P$  is true. Its definition includes, among other cases, the following:

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

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

Such a proof can be conducted by induction on the structure of  $P$ . That is, you consider each case, say  $P \wedge Q$ , and prove  $\omega \models P \wedge Q$  iff  $\omega \in \llbracket P \wedge Q \rrbracket$  from the inductive hypothesis that the conjecture already holds for the smaller subformulas:

$$\begin{aligned}\omega \models P &\text{ iff } \omega \in \llbracket P \rrbracket \\ \omega \models Q &\text{ iff } \omega \in \llbracket Q \rrbracket\end{aligned}$$

**4.14 (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:

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

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?

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

**4.15.** 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  $c = 1$ ?

**4.16 (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:

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

**4.17.** 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.

**4.18 (\*).** Find an intermediate condition for proving the first formula in (4.32). 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?

**4.19 (\*\*).** Sect. 4.8.1 used a mix of systematic and ad hoc approaches to produce an intermediate condition that was based on solving and combining differential equations. Can you think of a more systematic rephrasing?

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

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

by identifying an intermediate condition  $E$  and showing

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

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 a clever intermediate condition  $E$ ?

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

**4.22 (Direct velocity control).** Real cars have proper acceleration and braking. What would happen if cars had direct control of velocity with instantaneous effect? Your job is to fill in the blanks with a test condition that makes sure the car with position  $x$  and velocity  $v$  and reaction time  $\varepsilon$  cannot exceed the stoplight  $m$ .

$$\begin{aligned}
 x \leq m \wedge V \geq 0 \rightarrow & \\
 & [((? \underline{\hspace{2cm}}; v := V \cup v := 0); \\
 & \quad t := 0; \\
 & \quad \{x' = v, t' = 1 \ \& \ t \leq \varepsilon\} \\
 & \quad )^*] x \leq m
 \end{aligned}$$

## References

- [1] Isaac Asimov. Runaround. *Astounding Science Fiction* (Mar. 1942).
- [2] Lewis Carroll. *Alice's Adventures in Wonderland*. London: Macmillan, 1865.
- [3] Patricia Derler, Edward A. Lee, Stavros Tripakis, and Martin Törngren. Cyber-physical system design contracts. In: *ICCPs*. Ed. by Chenyang Lu, P. R. Kumar, and Radu Stoleru. New York: ACM, 2013, 109–118. DOI: [10.1145/2502524.2502540](https://doi.org/10.1145/2502524.2502540).
- [4] René Descartes. *Meditationes de prima philosophia, in qua Dei existentia et animae immortalitas demonstratur*. 1641.
- [5] Robert W. Floyd. Assigning meanings to programs. In: *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*. Ed. by J. T. Schwartz. Vol. 19. Providence: AMS, 1967, 19–32. DOI: [10.1007/978-94-011-1793-7\\_4](https://doi.org/10.1007/978-94-011-1793-7_4).
- [6] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle: Verlag von Louis Nebert, 1879.
- [7] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völpl, and André Platzer. KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: *CADE*. Ed. by Amy Felty and Aart Middeldorp. Vol. 9195. LNCS.

- Berlin: Springer, 2015, 527–538. DOI: [10.1007/978-3-319-21401-6\\_36](https://doi.org/10.1007/978-3-319-21401-6_36).
- [8] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Math. Zeit.* **39**(2) (1935), 176–210. DOI: [10.1007/BF01201353](https://doi.org/10.1007/BF01201353).
- [9] Gerhard Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Math. Ann.* **112** (1936), 493–565. DOI: [10.1007/BF01565428](https://doi.org/10.1007/BF01565428).
- [10] Kurt Gödel. Über die Vollständigkeit des Logikkalküls. PhD thesis. Universität Wien, 1929.
- [11] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte Math. Phys.* **38**(1) (1931), 173–198. DOI: [10.1007/BF01700692](https://doi.org/10.1007/BF01700692).
- [12] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* **12**(3-4) (1958), 280–287. DOI: [10.1111/j.1746-8361.1958.tb01464.x](https://doi.org/10.1111/j.1746-8361.1958.tb01464.x).
- [13] David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik*. Berlin: Springer, 1928.
- [14] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM* **12**(10) (1969), 576–580. DOI: [10.1145/363235.363259](https://doi.org/10.1145/363235.363259).
- [15] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica* **16** (1963), 83–94.
- [16] Gottfried Wilhelm Leibniz. *Generales inquisitiones de analysi notionum et veritatum*. 1686.
- [17] Francesco Logozzo. Practical verification for the working programmer with codecontracts and abstract interpretation - (invited talk). In: *VMCAI*. Ed. by Ranjit Jhala and David A. Schmidt. Vol. 6538. LNCS. Berlin: Springer, 2011, 19–22. DOI: [10.1007/978-3-642-18275-4\\_3](https://doi.org/10.1007/978-3-642-18275-4_3).
- [18] Bertrand Meyer. Applying design by contract. *Computer* **25**(10) (Oct. 1992), 40–51. DOI: [10.1109/2.161279](https://doi.org/10.1109/2.161279).
- [19] Stefan Mitsch and André Platzer. ModelPlex: verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.* **49**(1-2) (2016). Special issue of selected papers from RV’14, 33–74. DOI: [10.1007/s10703-016-0241-z](https://doi.org/10.1007/s10703-016-0241-z).
- [20] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In: *TABLEAUX*. Ed. by Nicola Olivetti. Vol. 4548. LNCS. Berlin: Springer, 2007, 216–232. DOI: [10.1007/978-3-540-73099-6\\_17](https://doi.org/10.1007/978-3-540-73099-6_17).
- [21] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.* **41**(2) (2008), 143–189. DOI: [10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [22] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer, 2010. DOI: [10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [23] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In: *CADE*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. LNCS. Berlin: Springer, 2011, 446–460. DOI: [10.1007/978-3-642-22438-6\\_34](https://doi.org/10.1007/978-3-642-22438-6_34).

- [24] André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Log. Meth. Comput. Sci.* **8**(4:17) (2012). Special issue for selected papers from CSL'10, 1–44. DOI: [10.2168/LMCS-8\(4:17\)2012](https://doi.org/10.2168/LMCS-8(4:17)2012).
- [25] André Platzer. Logics of dynamical systems. In: *LICS*. Los Alamitos: IEEE, 2012, 13–24. DOI: [10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [26] André Platzer. The complete proof theory of hybrid systems. In: *LICS*. Los Alamitos: IEEE, 2012, 541–550. DOI: [10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [27] André Platzer. Teaching CPS foundations with contracts. In: *CPS-Ed*. 2013, 7–10.
- [28] André Platzer. Differential game logic. *ACM Trans. Comput. Log.* **17**(1) (2015), 1:1–1:51. DOI: [10.1145/2817824](https://doi.org/10.1145/2817824).
- [29] André Platzer. Logic & proofs for cyber-physical systems. In: *IJCAR*. Ed. by Nicola Olivetti and Ashish Tiwari. Vol. 9706. LNCS. Berlin: Springer, 2016, 15–21. DOI: [10.1007/978-3-319-40229-1\\_3](https://doi.org/10.1007/978-3-319-40229-1_3).
- [30] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.* **59**(2) (2017), 219–265. DOI: [10.1007/s10817-016-9385-1](https://doi.org/10.1007/s10817-016-9385-1).
- [31] André Platzer. Differential hybrid games. *ACM Trans. Comput. Log.* **18**(3) (2017), 19:1–19:44. DOI: [10.1145/3091123](https://doi.org/10.1145/3091123).
- [32] André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In: *HSCC*. Ed. by Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo. Vol. 4416. LNCS. Springer, 2007, 473–486. DOI: [10.1007/978-3-540-71493-4\\_37](https://doi.org/10.1007/978-3-540-71493-4_37).
- [33] Amir Pnueli. The temporal logic of programs. In: *FOCS*. IEEE, 1977, 46–57.
- [34] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In: *17th Annual Symposium on Foundations of Computer Science, 25-27 October 1976, Houston, Texas, USA*. Los Alamitos: IEEE, 1976, 109–121. DOI: [10.1109/SFCS.1976.27](https://doi.org/10.1109/SFCS.1976.27).
- [35] Arthur Prior. *Time and Modality*. Oxford: Clarendon Press, 1957.
- [36] Dana S. Scott. Logic and programming languages. *Commun. ACM* **20**(9) (1977), 634–641. DOI: [10.1145/359810.359826](https://doi.org/10.1145/359810.359826).
- [37] Thoralf Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Videnskapsselskapets skrifter, I. Mat.-naturv. klasse* **4** (1920), 1–36.
- [38] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge: Cambridge Univ. Press, 1910.
- [39] Dana N. Xu, Simon L. Peyton Jones, and Koen Claessen. Static contract checking for Haskell. In: *POPL*. Ed. by Zhong Shao and Benjamin C. Pierce. New York: ACM, 2009, 41–52. DOI: [10.1145/1480881.1480889](https://doi.org/10.1145/1480881.1480889).
- [40] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to Simulink/Stateflow verification. *Form. Methods Syst. Des.* **43**(2) (2013), 338–367. DOI: [10.1007/s10703-013-0195-3](https://doi.org/10.1007/s10703-013-0195-3).