Chapter 2
Differential Dynamic Logic dℒ

Contents

A. Platzer, Logical Analysis of Hybrid Systems,
DOI 10.1007/978-3-642-14509-4_2, © Springer-Verlag Berlin Heidelberg 2010

Synopsis Hybrid systems are models for complex physical systems and are defined as dynamical systems with interacting discrete transitions and continuous evolutions along differential equations. With the goal of developing a theoretical and practical foundation for deductive verification of hybrid systems, we introduce a dynamic logic for hybrid programs, which is a program notation for hybrid systems. As a verification technique that is suitable for automation, we introduce a free-variable proof calculus with a novel combination of real-valued free variables and Skolemisation for lifting quantifier elimination for real arithmetic to dynamic logic. The calculus is compositional, i.e., it reduces properties of hybrid programs to properties of their parts. Our main result proves that this calculus axiomatises the transition behaviour of hybrid systems completely relative to differential equations. In a study with cooperating traffic agents of the European Train Control System, we further show that our calculus is well suited for verifying realistic hybrid systems with parametric system dynamics.

2.1 Introduction

In this chapter, we introduce the differential dynamic logic $\mathsf{d}\mathscr{L}$, its syntax, semantics, and proof calculus. It forms the core of this book and is the basis for the extensions, algorithmic refinements, and applications in subsequent chapters of this book.

Contributions

Our main conceptual contribution in this chapter is the differential dynamic logic $\mathsf{d}\mathscr{L}$ for hybrid programs, which captures the logical quintessence of the dynamics of hybrid systems succinctly. Our main practical contribution is a concise free-variable calculus for $\mathsf{d}\mathscr{L}$ that axiomatises the transition behaviour of hybrid systems relative to differential equation solving. It is suitable for automated theorem proving and for verifying hybrid interacting discrete and continuous dynamics compositionally. Our main theoretical contribution is that we prove the $\mathsf{d}\mathscr{L}$ calculus to be sound and complete relative to the handling of differential equations. To the best of our knowledge, this is the first relative completeness proof for a logic of hybrid systems, and even the first formal notion of hybrid completeness. Our results fully align hybrid and continuous reasoning proof-theoretically and show that hybrid systems with interacting repetitive discrete and continuous evolutions can be verified whenever differential equations can. As an applied contribution, we further demonstrate that our logic and calculus can be used successfully for verifying collision avoidance in realistic train control applications.

2.1.1 Structure of This Chapter

After introducing syntax and semantics of the differential dynamic logic $\mathsf{d}\mathscr{L}$ in Sects. 2.2 and 2.3, we introduce a free-variable sequent calculus for $\mathsf{d}\mathscr{L}$ in Sect. 2.5 and prove soundness and relative completeness in Sects. 2.6 and 2.7, respectively. We present relatively semidecidable fragments of $\mathsf{d}\mathscr{L}$ in Sect. 2.8. In Sect. 2.9, we use our calculus to prove an inductive safety property of the train control system that we present in Sect. 2.4. We draw conclusions and discuss future work in Sect. 2.10.

2.2 Syntax of Differential Dynamic Logic

In this section, we introduce the *differential dynamic logic* $\mathsf{d}\mathscr{L}$ in which operational models of hybrid systems are internalised as first-class citizens, so that correctness statements about the transition behaviour of hybrid systems can be expressed as formulas. As a basis, $\mathsf{d}\mathscr{L}$ includes (nonlinear) real arithmetic for describing concepts like safe regions of the state space. Further, $\mathsf{d}\mathscr{L}$ supports real-valued quantifiers for quantifying over the possible values of system parameters or durations of continuous evolutions. For talking about the transition behaviour of hybrid systems, $\mathsf{d}\mathscr{L}$ provides modal operators such as $[\alpha]$ or $\langle\alpha\rangle$ that refer to the states reachable by following the transitions of hybrid system α.

The logic $\mathsf{d}\mathscr{L}$ is a first-order dynamic logic over the reals for hybrid programs, which is a compositional program notation for hybrid systems. Hybrid programs provide the following constructs.

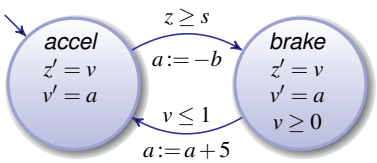
Discrete jump sets. Discrete transitions are represented as instantaneous assignments of values to state variables, which are, essentially, difference equations. They can express resets like $a := -b$ or adjustments of control variables like $a := a+5$, as occurring in the discrete transformations attached to edges in hybrid automata; see Fig. 2.1. Likewise, implicit discrete state changes such as the changing of evolution modes from one node of an automaton to the other can be expressed uniformly as, e.g., $q := \mathit{brake}$, where variable q remembers the current node. To handle simultaneous changes of multiple variables, discrete jumps can be combined to sets of jumps with simultaneous effect following corresponding techniques in the discrete case [37]. For instance, the discrete jump set $a := a+5, A := 2a^2$ expresses that a is increased by 5 and, simultaneously, variable A is set to $2a^2$, which is evaluated *before* a receives its new value $a+5$.

Differential equation systems. Continuous variation in system dynamics is represented using differential equation systems as evolution constraints. For example the (second-order) differential equation $z'' = -b$ describes deceleration with braking force b and $z' = v, v' = -b \,\&\, v \geq 0$ expresses that the evolution only applies as long as the speed is $v \geq 0$, which represents mode *brake* of Fig. 2.1. This is an evolution along the differential equation system $z' = v, v' = -b$ that is restricted (written &) to remain within the evolution domain region $v \geq 0$, i.e., to

stop braking before $v < 0$. Such an evolution can stop at any time within $v \geq 0$, it could even continue with transient grazing along the border $v = 0$, but it is never allowed to enter $v < 0$. The second-order differential equation $z'' = -b$ itself is equivalent to the first-order differential equation system $z' = v, v' = -b$, in which the velocity v is explicit. In this chapter, we separate the respective differential equations in a differential equation system by a comma (,) and separate the evolution domain region (if any) from the differential equations by an ampersand (&). We choose this notation for this chapter to make it easier to identify the evolution domain region. In Chap. 3, we will see that both (,) and (&) can be understood more uniformly as conjunctions.

Control structure. Discrete and continuous transitions—represented as difference or differential equations, respectively—can be combined to form a hybrid program with interacting hybrid dynamics using regular expression operators $(\cup, ^*, ;)$ of regular programs [149] as control structure. For example, the hybrid program $q := accel \cup z'' = -b$ describes a train controller that can choose to either switch to acceleration mode ($q := accel$) or brake by the differential equation $z'' = -b$, by a nondeterministic choice ($\cup$). The nondeterministic choice $q := accel \cup z'' = -b$ expresses that either $q := accel$ or $z'' = -b$ happens. The sequential composition $q := accel; z'' = -b$, instead, expresses that first $q := accel$, and then $z'' = -b$ happens. In conjunction with other regular combinations, control constraints can be expressed using tests like $?z \geq s$ as guards for the system state. This test will succeed if, indeed, the current state of the system satisfies $z \geq s$; otherwise the test will fail and execution cannot proceed. In that respect, a test is like an assert statement in conventional programs and cuts the system run if the test is not successful.

Example 2.1 (Embedding hybrid automata). With these operations, hybrid systems can be represented naturally as hybrid programs. For example, the right of Fig. 2.1 depicts a hybrid program rendition of the hybrid automaton on the left, which repeats the automaton from Fig. 1.4 on p. 5. Line 1 represents that, in the beginning,



$q := accel;$ /* initial mode is node accel */
$\big($ $(?q = accel;\ z' = v, v' = a)$
$\cup\ (?q = accel \wedge z \geq s;\ a := -b;\ q := brake;\ ?v \geq 0)$
$\cup\ (?q = brake;\ z' = v, v' = a \,\&\, v \geq 0)$
$\cup\ (?q = brake \wedge v \leq 1;\ a := a + 5;\ q := accel)\big)^*$

Fig. 2.1 Hybrid program rendition of hybrid automaton for (overly) simplified train control

the current node q of the system is the initial node *accel*. We represent each discrete and continuous transition of the automaton as a sequence of statements with a nondeterministic choice ($\cup$) between these transitions. Line 4 represents a continuous transition of the automaton. It tests if the current node q is *brake*, and then (i.e., if the test was successful) follows the differential equation system $z' = v, v' = a$ restricted

to the evolution domain $v \geq 0$. Line 3 characterises a discrete transition of the automaton. It tests the guard $z \geq s$ when in node *accel*, and, if successful, resets $a := -b$ and then switches q to node *brake*. By the semantics of hybrid automata [8, 156], an automaton in node *accel* is only allowed to make a transition to node *brake* if the evolution domain restriction of *brake* is true when entering the node, which is expressed by the additional test $?v \geq 0$ at the end of line 3. Observe that this test of the evolution domain region generally needs to be checked as the last operation after the guard and reset, because a reset like $v := v - 1$ could affect the outcome of the evolution domain region test. In order to obtain a fully compositional model, hybrid programs make all these implicit side conditions explicit. Line 2 represents the continuous transition when staying in node *accel* and following the differential equation system $z' = v, v' = a$. Line 5 represents the discrete transition from node *brake* of the automaton to node *accel*.

Lines 2–5 cannot be executed unless their tests succeed. In particular, at any state, the nondeterministic choice ($\cup$) among lines 2–5 reduces de facto to a nondeterministic choice between either lines 2–3 or between lines 4–5. At any state, q can have value either *accel* or *brake* (assuming these are different constants), not both. Consequently, when $q = brake$, a nondeterministic choice of lines 2–3 would immediately fail the tests in the beginning and not execute any further. The only remaining choices that have a chance to succeed are lines 4–5 then. In fact, only the single successful choice of line 4 would remain if the second conjunct $v \leq 1$ of the test in line 5 does not hold for the current state. Note that, still, all four choices in lines 2–5 are available, but at least two of these nondeterministic choices will always be unsuccessful. Finally, the repetition operator (*) at the end of Fig. 2.1 expresses that the transitions of a hybrid automaton, as represented by lines 2–5, can repeat indefinitely, possibly taking different nondeterministic choices between lines 2–5 at every repetition. □

2.2.1 Terms

The construction of the logic $\mathsf{d}\mathscr{L}$ starts with a set V of logical variables and a *signature* Σ, which is the set of names (called *symbols*) of all entities nameable in a certain context. The signature Σ and set V form the vocabulary or alphabet of signs from which well-formed formulas can be built. For $\mathsf{d}\mathscr{L}$ we assume all variables in V are interpreted over the reals and that Σ is a (finite) set of real-valued function and predicate symbols, with the usual function and predicate symbols for real arithmetic, such as $0, 1, +, -, \cdot, /, =, \leq, <, \geq, >$, where $+$ is addition, $\cdot$ is multiplication, $/$ is division and so on. For each function and predicate symbol, we are given the number of arguments that it expects, which is called *arity*, and is a natural number. The arity can be zero, in which case the function or predicate symbol does not have any arguments. The function symbols for the numbers 0,1 have arity zero, because they do not need arguments. The binary arithmetic operators $+, -, \cdot, /$ have arity

2, because they expect two arguments. The binary predicate symbols $=, \leq, <, \geq, >$ also have arity 2, because they need two arguments to compare.

The difference between function and predicate symbols is that function symbols stand for functions that take the values of arguments and give back a function value. Predicate symbols, on the other hand, are interpreted either as true for a vector of arguments or as false. That is, they take the values of arguments and give either the truth-value "*true*" or the truth-value "*false*". No other result is permitted for predicate symbols. For instance, the predicate symbol $\geq$ will be understood such that $\geq(x,y)$ is true if and only if the value of x is greater than or equal to the value of y. For real arithmetic, we use standard notation and standard semantics. In particular, we write $x \geq y$ instead of $\geq(x,y)$. We fix the semantics of $\cdot$ to be multiplication, i.e., the value of $\cdot(x,y)$ is always meant to be the product of the value of x and the value of y. Again, we use the standard notation $x \cdot y$, or just xy if no confusion arises, instead of $\cdot(x,y)$. The denotation of a function symbol could also be, e.g., the function that takes an argument and gives back its cube. A predicate, in contrast, cannot give back any value other than "*true*" or "*false*", but could hold, say, for all real numbers larger than 5. Or it could be the relation that holds for all pairs where the second element of the pair is larger than the square of the first element of the pair. Function symbols are often written as f, g, h, a, b, c and predicate symbols are often written as p, q, r.

State variables of hybrid systems, such as positions, velocities, and accelerations, are represented as real-valued function symbols of Σ of arity 0. Unlike fixed symbols like the number 1, state variables are *flexible*, i.e., their interpretation can change from state to state during the execution of a hybrid program. Flexibility of symbols will be used to represent the progression of system values along states over time during a hybrid evolution. Symbols like 1, on the other hand, are *rigid*, i.e., they have the same value at all states. The symbols of real arithmetic like 1 and $+, \cdot$ are rigid, because we do not want them to change their meaning at any time. State variables like velocity v, in contrast, are flexible, because they can change their value depending on the state. While the velocity v may have been 0 in the beginning, the train could increase its velocity to 10 and then decrease it again later when approaching another train.

Note that there is no need to distinguish between discrete and continuous variables in dℒ. The distinction between logical variables in V, which can be quantified universally or existentially, and state variables in Σ, which can change their value by discrete jumps and differential equations of hybrid programs in modalities, is not strictly required either. For instance, universal and existential quantification of state variables is definable using auxiliary logical variables. The distinction makes the semantics and soundness proof less subtle, though. Our calculus assumes that V contains sufficiently many variables and Σ contains additional Skolem function symbols, which are reserved for use by the calculus.

Terms

Well-formed arguments to function symbols and predicate symbols are called terms. Logical variables are well-formed terms, and functions applied to the appropriate number of terms as arguments are well-formed terms. The set $\mathrm{Trm}(\Sigma, V)$ of *terms* is defined as in classical first-order logic, yielding polynomial (or rational) expressions over V and over additional Skolem terms $s(t_1, \ldots, t_n)$ with terms t_i. Our calculus actually only uses Skolem terms $s(X_1, \ldots, X_n)$ with logical variables $X_i \in V$ as arguments.

Definition 2.1 (Terms). $\mathrm{Trm}(\Sigma, V)$ is the set of all *terms*, which is the smallest set such that:

- If $x \in V$, then $x \in \mathrm{Trm}(\Sigma, V)$.
- If $f \in \Sigma$ is a function symbol of arity $n \geq 0$ and, for $1 \leq i \leq n$, $\theta_i \in \mathrm{Trm}(\Sigma, V)$, then $f(\theta_1, \ldots, \theta_n) \in \mathrm{Trm}(\Sigma, V)$. The case $n = 0$ is permitted (e.g., for state variables).

More succinctly, we also say that the terms of $\mathsf{d}\mathscr{L}$ are defined by the following grammar (where $\theta_1, \ldots, \theta_n$ are terms, f a function symbol of arity n, and $x \in V$ is a logical variable):

$$\theta \ ::= \ x \mid f(\theta_1, \ldots, \theta_n).$$

Example 2.2. (Well-formed) terms of $\mathsf{d}\mathscr{L}$ include:

- Logical variables $X \in V$
- State variables $x \in \Sigma$ that may change their value during system evolution
- Expressions of nonlinear polynomial real arithmetic like $x + 5y \cdot (x - 3y + za)$, which we consider as a short notation for $x + 5 \cdot y \cdot (x - 3 \cdot y + z \cdot a)$. Here we assume that $x, y, z, a \in \Sigma$ are state variables. In principle, we also have to mention that the number symbols $5, 3 \in \Sigma$ are (rigid) function symbols without arguments. Yet these number symbols are what we assume as given throughout this book. Note that we could just as well have assumed that $x, y \in \Sigma$ are state variables, $a \in \Sigma$ is a rigid function symbol of arity 0, and $z \in V$ is a logical variable. Then $x + 5y \cdot (x - 3y + za)$ is still a term for this different signature and variables set. For terms, all ways of declaring symbols as state variables, rigid function symbols of arity 0, or logical variables are essentially equivalent. There are many ways to say the same thing. The differences only play a role later for quantification and state change.
- Expressions with Skolem function terms like $x + 5s(X_1, X_2) \cdot (x - 3y + z \cdot t(X_2))$. Here we assume that $x, y \in \Sigma$ are state variables, that $s, t \in \Sigma$ are rigid function symbols of arity 2 and 1, respectively, and that $X_1, X_2 \in V$ are logical variables.
- Real arithmetic expressions with integer powers like $8x^2 + 2x^3(y - a^2bc)$ that can easily be rewritten as $8 \cdot x \cdot x + 2 \cdot x \cdot x \cdot x \cdot (y - a \cdot a \cdot b \cdot c)$. Again, we assume that x, y, a, b, c are symbols in Σ or V.

The following, however, are *no terms* with respect to Σ and V:

- $1+x^y$, because the exponential function x^y cannot be rewritten as a finite product, quite unlike $x^3 = x \cdot x \cdot x$ or $x^4 = x \cdot x \cdot x \cdot x$. In fact, the logical properties of the exponential function are a very exciting and a challenging object of study in recent model theory [107, 206, 44, 108, 45, 2].
- $y^2 - \pi$, unless the transcendental number $\pi = 3.1415926\ldots$ is explicitly added to Σ, because unlike rational numbers, the transcendental number cannot be characterised exactly with a finite combination of sums, products, and 0, 1. Arbitrarily precise approximations of π, instead, can be defined; see Example 2.3. □

First-Order Formulas

Meaningful propositions that are either true or false in a context are called (well-formed) formulas. The well-formed formulas of a logic form a formal language over the alphabet $\Sigma \cup V$ of symbols. The formulas consist of all words that can be built by recursively combining symbols of the signature with logical operator symbols appropriately. We first define only the fragment of first-order logic, then the syntax of hybrid programs, and define the actual formulas of differential dynamic logic afterwards.

The set of formulas of *first-order logic* is defined as usual (cf. App. A), giving first-order real arithmetic [288] augmented with Skolem terms. We will show the precise relationship to standard first-order real arithmetic without Skolem terms in Lemma 2.5 of Sect. 2.5.3.2.

Definition 2.2 (First-order formulas). The set $\mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$ of *formulas of first-order logic* is the smallest set with:

- If $p \in \Sigma$ is a predicate symbol of arity $n \geq 0$ and $\theta_i \in \mathrm{Trm}(\Sigma, V)$ for $1 \leq i \leq n$, then $p(\theta_1, \ldots, \theta_n) \in \mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$.
- If $\phi, \psi \in \mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$, then $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi) \in \mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$.
- If $\phi \in \mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$ and $x \in V$, then $(\forall x\, \phi), (\exists x\, \phi) \in \mathrm{Fml}_{\mathrm{FOL}}(\Sigma, V)$.

More succinctly, we also say that first-order formulas are defined by the following grammar (where ϕ, ψ are first-order formulas, θ_i are terms, p is a predicate symbol of arity n, and $x \in V$ is a logical variable):

$$\phi, \psi \ ::= \ p(\theta_1, \ldots, \theta_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi.$$

Example 2.3. (Well-formed) first-order formulas in our context include:

- $v \cdot v \leq 2b \cdot (m - z)$. Again we assume that v, b, m, z are symbols in the vocabulary Σ or V. For instance, we could assume that $z, v \in \Sigma$ are (flexible) state variables and $b, m \in \Sigma$ are rigid function symbols of arity 0. The rationale for this classification would be that z and v are meant to represent the position and velocity of a train, which of course can change over time (flexible). The symbols b and m are meant to represent the braking force and movement authority of a

train, which we assume not to change in this formula (rigid). We could just as well assume that $z, v, b, m \in \Sigma$ are (flexible) state variables.

- $v > 0 \rightarrow v \cdot v \leq 2b \cdot (m - z) \vee b = 0$
- $\forall x \forall y\,(x > y \leftrightarrow x - y > 0)$. Here we assume $x, y \in V$ are logical variables; otherwise the syntax would not allow us to quantify over x, y.
- $x > 0 \wedge \forall y \exists z\,(x > z^2 + y \cdot z - 5)$. Here we assume $y, z \in V$ are logical variables and we could either assume $x \in \Sigma$ to be a state variable, or a rigid function symbol of arity zero, or a logical variable $x \in V$. All those choices are reasonable for this formula, and, in fact, it does not make a real difference here, because they will essentially have the same meaning. These distinctions become somewhat more important for $\mathsf{d}\mathscr{L}$ formulas later.
- Formulas with divisions like $b < x/y$, which can easily be defined in terms of multiplication $(b \cdot y < x \wedge y > 0) \vee (b \cdot y > x \wedge y < 0)$.
- Formulas with rational constants like $a > \frac{2}{3}x^2 + 3.1415x \cdot y^4$, which can easily be defined in terms of successive addition and inverses, say,

$$a > (1+1)/(1+1+1) \cdot x^2 + 31415/10000 \cdot x \cdot y^4.$$

- Arithmetic expressions with roots like $x^4 - y\sqrt{2z} > 0$, which can easily be defined in terms of their characteristic polynomials as $\exists r\,(r^2 = 2z \wedge r \geq 0 \wedge x^4 - y \cdot r > 0)$.

□

2.2.2 Hybrid Programs

As uniform compositional models for hybrid systems, hybrid programs can combine discrete and continuous transitions to structured control programs using the regular-expression-style operators of Kleene algebras [182].

Definition 2.3 (Hybrid programs). The set $\mathrm{HP}(\Sigma, V)$ of *hybrid programs*, with typical elements α, β, is defined inductively as the smallest set such that

1. If $x_i \in \Sigma$ is a state variable and $\theta_i \in \mathrm{Trm}(\Sigma, V)$ for $1 \leq i \leq n$, then the *discrete jump set* $(x_1 := \theta_1, \ldots, x_n := \theta_n) \in \mathrm{HP}(\Sigma, V)$ is a hybrid program. We assume that the $x_1, \ldots, x_n$ are pairwise different state variables.
2. If $x_i \in \Sigma$ is a state variable and $\theta_i \in \mathrm{Trm}(\Sigma, V)$ for $1 \leq i \leq n$, then $x_i' = \theta_i$ is a *differential equation* in which x_i' represents the time derivative of variable x_i. If χ is a first-order formula, then $(x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi) \in \mathrm{HP}(\Sigma, V)$. We assume that the $x_1, \ldots, x_n$ are pairwise different state variables.
3. If χ is a first-order formula, then $(?\chi) \in \mathrm{HP}(\Sigma, V)$.
4. If $\alpha, \beta \in \mathrm{HP}(\Sigma, V)$, then $(\alpha \cup \beta) \in \mathrm{HP}(\Sigma, V)$.
5. If $\alpha, \beta \in \mathrm{HP}(\Sigma, V)$, then $(\alpha; \beta) \in \mathrm{HP}(\Sigma, V)$.
6. If $\alpha \in \mathrm{HP}(\Sigma, V)$, then $(\alpha^*) \in \mathrm{HP}(\Sigma, V)$.

Table 2.1 summarises the statements and (informal) effects of hybrid programs. More succinctly, hybrid programs are defined by the following grammar (α, β are

Table 2.1 Statements and (informal) effects of hybrid programs (HPs)

HP Notation	Operation	Effect
$x_1 := \theta_1, \ldots, x_n := \theta_n$	discrete jump set	simultaneously assigns terms θ_i to variables x_i
$x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi$	continuous evolution	differential equations for x_i with terms θ_i within first-order constraint χ (evolution domain)
$?\chi$	state test / check	test first-order formula χ at current state
$\alpha;\beta$	seq. composition	HP β starts after HP α finishes
$\alpha \cup \beta$	nondet. choice	choice between alternatives HP α or HP β
α^*	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}$

hybrid programs, θ_i are terms, $x_i \in \Sigma$ are state variables, and χ is a formula of first-order logic):

$$\alpha, \beta \ ::= \ x_1 := \theta_1, \ldots, x_n := \theta_n \mid x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi \mid ?\chi \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*.$$

The effect of the discrete jump set $x_1 := \theta_1, \ldots, x_n := \theta_n$ is to simultaneously change the interpretations of the x_i to the respective θ_i by performing a discrete jump in the state space. In particular, the new values θ_i are evaluated before changing the value of any variable x_j. The effect of $x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi$ is an ongoing continuous evolution respecting the differential equation system $x_1' = \theta_1, \ldots, x_n' = \theta_n$ that is restricted to remain within the evolution domain region χ. The evolution is allowed to stop at any point in χ. It is, however, required to stop before it leaves χ. For unconstrained evolutions, we write $x' = \theta$ in place of $x' = \theta \,\&\, \mathit{true}$. For structural reasons, we expect both difference equations (discrete jump sets) and differential equations to be given in explicit form, i.e., with the affected variable on the left (we allow more general implicit forms in Chap. 3). The $\mathsf{d}\mathscr{L}$ semantics allows arbitrary differential equations. To retain feasible arithmetic, some of our calculus rules in this chapter assume that, as in [8, 125, 217, 156], the differential equations have first-order definable flows or approximations. We assume that standard techniques are used to determine corresponding solutions or approximations, e.g., [15, 189, 238, 227, 297]. We consider verification techniques for more advanced differential equations in Chap. 3.

The test action or state check $?\chi$ is used to define conditions. Its semantics is that of a no-op if the formula χ is true in the current state; otherwise, like *abort*, it allows no transitions. That is, if the test succeeds because formula χ holds in the current state, then the state does not change, and the system execution continues normally. If the test fails because formula χ does not hold in the current state, then the system execution cannot even continue. Thus, the effect of a test action is similar to an *assert* statement in Java. Note that, according to Definition 2.3, we have only allowed first-order formulas as tests. Instead, we could actually allow *rich tests*, i.e., arbitrary $\mathsf{d}\mathscr{L}$ formulas χ with nested modalities as tests $?\chi$ inside hybrid programs (and even in evolution domains χ of differential equations). The calculus and our meta-results, including soundness and relative completeness, directly carry over to this rich test version of $\mathsf{d}\mathscr{L}$. To simplify the presentation, however, we refrain from allowing

arbitrary $\mathsf{d}\mathscr{L}$ formulas as tests, because that requires simultaneous inductive handling of hybrid programs and $\mathsf{d}\mathscr{L}$ formulas in syntax, semantics, and completeness proofs, because $\mathsf{d}\mathscr{L}$ formulas would then be allowed to occur in hybrid programs, and vice versa.

The nondeterministic choice $\alpha \cup \beta$, sequential composition $\alpha;\beta$, and nondeterministic repetition α^* of programs are as in regular expressions but generalised to a semantics in hybrid systems. Choices $\alpha \cup \beta$ are used to express behavioural alternatives between the transitions of α and β. That is, the hybrid program $\alpha \cup \beta$ can choose nondeterministically to follow the transitions of the hybrid program α, or, instead, to follow the transitions of the hybrid program β. The sequential composition $\alpha;\beta$ says that the hybrid program β starts executing after α has finished (β never starts if α does not terminate). In $\alpha;\beta$, the transitions of α take effect first, until α terminates (if it does), and then β continues. Observe that, like repetitions, continuous evolutions within α can take more or less time, which causes uncountable nondeterminism. This nondeterminism is inherent in hybrid systems, because they can operate in so many different ways, and as such reflected in hybrid programs. Repetition α^* is used to express that the hybrid process α repeats any number of times, including zero times. When following α^*, the transitions of hybrid program α can be repeated over and over again, any nondeterministic number of times (≥ 0). Hybrid programs form a regular-expression-style Kleene algebra with tests [182].

Example 2.4 (Simplistic train). The differential equation $z' = v, v' = a$ expresses continuous movement of position z with velocity v and acceleration a. A very simple (in fact much too simplistic) train controller could be the following hybrid program:

$$((a := -b) \cup (?v < 8; a := A)); \; z' = v, v' = a.$$

By a nondeterministic choice ($\cup$), the system either chooses to set the acceleration a to the braking force $-b$ by executing $a := -b$, or the system tries to pass the test $?v < 8$ instead. If the system tries the second choice and the latter test succeeds, i.e., the current velocity is indeed less than 8, then the system sets the acceleration a to A. Otherwise, if it tries the second choice but the test fails, then nothing happens as this execution is blocked and cannot continue. Afterwards (after executing the first part of the sequential composition, which is the nondeterministic choice), the system follows the second part of the sequential composition, which is the differential equation $z' = v, v' = a$ with the previously chosen acceleration. The system then follows this differential equation for a certain (unspecified) period of time.

This controller leaves open too many aspects to be useful, but already illustrates a very simple hybrid program. One of the problems is that the controller can only take a control action for choosing the acceleration a once, at the beginning of the system evolution, and then follows the differential equation for an arbitrarily long time. But the above controller can never react to situation changes and change its mind with a different choice of a when necessary. To improve this issue, the following hybrid program allows repetitive choices by the repetition operator *:

$$\big(((a:=-b)\cup(?v<8;a:=A));\, z'=v,v'=a\big)^*. \tag{2.1}$$

Unlike the previous hybrid program, the hybrid program in (2.1) contains a repetition, which can change the acceleration repeatedly over and over again after following the continuous evolution for a certain period of time. While already an improvement over the last controller, this hybrid program has shortcomings. For one thing, the differential equation does not say when it stops. It has no evolution domain restriction and would thus be allowed to evolve as long or as short as it pleases. This may be unsafe if the differential equation would continue indefinitely without giving the controller for the acceleration a chance to react to situation changes. Furthermore, a velocity of 8 may not be a safe choice for the switching condition between acceleration and braking. We will see in Sect. 2.4 how a reasonable train controller can be designed as a hybrid program from first principles and elaborate on train control further in Chap. 7 in full detail. □

Definable Operations

The control flow operations of choice, sequential composition, and repetition in hybrid programs can be combined with $?\chi$ to form all other control structures [149]. All classical discrete control structures can be defined in terms of the basic hybrid program operators (it is easy to see that hybrid programs are Turing-complete). See Table 2.2 for a selection of control structures and statements that are definable as a hybrid program. For instance, $(?\chi;\alpha)^*;?\neg\chi$ corresponds to a while loop that repeats α while χ holds and only stops when χ ceases to hold after α. Because the *-operator can repeat arbitrarily often, the subprogram $(?\chi;\alpha)^*$ can repeat α any number of times, but a repetition can only be successful if the test $?\chi$ succeeds. Hence the repetitions have to stop, at the latest, when the test $?\chi$ fails. Now the

Table 2.2 Statements and control structures definable with hybrid programs

HP Notation	Operation	Effect
$x:=*$	nondet. assignment	assigns any real value to x equivalently definable, see Chap. 3
if χ then α else β	if-then-else	executes HP α if χ holds, otherwise HP β equivalent to $(?\chi;\alpha)\cup(?\neg\chi;\beta)$
if χ then α	if-then	executes HP α if χ holds, otherwise no effect equivalent to $(?\chi;\alpha)\cup(?\neg\chi)$
while χ do α	while loop	repeats α if χ holds, only stops if $\neg\chi$ holds at end equivalent to $(?\chi;\alpha)^*;?\neg\chi$
repeat α until χ	repeat until	repeats HP α until χ holds at end (at least once) equivalent to $\alpha;(?\neg\chi;\alpha)^*;?\chi$
skip	do nothing	has no effect and does not change the state space equivalent to $?\mathit{true}$
abort	aborts execution	blocks current execution and allows no transition equivalent to $?\mathit{false}$

subprogram $(?\chi;\alpha)^*$ can repeat any number of times and is allowed to stop even if the test $?\chi$ is successful and the loop could be repeated again. But the subsequent sequential composition with the test $?\neg\chi$ makes sure that $(?\chi;\alpha)^*$ can only stop repeating when χ actually ceases to hold. Overall, the hybrid program $(?\chi;\alpha)^*;?\neg\chi$ executes α if χ holds and repeats α again exactly as often as χ still holds after executing α.

If-then-else can be defined with nondeterministic choices and tests. The corresponding hybrid program $(?\chi;\alpha)\cup(?\neg\chi;\beta)$ in Table 2.2 makes a nondeterministic choice between $?\chi;\alpha$ and $?\neg\chi;\beta$. While this choice is nondeterministic, at any state only one of the subsequent tests in the two cases can succeed, because they are complementary. Consequently, hybrid program α will be executed if and only if the test $?\chi$ succeeds because χ is true at the current state. Likewise, hybrid program β will be executed if and only if the dual test $?\neg\chi$ succeeds because $\neg\chi$ is true, i.e., χ is false at the current state. The nondeterministic assignment $x:=*$ that assigns an arbitrary real number to state variable x is definable also. While it is possible define nondeterministic assignments in hybrid programs already, we will come back to this in Chap. 3, where the definition is easier to see.

Example 2.5 (Parametric bouncing ball). As a classical example from the hybrid systems literature [110], consider the bouncing ball. We will describe the bouncing ball as a hybrid program, using the definable hybrid program operations from Table 2.2. Figure 2.2 depicts a hybrid automaton, an illustration of the bouncing ball dynamics, and a representation of the system as a hybrid program.

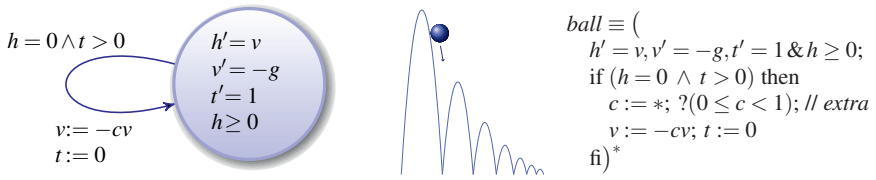


Fig. 2.2 Parametric bouncing ball

The bouncing ball is let loose and falls from height h, but bounces back from the ground (which corresponds to height $h=0$) after an elastic deformation. The current speed of the ball is denoted by v, and t is a clock measuring the falling time. The bouncing ball follows the continuous dynamics of physical movement by gravity. The ball is affected by gravity of force g, so its height follows the differential equation $h''=-g$. This second-order differential equation is equivalent to the first-order differential equation system $h'=v,v'=-g$, with an explicit velocity v. Simultaneously, clock t evolves according to the differential equation $t'=1$. Finally, the ball always stays above the ground and cannot fall through, thus its evolution domain is restricted to $h\geq 0$. Altogether, this gives the continuous evolution $h'=v,v'=-g,t'=1\,\&\,h\geq 0$ in the beginning of the hybrid program in Fig. 2.2.

At the ground (which is at height $h=0$), the ball bounces back after losing energy in an elastic deformation according to a damping factor $0\leq c<1$. That is, if the ball

is on the ground ($h = 0$) and it actually fell (so time has passed, $t > 0$), then the ball changes its direction and bounces back into the air by reflecting its current velocity v by a discrete jump $v := -cv$ and resetting the falling time by $t := 0$.

Now for illustration purposes we have added an extra twist to the hybrid program in Fig. 2.2 that is not in the hybrid automaton. The automaton still enforces infinite bouncing so that the ball can never stop (unless $c = 0$, where it stops immediately). In reality, the ball bounces a couple of times and can then come to a standstill when its remaining kinetic energy is insufficient. To model this phenomenon without the need to have a precise physical model for all physical forces and frictions, we allow for the damping factor c to change at each bounce. Line 4 of the hybrid program in Fig. 2.2 represents a corresponding uncountably infinite nondeterministic choice for c as a nondeterministic assignment. The subsequent test $?(0 \leq c < 1)$ restricts the arbitrary choices for c to choices in the half-open interval $[0, 1)$ and discards all other choices.

For comparison, Fig. 2.3 shows an equivalent hybrid program for the same bouncing, now with all abbreviations for extended statements resolved according to Table 2.2. Note that it is fairly easy to see that height h and clock t always stay nonnegative if they start nonnegative. For that reason, the last test $?(h \neq 0 \vee t \leq 0)$ in Fig. 2.3 could even be replaced equivalently by $?(h > 0 \vee t = 0)$. □

Fig. 2.3 Parametric bouncing ball (with abbreviations resolved)

$$
\begin{array}{l}
ball \equiv \big(\\
\quad h' = v, v' = -g, t' = 1 \,\&\, h \geq 0; \\
\quad \big(?(h = 0 \wedge t > 0); \\
\qquad\qquad (c' = 1 \cup c' = -1); \\
\qquad\qquad ?(0 \leq c < 1); \\
\qquad\qquad v := -cv; \, t := 0 \\
\quad \big) \cup ?(h \neq 0 \ \vee \ t \leq 0) \\
\big)^*
\end{array}
$$

Classification of Hybrid Programs

Hybrid programs are designed as a minimal extension of conventional discrete programs. They characterise hybrid systems succinctly by adding continuous evolution along differential equations as the only additional primitive operation to a regular basis of conventional discrete programs. To yield hybrid systems, their operations are interpreted over the domain of real numbers. This gives rise to an elegant syntactic hierarchy of discrete, continuous, and hybrid systems. Hybrid automata [156] can be represented as hybrid programs using a straightforward generalisation of standard program encodings of automata; see App. C for formal details. The fragment of hybrid programs without differential equations corresponds to conventional discrete programs generalised over the reals or to discrete-time dynamical systems [56]. The fragment without discrete jumps corresponds to switched continuous systems [56, 58], whereas the fragment of differential equations gives purely con-

Table 2.3 Operators and (informal) meaning in differential dynamic logic ($\mathsf{d}\mathscr{L}$)

$\mathsf{d}\mathscr{L}$ Notation	Operator	Meaning
$p(\theta_1,\ldots,\theta_n)$	atomic formula	true iff predicate p holds for $(\theta_1,\ldots,\theta_n)$
$\neg\phi$	negation / not	true if ϕ is false
$\phi\wedge\psi$	conjunction / and	true if both ϕ and ψ are true
$\phi\vee\psi$	disjunction / or	true if ϕ is true or if ψ is true
$\phi\rightarrow\psi$	implication / implies	true if ϕ is false or ψ is true
$\phi\leftrightarrow\psi$	bi-implication / equivalent	true if ϕ and ψ are both true or both false
$\forall x\,\phi$	universal quantifier / for all	true if ϕ is true for all values of variable x
$\exists x\,\phi$	existential quantifier / exists	true if ϕ is true for some values of variable x
$[\alpha]\phi$	$[\cdot]$ modality / box	true if ϕ is true after all runs of HP α
$\langle\alpha\rangle\phi$	$\langle\cdot\rangle$ modality / diamond	true if ϕ is true after at least one run of HP α

tinuous dynamical systems [279]. Only the composition of mixed discrete jumps and continuous evolutions gives rise to truly hybrid behaviour.

2.2.3 Formulas of Differential Dynamic Logic

The formulas of the differential dynamic logic $\mathsf{d}\mathscr{L}$ are defined as in first-order dynamic logic [253, 148, 149] but with real arithmetic as a semantic domain and with hybrid programs as system models. That is, they are built using propositional connectives $\neg,\wedge,\vee,\rightarrow,\leftrightarrow$ and quantifiers $\forall,\exists$ over the reals (first-order part). In addition, if ϕ is a $\mathsf{d}\mathscr{L}$ formula and α a hybrid program, then $[\alpha]\phi,\langle\alpha\rangle\phi$ are formulas (dynamic part).

Definition 2.4 ($\mathsf{d}\mathscr{L}$ formulas). The set $\mathrm{Fml}(\Sigma,V)$ of *formulas* of $\mathsf{d}\mathscr{L}$, with typical elements ϕ,ψ, is the smallest set such that

1. If p is a predicate symbol of arity $n\geq 0$ and $\theta_i\in\mathrm{Trm}(\Sigma,V)$ for $1\leq i\leq n$, then $p(\theta_1,\ldots,\theta_n)\in\mathrm{Fml}(\Sigma,V)$.
2. If $\phi,\psi\in\mathrm{Fml}(\Sigma,V)$, then $\neg\phi,(\phi\wedge\psi),(\phi\vee\psi),(\phi\rightarrow\psi)\in\mathrm{Fml}(\Sigma,V)$.
3. If $\phi\in\mathrm{Fml}(\Sigma,V)$ and $x\in V$, then $\forall x\,\phi,\exists x\,\phi\in\mathrm{Fml}(\Sigma,V)$.
4. If $\phi\in\mathrm{Fml}(\Sigma,V)$ and $\alpha\in\mathrm{HP}(\Sigma,V)$, then $[\alpha]\phi,\langle\alpha\rangle\phi\in\mathrm{Fml}(\Sigma,V)$.

For reference, the logical operators of differential dynamic logic are summarised in Table 2.3. More succinctly, we also say that the formulas of $\mathsf{d}\mathscr{L}$ are defined by the following grammar (where ϕ,ψ are $\mathsf{d}\mathscr{L}$ formulas, θ_i are terms, p a predicate symbol of arity n, $x\in V$ is a logical variable, and α is a hybrid program):

$$\phi,\psi ::= p(\theta_1,\ldots,\theta_n)\mid\neg\phi\mid\phi\wedge\psi\mid\phi\vee\psi\mid\phi\rightarrow\psi\mid\forall x\,\phi\mid\exists x\,\phi\mid[\alpha]\phi\mid\langle\alpha\rangle\phi.$$

We consider the bi-implication or equivalence $\phi\leftrightarrow\psi$ as an abbreviation for $(\phi\rightarrow\psi)\wedge(\psi\rightarrow\phi)$ to simplify the calculus. We often leave out superfluous brackets and use binding priorities instead in order to improve readability. Quantifiers

and modalities bind strongly, i.e., their scope only extends to the formula immediately after. Unary operators (negation $\neg$), quantifiers ($\forall, \exists$), and modalities ($[\alpha], \langle\alpha\rangle$) bind stronger than binary operators. Further, conjunction $\wedge$ and disjunction $\vee$ bind stronger than implication $\rightarrow$ and bi-implication $\leftrightarrow$. Thus

$$\phi_0 \wedge \langle\alpha\rangle\phi_1 \wedge \forall x\, \phi_2 \wedge \phi_3 \rightarrow \neg\phi_4 \vee [\alpha]\phi_5 \vee \phi_6$$

is taken to mean

$$\big(\phi_0 \wedge (\langle\alpha\rangle\phi_1) \wedge (\forall x\, \phi_2) \wedge \phi_3\big) \rightarrow \big((\neg\phi_4) \vee ([\alpha]\phi_5) \vee \phi_6\big)$$

and does not mean

$$\phi_0 \wedge \Big(\langle\alpha\rangle\big(\phi_1 \wedge \forall x\, (\phi_2 \wedge \phi_3)\big) \rightarrow \neg\big(\phi_4 \vee [\alpha](\phi_5 \vee \phi_6)\big)\Big).$$

Example 2.6 (Train control). When *train* denotes the hybrid program in Fig. 2.1 or the hybrid program in Example 2.4, or, in fact, any other hybrid program model for a train system, then the following $\mathsf{d}\mathscr{L}$ formula expresses that this train is able ($\langle train\rangle$) to enter region $z \geq m$, thereby leaving region $z < m$ when it starts in region $z < m$ with nonnegative initial velocity $v \geq 0$:

$$v \geq 0 \wedge z < m \rightarrow \langle train\rangle\, z \geq m. \tag{2.2}$$

Dually, the following $\mathsf{d}\mathscr{L}$ formula expresses that the train will always ($[train]$) stay inside the region $z < m$ when it starts inside it with an initial nonnegative velocity less than 5:

$$v \geq 0 \wedge v < 5 \wedge z < m \rightarrow [train]\, z < m.$$

For most train models *train*, the latter safety property will only be true for additional constraints on the initial state and on the internal parameter choices, including, e.g., braking forces, reaction times, and start braking points. □

Example 2.7 (Parametric bouncing ball). Let *ball* denote the hybrid program for the bouncing ball from Example 2.5. The ball loses energy at every bounce, thus the ball never bounces higher than the initial height. This can be expressed by the property $0 \leq h \leq H$, where H denotes the initial energy level (which corresponds to the initial height if $v = 0$ initially). Then, for instance, the following $\mathsf{d}\mathscr{L}$ formula expresses that (under a list of assumptions on the free variables h, v, t and H, g, c) the ball always stays in the region $0 \leq h \leq H$:

$$(v^2 \leq 2g(H-h) \wedge h \geq 0 \wedge g > 0 \wedge H \geq 0 \wedge 1 > c \geq 0) \rightarrow [ball](0 \leq h \leq H). \tag{2.3}$$

This $\mathsf{d}\mathscr{L}$ formula follows the pattern of Hoare triples [161]. It expresses that the bouncing ball, when started in an initial state satisfying the precondition on the left of the implication ($\rightarrow$), always respects the postcondition $0 \leq h \leq H$ of the dynamic modality $[ball]$, i.e., all runs of the bouncing ball stay in the region $0 \leq h \leq H$. □

A dℒ formulas of the form $\psi \to [\alpha]\phi$ corresponds to Hoare triples [161], generalised for hybrid systems. They occur quite frequently, because they specify that system α, when starting in a state satisfying the *precondition* ψ, always respects the *postcondition* ϕ. That is, when started in a state satisfying ψ, all states reachable by α satisfy ϕ. There are several other relevant shapes of dℒ formulas in practical systems verification; see Part III.

Note that, according to Definition 2.4, hybrid programs are not additional external objects but fully internalised [48] as first-class citizens within the logic dℒ itself, and the logic is closed. That is, modalities can be combined propositionally, by quantifiers, or nested. For instance, $[\alpha]\langle\beta\rangle x \leq c$ says that, whatever hybrid program α is doing, hybrid program β can react in some way to reach a controlled state where x is less than some critical value c. That is, for all α actions, there is a β (re)action such that $x \leq c$ holds. Dually, $\langle\beta\rangle[\alpha]x \leq c$ expresses that hybrid program β can stabilise $x \leq c$, i.e., behave in such a way that $x \leq c$ remains true no matter how hybrid program α reacts. That is, there is a β action such that all α actions maintain $x \leq c$. Accordingly, $\exists p\,[\alpha]x \leq c$ says that there is a choice of parameter p such that α remains in $x \leq c$. Nesting modalities and quantifiers in this way can be quite useful for describing interactions of a hybrid program α with an environment β, or for describing the impact of parameter choices on properties of the system behaviour.

During our analysis, we assume differential equations and discrete transitions to be well-defined. In particular, we assume that all divisions p/q are guarded by conditions that ensure $q \neq 0$ as, otherwise, the system behaviour is not well-defined due to an undefined value at a singularity. It is simple but tedious to augment the semantics and the calculus with corresponding side conditions to show that this is respected. For instance, we assume that $x := p/q$ is guarded by $?q \neq 0$ and that continuous evolutions are restricted such that the differential equations are well-defined as $x' = p/q \,\&\, q \neq 0$. Also see our joint work with Beckert [37] for techniques of how such exceptional behaviour can be handled by program transformation while avoiding partial valuations of undefinedness in the semantics. In logical formulas, partiality can be avoided altogether by writing $p = c \cdot q \wedge q \neq 0$ rather than $p/q = c$, and writing $(p > c \cdot q \wedge q > 0) \vee (p < c \cdot q \wedge q < 0)$ rather than $p/q > c$.

2.3 Semantics of Differential Dynamic Logic

We define the semantics of dℒ as a possible world Kripke semantics [185] with worlds representing the possible system states and with reachability along the hybrid transitions of the system representing accessibility relations between worlds. The interpretations of dℒ consist of states (worlds) that are essentially first-order structures over the reals. In particular, real values are assigned to state variables, possibly different values in each state. A potential behaviour of a hybrid system corresponds to a succession of states that contain the observable values of system variables during its hybrid evolution.

2.3.1 Valuation of Terms

Symbols in the logic dL come from three different syntactic categories that we decided to distinguish in Sect. 2.2.1:

1. rigid symbols in Σ that cannot change their value, e.g., $0, 1, +, \cdot$;
2. flexible symbols in Σ, which are the state variables, whose value can change depending on the current state of the system;
3. logical variables in V that cannot change their value over time by running hybrid programs, but which can be quantified over universally or existentially.

All of those symbols need to be interpreted to give meaning to terms in which they occur. We associate values with rigid symbols by what we call an interpretation I, associate values with flexible symbols by a state ν, and associate values with logical variables by an assignment η. Recall that there is some leeway in declaring symbols as either rigid or flexible symbols or as logical variables. The semantics is unambiguous for each choice, though.

An *interpretation* I assigns functions and relations over the reals to the respective rigid symbols in Σ. The function and predicate symbols of real arithmetic are interpreted as usual by I. Especially, the interpretation $I(+)$ is addition and $I(\cdot)$ is multiplication of real numbers. A *state* is a map $\nu : \Sigma_{fl} \to \mathbb{R}$; the set of all states is denoted by $\mathrm{Sta}(\Sigma)$. Here, Σ_{fl} denotes the set of (flexible) state variables in Σ (they have arity 0, thus take no arguments). Finally, an *assignment for logical variables* is a map $\eta : V \to \mathbb{R}$. It contains the values for logical variables, which are not subject to change by modalities but only by quantification. Observe that flexible symbols (which represent state variables) are allowed to assume different interpretations in different states. Logical variable symbols, however, are rigid in the sense that their value is determined by η alone and does not depend on the state ν.

The *valuation* $val_{I,\eta}(\nu,\cdot)$ of terms is defined as usual [123, 149] with an extra distinction of rigid and flexible functions [37]. It is defined inductively by recursion on the structure of the term, based on the interpretation that assignment η assigns to logical variables, that interpretation I assigns to rigid function symbols, and that state ν assigns to flexible state variables. The semantics of terms is compositional and denotational [277], that is, the semantics of a complex term is defined as a combination of the semantics of its subterms.

Definition 2.5 (Valuation of terms). The *valuation of terms* with respect to interpretation I, assignment η, and state ν is defined by

1. $val_{I,\eta}(\nu,x) = \eta(x)$ if $x \in V$ is a logical variable.
2. $val_{I,\eta}(\nu,a) = \nu(a)$ if $a \in \Sigma$ is a state variable (flexible function symbol of arity 0).
3. $val_{I,\eta}(\nu, f(\theta_1,\ldots,\theta_n)) = I(f)\big(val_{I,\eta}(\nu,\theta_1),\ldots,val_{I,\eta}(\nu,\theta_n)\big)$ when $f \in \Sigma$ is a rigid function symbol of arity $n \geq 0$.

Example 2.8. Let interpretation I interpret the constant function symbol $b \in \Sigma$ as $I(b) = 2.14$, and interpret the unary function symbol $c \in \Sigma$ as the cubic function

$d \mapsto d^3$, i.e., $\big(I(c)\big)(d) = d^3$. Let the assignment η interpret the logical variable $X \in V$ as $\eta(X) = 4.2$. Finally let the state ν interpret the state variables $x, y, z \in \Sigma$ as $\nu(x) = 3$, $\nu(y) = -5.01$, $\nu(z) = 0$. Throughout this book we assume that the interpretation of $0, 1, +, -, \cdot$ always is as usual in real arithmetic, that is:

$$\begin{aligned} I(0) &= 0 \\ I(1) &= 1 \\ \big(I(+)\big)(d,e) &= d+e && \text{(addition)} \\ \big(I(-)\big)(d,e) &= d-e && \text{(subtraction)} \\ \big(I(\cdot)\big)(d,e) &= d \cdot e && \text{(multiplication)} \end{aligned}$$

With this we can valuate terms recursively with respect to I, η, ν as follows:

$$\begin{aligned} val_{I,\eta}(\nu, x+y) &= val_{I,\eta}(\nu,x) + val_{I,\eta}(\nu,y) = \nu(x) + \nu(y) \\ &= 3 + (-5.01) = -2.01, \\ val_{I,\eta}(\nu, y+2\cdot X) &= val_{I,\eta}(\nu,y) + val_{I,\eta}(\nu,2) \cdot val_{I,\eta}(\nu,X) \\ &= \nu(y) + I(2)\cdot\eta(X) = -5.01 + 2 * 4.2 = 3.39, \\ val_{I,\eta}(\nu, X + b\cdot(x+y\cdot X)) &= \eta(X) + I(b)\cdot(\nu(x) + \nu(z)\cdot\eta(X)) \\ &= 4.2 + 2.14\cdot(3+0\cdot 4.2) = 10.62, \\ val_{I,\eta}(\nu, c(x+X)-x) &= val_{I,\eta}(\nu, c(x+X)) - val_{I,\eta}(\nu,x) \\ &= I(c)\big(val_{I,\eta}(\nu, x+X)\big) - \nu(x) \\ &= I(c)\big(\nu(x) + \eta(X)\big) \; - \; \nu(x) \\ &= (3+4.2)^3 - 3 = 370.248. \end{aligned}$$

Note here, that the decision about which symbols we consider as rigid function symbols, which ones we consider as flexible function symbols (state variables), and which ones we consider as logical variables is somewhat arbitrary in this example. This decision only becomes relevant when we add quantifiers (for only logical variables can be quantified over) or hybrid programs (for only state variables can be assigned to in hybrid programs). Overall, the syntactic category of symbols is not crucial, as there are often many equivalent ways to assign symbols to syntactic categories. But if we fix a choice of symbols, the semantics becomes less subtle, so we assume a choice has been made for every formula. □

2.3.2 Valuation of Formulas

The *valuation* $val_{I,\eta}(\nu,\cdot)$ of formulas is defined as usual for first-order modal logic [123, 149] with a distinction of rigid and flexible functions [37]. Modalities parametrised by a hybrid program α follow the accessibility relation spanned by the

respective hybrid state transition relation $\rho_{I,\eta}(\alpha)$, which is simultaneously inductively defined in Definition 2.7.

The valuation of formulas is defined inductively by recursion on the structure of formulas, based on the interpretation of the terms occurring in it. The semantics of formulas is compositional and denotational, that is, the semantics of a complex formula is defined as a simple function of the semantics of its subformulas. We will use $\eta[x \mapsto d]$ to denote the *modification* of an assignment η that agrees with η except for the interpretation of the logical variable $x \in V$, which is assigned $d \in \mathbb{R}$ in $\eta[x \mapsto d]$.

Definition 2.6 (Valuation of dℒ formulas). The *valuation* $val_{I,\eta}(\nu,\cdot)$ *of formulas* with respect to interpretation I, assignment η, and state ν is defined as

1. $val_{I,\eta}(\nu, p(\theta_1,\dots,\theta_n)) = I(p)\big(val_{I,\eta}(\nu,\theta_1),\dots,val_{I,\eta}(\nu,\theta_n)\big)$.
2. $val_{I,\eta}(\nu,\phi\wedge\psi) = true$ iff $val_{I,\eta}(\nu,\phi) = true$ and $val_{I,\eta}(\nu,\psi) = true$.
3. $val_{I,\eta}(\nu,\phi\vee\psi) = true$ iff $val_{I,\eta}(\nu,\phi) = true$ or $val_{I,\eta}(\nu,\psi) = true$.
4. $val_{I,\eta}(\nu,\neg\phi) = true$ iff $val_{I,\eta}(\nu,\phi) \neq true$.
5. $val_{I,\eta}(\nu,\phi\to\psi) = true$ iff $val_{I,\eta}(\nu,\phi) \neq true$ or $val_{I,\eta}(\nu,\psi) = true$.
6. $val_{I,\eta}(\nu,\forall x\,\phi) = true$ iff $val_{I,\eta[x\mapsto d]}(\nu,\phi) = true$ for all $d \in \mathbb{R}$.
7. $val_{I,\eta}(\nu,\exists x\,\phi) = true$ iff $val_{I,\eta[x\mapsto d]}(\nu,\phi) = true$ for some $d \in \mathbb{R}$.
8. $val_{I,\eta}(\nu,[\alpha]\phi) = true$ iff $val_{I,\eta}(\omega,\phi) = true$ for all states ω for which the transition relation satisfies $(\nu,\omega) \in \rho_{I,\eta}(\alpha)$.
9. $val_{I,\eta}(\nu,\langle\alpha\rangle\phi) = true$ iff $val_{I,\eta}(\omega,\phi) = true$ for some state ω for which the transition relation satisfies $(\nu,\omega) \in \rho_{I,\eta}(\alpha)$.

Following the usual notation, we also write $I,\eta,\nu \models \phi$ iff $val_{I,\eta}(\nu,\phi) = true$. We then say that ϕ is *satisfied* in I,η,ν or holds in I,η,ν. We also say that I,η,ν is a *model* of ϕ. Dually, we write $I,\eta,\nu \not\models \phi$ iff $val_{I,\eta}(\nu,\phi) \neq true$. If ϕ is satisfied for at least one I,η,ν, then ϕ is called *satisfiable*. Occasionally, we write just $\models \phi$ iff $I,\eta,\nu \models \phi$ holds for all I,η,ν. Formula ϕ is then called *valid*, i.e., true in all I,η,ν.

The semantics of modal formulas $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ in dℒ is illustrated in Fig. 2.4, showing how the truth of ϕ at (all or some) states ω_i reachable by α relates to the truth of $[\alpha]\phi$ or $\langle\alpha\rangle\phi$ at state ν.

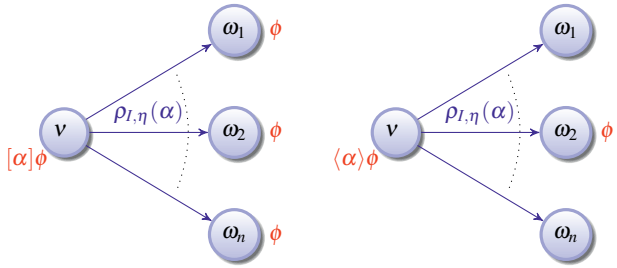


Fig. 2.4 Transition semantics of modalities in dℒ formulas

Example 2.9. Consider the following formula that we want to evaluate:

$$v > 0 \to v \cdot v \leq 2b \cdot (m - z) \lor b = 0. \tag{2.4}$$

First we have to declare which syntactic category the symbols are meant to come from. Suppose $z, v \in \Sigma$ are state variables (flexible function symbols of arity 0), because they represent position and velocity of the train, which are intended to be able to change over time from state to state. Further suppose that $m, b \in \Sigma$ are rigid function symbols, because, for the moment, movement authority and braking force are not allowed to change over time. Note that we could just as well have chosen all symbols z, v, m, b to be flexible state variables. The only notable difference is that if b is a flexible symbol, we would have to prove that a particular hybrid program never changes the value of b to know that it denotes the same value in every part of the program. Otherwise, if b is a rigid symbol, we already know that it cannot possibly change its value by running a hybrid program, because b then is a constant, and only flexible symbols are syntactically allowed to be assigned to or have differential equations in Definition 2.3. While it is certainly not crucial to make this distinction, it can make some things easier to see syntactically.

Now let interpretation I interpret rigid symbol $m \in \Sigma$ as $I(m) = 20$ and interpret $b \in \Sigma$ as $I(b) = 2.2$. Let state ω interpret state variables $v, z \in \Sigma$ as $\omega(v) = 10$, $\omega(z) = 0$. In formula (2.4), suppose we do not have any free logical variables, so that the assignment η of logical variables does not matter. Then we can evaluate formula (2.4) with respect to I, η, ω:

$$\begin{aligned}
&val_{I,\eta}(\omega, v > 0 \to v \cdot v \leq 2b \cdot (m - z) \lor b = 0) = \mathit{true} \text{ iff} \\
&val_{I,\eta}(\omega, v > 0) \neq \mathit{true}, \text{ or} \\
&val_{I,\eta}(\omega, v \cdot v \leq 2b \cdot (m - z)) = \mathit{true}, \text{ or} \\
&val_{I,\eta}(\omega, b = 0) = \mathit{true}.
\end{aligned}$$

Let us evaluate the terms to determine if the subformulas are true or not:

$$\begin{aligned}
val_{I,\eta}(\omega, v > 0) &= (val_{I,\eta}(\omega, v) \overset{?}{>} val_{I,\eta}(\omega, 0)) = (\omega(v) \overset{?}{>} I(0)) \\
&= (10 \overset{?}{>} 0) = \mathit{true}, \\
val_{I,\eta}(\omega, v \cdot v \leq 2b \cdot (m - z))&(val_{I,\eta}(\omega, v \cdot v) \overset{?}{\leq} val_{I,\eta}(\omega, 2b \cdot (m - z))) \\
&= \big(\omega(v) \cdot \omega(v) \overset{?}{\leq} 2I(b) \cdot (I(m) - \omega(z))\big) \\
&= \big(10 \cdot 10 \overset{?}{\leq} 2 \cdot 2.2 \cdot (20 - 0)\big) = (100 \overset{?}{\leq} 88) = \mathit{false}, \\
val_{I,\eta}(\omega, b = 0) &= (I(b) \overset{?}{=} 0) = (2.2 \overset{?}{=} 0) = \mathit{false}.
\end{aligned}$$

Consequently the formula (2.4) evaluates to *false*. For a different state ν with slower speed $\nu(v) = 8$ and the same position $\nu(z) = 0$, we instead evaluate (2.4) to *true*:

$$val_{I,\eta}(\nu, v > 0 \to v \cdot v \leq 2b \cdot (m - z) \vee b = 0) = true.$$

Also for a different interpretation J with $J(m) = 20$ and another braking force $J(b) = 4$, but the same original state ω, we evaluate (2.4) to *true*:

$$val_{J,\eta}(\omega, v > 0 \to v \cdot v \leq 2b \cdot (m - z) \vee b = 0) = true.$$

So we see that the truth-value of formula (2.4) depends on I, η, ω. For some choices of I, η, ω, it evaluates to *true*, for others it evaluates to *false*. Thus, formula (2.4) is not valid, because

$$I, \eta, \omega \models v > 0 \to v \cdot v \leq 2b \cdot (m - z) \vee b = 0$$

does not hold for all I, η, ω. Still, the formula (2.4) is at least satisfiable, because it holds for some I, η, ω. □

Example 2.10. Consider the assignment η with $\eta(Z) = -2$ and the state ν with $\nu(x) = -4$. Then we can evaluate

$$val_{I,\eta}(\nu, x > -5 \wedge \forall y\, (y^2 + Z > x)) = true$$

because $\nu(x) > -5$ and all squares are greater than or equal zero, so that for all $d \in \mathbb{R}$:

$$\begin{aligned} val_{I,\eta[y \mapsto d]}(\nu, y^2 + Z > x) &= \big((\eta[y \mapsto d](y))^2 + \eta[y \mapsto d](Z) \overset{?}{>} \nu(x) \\ &= \big(d^2 + (-2) \overset{?}{>} -4\big) = true. \end{aligned}$$

□

Note, that we have not yet explained how to evaluate formulas with modalities like $x > 0 \to [ctrl; drive^*]\, z \leq m$ in any I, η, ν, because we first have to define the transition semantics $\rho_{I,\eta}(\alpha)$ of hybrid programs, which we will do next.

2.3.3 Transition Semantics of Hybrid Programs

Now we define the transition semantics, $\rho_{I,\eta}(\alpha)$, of hybrid program α. The semantics of a hybrid program is captured by its hybrid state transition relation. For discrete jumps this transition relation holds for pairs of states that respect the discrete jump set. For continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous flow respecting the differential equations and evolution domain restriction throughout the evolution.

The transition semantics of hybrid programs is defined by induction based on the structure of the programs. The semantics of hybrid programs is compositional, that is, the semantics of a complex program is defined as a simple function of the

transition semantics of its parts. We will use $\nu[x \mapsto d]$ to denote the *modification* of a state ν that agrees with ν except for the interpretation of the symbol $x \in \Sigma_{fl}$, which is changed to $d \in \mathbb{R}$ in $\nu[x \mapsto d]$.

Definition 2.7 (Transition semantics of hybrid programs). The *valuation of a hybrid program* α, denoted by $\rho_{I,\eta}(\alpha)$, is a *transition relation* on states. It specifies which state ω is reachable from a state ν by operations of the hybrid program α and is defined as follows

1. $(\nu,\omega) \in \rho_{I,\eta}(x_1 := \theta_1,\ldots,x_n := \theta_n)$ iff the state ω equals the state obtained by semantic modification of state ν as $\nu[x_1 \mapsto val_{I,\eta}(\nu,\theta_1)]\ldots[x_n \mapsto val_{I,\eta}(\nu,\theta_n)]$. Particularly, the values of other variables $z \notin \{x_1,\ldots,x_n\}$ remain constant, i.e., $val_{I,\eta}(\omega,z) = val_{I,\eta}(\nu,z)$, and the x_i receive their new values simultaneously, i.e., $val_{I,\eta}(\omega,x_i) = val_{I,\eta}(\nu,\theta_i)$.
2. $(\nu,\omega) \in \rho_{I,\eta}(x_1' = \theta_1,\ldots,x_n' = \theta_n \,\&\, \chi)$ iff there is a *flow* f of some duration $r \geq 0$ from state ν to state ω along $x_1' = \theta_1,\ldots,x_n' = \theta_n \,\&\, \chi$, i.e., a function $f : [0,r] \to \mathrm{Sta}(\Sigma)$ such that:
   - $f(0) = \nu, f(r) = \omega$;
   - f respects the differential equations: For each variable x_i, the valuation $val_{I,\eta}(f(\zeta),x_i) = f(\zeta)(x_i)$ of x_i at state $f(\zeta)$ is continuous in ζ on $[0,r]$ and has a derivative of value $val_{I,\eta}(f(\zeta),\theta_i)$ at each time $\zeta \in (0,r)$;
   - the value of other variables $z \notin \{x_1,\ldots,x_n\}$ remains constant, that is, we have $val_{I,\eta}(f(\zeta),z) = val_{I,\eta}(\nu,z)$ for all $\zeta \in [0,r]$;
   - and f respects the invariant: $val_{I,\eta}(f(\zeta),\chi) = \mathit{true}$ for each $\zeta \in [0,r]$.
3. $\rho_{I,\eta}(?\chi) = \{(\nu,\nu) \,:\, val_{I,\eta}(\nu,\chi) = \mathit{true}\}$
4. $\rho_{I,\eta}(\alpha \cup \beta) = \rho_{I,\eta}(\alpha) \cup \rho_{I,\eta}(\beta)$
5. $\rho_{I,\eta}(\alpha;\beta) = \{(\nu,\omega) : (\nu,\mu) \in \rho_{I,\eta}(\alpha), (\mu,\omega) \in \rho_{I,\eta}(\beta) \text{ for a state } \mu\}$
6. $(\nu,\omega) \in \rho_{I,\eta}(\alpha^*)$ iff there is an $n \in \mathbb{N}$ and states $\nu = \nu_0,\ldots,\nu_n = \omega$ such that $(\nu_i,\nu_{i+1}) \in \rho_{I,\eta}(\alpha)$ for all $0 \leq i < n$.

For graphical illustrations of the transition semantics of hybrid programs and example dynamics, see Fig. 2.5. On the left of Fig. 2.5, we illustrate the generic shape of the transition structure $\rho_{I,\eta}(\alpha)$ for transitions along various cases of hybrid programs α from state ν to state ω. On the right of Fig. 2.5, we show examples of how the value of a variable x may evolve over time t when following the dynamics of the respective hybrid program α. The shape of the transition structure of a discrete jump $x := \theta$ (row 1) and of a differential equation $x' = \theta \,\&\, \chi$ (row 2) is an elementary one-step transition from ν to ω. For discrete jumps, however, the transition is an instant jump in the state space (row 1 on the right), while the transition for a differential equation is a continuous evolution in the state space (row 2 on the right). Note that the modifications of a discrete jump set $x_1 := \theta_1,..,x_n := \theta_n$ are executed simultaneously in Definition 2.7 in the sense that all terms θ_i are evaluated in the initial state ν. For simplicity, we assume the x_i to be different, and refer to previous work [37] for a compatible semantics and calculus handling concurrent modifications of the same x_i.

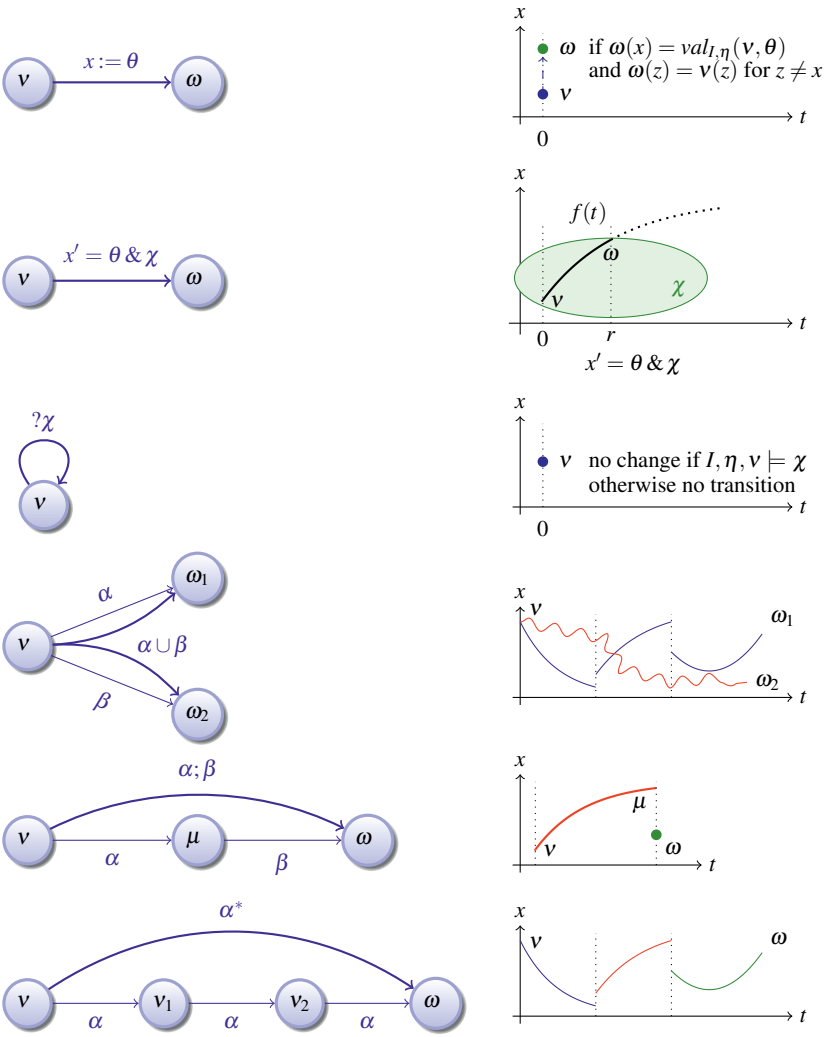


Fig. 2.5 Transition semantics (left) and example dynamics (right) of hybrid programs

For test $?\chi$ (row 3), the only possible transitions in the transition structure are self-loops that do not change the state ν, but even those transitions are only possible if the test succeeds, i.e., $I, \eta, \nu \models \chi$; see Fig. 2.5. The transition structure for choice $\alpha \cup \beta$ (row 4) is a choice between any transition of α and any transition of β. Thus, in the example on the right of row 4, the system can choose between either an evolution like the hybrid evolution (consisting, in this example, of 3 continuous flows and 2 intermediate jumps) leading to ω_1 or the squiggly evolution from ν to ω_2. The transition structure for sequential composition $\alpha;\beta$ (row 5) is that of any α transition to an intermediate state μ, followed by any β transition to ω. In the

example evolution on the right of row 5, the system first follows a continuous evolution (which would come from α in this example) and then a discrete jump (which would come from β). The transition structure of a repetition α^* (row 6) repeats any number of α transitions to go from ν to ω via some number of intermediate states ν_i. In the example on the right, the system follows a sequence of various continuous evolutions and discrete jumps, giving truly hybrid behaviour.

For differential equations like $x' = \theta$, Definition 2.7 characterises transitions along a continuous evolution respecting the differential equation; see Fig. 2.6a. A continuous transition along $x' = \theta$ is possible from state ν to state ω whenever there is a continuous flow f of some duration $r \geq 0$ connecting state ν with ω such that f gives a solution of the differential equation $x' = \theta$. That is, its value is continuous on the closed interval $[0,r]$ and differentiable with the value of θ as derivative on the open interval $(0,r)$. Further, only variables subject to a differential equation change during such a continuous transition. Similarly, the continuous transitions of $x' = \theta \,\&\, \chi$ with evolution domain χ are those where f always resides within χ during the whole evolution; see Fig. 2.6b. The evolutions of $x' = \theta \,\&\, \chi$ may still stop at any point in time, but they are no longer allowed to leave χ and have to stop at an arbitrary point in time before that happens; see Fig. 2.6c.

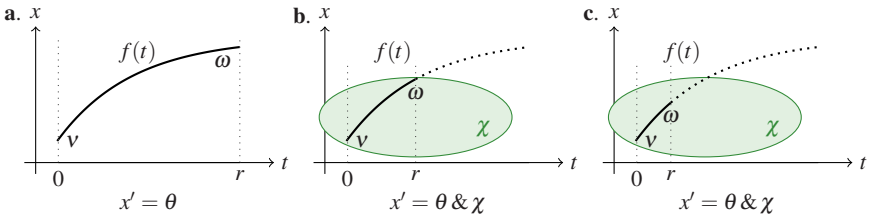


Fig. 2.6 Continuous flow along differential equation $x' = \theta$ over time t

For the semantics of differential equations, derivatives are well defined on the open interval $(0,r)$, because the set $\mathrm{Sta}(\Sigma)$ of states is isomorphic to some finite-dimensional metric real vector space spanned by the variables of the differential equations (derivatives are not defined on the closed interval $[0,r]$ if $r = 0$). For the purpose of a differential equation system, states are fully determined by an assignment of a real value to each occurring variable, which are finitely many. Furthermore, the terms of $\mathsf{d}\mathscr{L}$ are continuously differentiable on the open domain where divisors are nonzero, because the zero set of divisors is closed. Hence, solutions in $\mathsf{d}\mathscr{L}$ are unique:

Lemma 2.1 (Uniqueness). *Differential equations of $\mathsf{d}\mathscr{L}$ have unique solutions, i.e., for each differential equation system, each state ν, and each duration $r \geq 0$, there is at most one flow $f : [0,r] \to \mathrm{Sta}(\Sigma)$ satisfying the conditions of Case 2 of Definition 2.7.*

Proof. Let $x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi$ be a differential equation system with evolution domain χ. Using simple computations in the field of rational fractions, we can as-

sume the right-hand sides θ_i of the differential equations to be of the form p_i/q_i for polynomials p_i, q_i. The set of points in real space where $q_i = 0$ holds is closed. As a finite union of closed sets, the set where $q_1 = 0 \vee \cdots \vee q_n = 0$ holds is closed. Hence, the valuations of the θ_i are continuously differentiable on the complement of the latter set, which is open. Thus, as a consequence of Picard-Lindelöf's theorem, a.k.a. the Cauchy-Lipschitz theorem (Theorem B.2), the solutions are unique on each connected component of this open domain. Consequently, solutions are unique when restricted to χ, which, by assumption, entails $q_1 \neq 0 \wedge \cdots \wedge q_n \neq 0$.

□

Example 2.11 (Evaluation of formula and transition semantics). Recall the following hybrid program from Example 2.4 that models an (overly) simplistic train controller:

$$train \equiv \big(((a := -b) \cup (?v < 8; a := A)); \, z' = v, v' = a\big)^* \tag{2.1*}$$

Recall the $\mathsf{d}\mathscr{L}$ formula (2.2) from p. 48 that claims that this simplistic train model can leave the movement authority region m:

$$v \geq 0 \wedge z < m \rightarrow \langle train \rangle z \geq m \tag{2.2*}$$

Let us evaluate this $\mathsf{d}\mathscr{L}$ formula. Consider the interpretation I that interprets rigid symbol $m \in \Sigma$ as $I(m) = 20$ and interprets $b \in \Sigma$ as $I(b) = 2$ and $I(A) = 1$. Let state ν interpret state variables $v, z \in \Sigma$ as $\nu(v) = 9$, $\nu(z) = 0$. Then the assumptions $v \geq 0 \wedge z < m$ from the left-hand side of the implication of (2.2) are satisfied, so for formula (2.2) to be evaluated to true, the right-hand side of the implication needs to evaluate to $val_{I,\eta}(\nu, \langle train \rangle z \geq m) = true$. To find out if this is the case, the semantics of $\langle train \rangle$ in Definition 2.6 requires us to find a transition $(\nu, \omega) \in \rho_{I,\eta}(train)$ of the hybrid program *train* from ν to some state ω, according to Definition 2.7, after which $val_{I,\eta}(\omega, z \geq m)$ holds true. Let us try to find such a state ω by following the transition structure $\rho_{I,\eta}(train)$ depicted in Fig. 2.7a. Essentially, we obtain the transition structure in Fig. 2.7a by gluing the elementary transition patterns from Fig. 2.5 together according to the structure of hybrid program (2.1). We will find a path in the transition structure Fig. 2.7a from ν to ω along the transitions illustrated in Fig. 2.7b, as we explain in the following.

The top-level statement in *train* is a repetition (corresponding to the outer loop in Fig. 2.7a). We are allowed to execute the repetition twice as illustrated in the double unrolling in Fig. 2.7b (we could also repeat it any other number of times, but two times is sufficient). Thus, we hope to find an intermediate state σ_2 such that both

$$(\nu, \sigma_2) \in \rho_{I,\eta}(((a := -b) \cup (?v < 8; a := A)); \, z' = v, v' = a) \tag{2.5}$$

and

$$(\sigma_2, \omega) \in \rho_{I,\eta}(((a := -b) \cup (?v < 8; a := A)); \, z' = v, v' = a) \tag{2.6}$$

In the first transition (2.5), the top-level statement is a sequential composition (;) with a nondeterministic choice ($\cup$) as its first action. This nondeterministic choice can choose either side, indicated as upper and lower choices on the left of Fig. 2.7b.

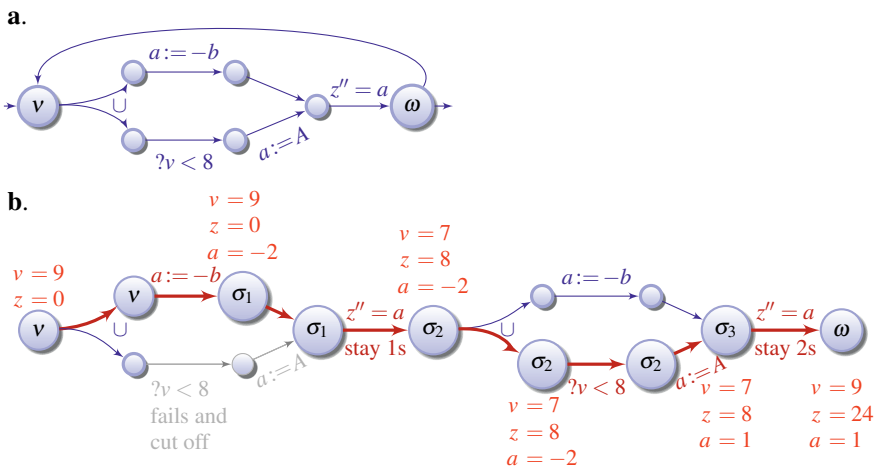


Fig. 2.7 Transition structure and transition example in (overly) simple train control

If it chooses to try to run the second (lower) choice $(?v < 8; a := A)$, however, the hybrid program cannot run successfully, because the test $?v < 8$ will fail and abort the transition as a dead end, since this test evaluates to $val_{I,\eta}(\nu, v < 8) = \mathit{false}$ at state ν. Hence, the hybrid program can only choose the first (upper) option $(a := -b)$ to a state σ_1 whose only difference with ν is that $\sigma_1(a) = -2 = val_{I,\eta}(\nu, -b)$. For this state, we have $(\nu, \sigma_1) \in \rho_{I,\eta}(a := -b)$. Next, the differential equation will run from σ_1 as the second part of the sequential composition. Because $\sigma_1(a) < 0$, it will brake and the velocity v will decrease over time. If we just follow this differential equation long enough, say for one second, then the velocity at the end of it will be less than 8. Indeed, after staying in the differential equation for one second, we reach a state σ_2 with $(\sigma_1, \sigma_2) \in \rho_{I,\eta}(z' = v, v' = a)$ and $\sigma_2(v) = 7$ and $\sigma_2(z) = 8$, because $z(t) := \frac{-2}{2}t^2 + 9t + 0$ and $v(t) := -2t + 9$ is the solution of the differential equation when starting in state σ_1 with the interpretation I and staying for t time units. Thus, by the semantics of sequential composition and nondeterministic choice, Definition 2.7, we have that relation (2.5) holds for this state σ_2, and all we need to do is make sure that relation (2.6) holds as well.

For the transition (2.6), we can choose the second (lower) part of the nondeterministic choice, because, unlike before, the test $?v < 8$ succeeds in the new state now: $val_{I,\eta}(\sigma_2, v < 8) = \mathit{true}$. Hence, we follow $?v < 8; a := A$ to the state σ_3 that is like σ_2 except that we now have $\sigma_3(a) = I(A) = 1$. From state σ_3, we can stay with and follow the subsequent differential equation as long as we want to, because there is no evolution domain restriction on it. From this particular initial state σ_3, the solution of the differential equation is $z(t) := \frac{1}{2}t^2 + 7t + 8$ and $v(t) := 1t + 7$ when staying for t time units. If now we just follow the continuous evolution along this differential equation for long enough, we will eventually reach a state ω with $(\sigma_3, \omega) \in \rho_{I,\eta}(z' = v, v' = a)$ such that $val_{I,\eta}(\omega, z \geq m) = \mathit{true}$. In fact, the minimum time for this to happen is 1.544 time units, after which z has

a value greater or equal $I(m) = 20$. But any longer period of time will do too. For instance, after 2 time units, we would have $\omega(z) = 24 \geq 20$ and $\omega(v) = 9$. Thus we have shown $I, \eta, \omega \models z \geq m$ and

$$(v, \omega) \in \rho_{I,\eta}\big(\big(((a := -b) \cup (?v < 8; a := A)); \, z' = v, v' = a\big)^*\big) \ ,$$

which implies that formula (2.2) holds for I, η, v.

So far, we have shown by semantic reasoning that formula (2.2) is true for I, η, v. Yet formula (2.2) does not evaluate to *true* under all interpretations and states. For a different interpretation J with $J(m) = 1{,}000$, braking force $J(b) = 4$, and (now negative) acceleration $J(A) = -2$, but the same original state v, we evaluate (2.4) to *false*. The reason is that, no matter which choice the hybrid program uses, the train always brakes, either with braking acceleration $-J(b) = -4$ or with negative acceleration $J(A) = -2$. Either way, the initial velocity $v(v) = 9$ is not high enough to reach $I(m) = 1{,}000$ from the initial position $v(z) = 0$. Eventually, the train velocity will be 0 and it cannot move forward anymore.

Another trivial example to show that formula (2.2) can evaluate to *false* for some J, η, ω is the interpretation J with $J(m) = 20$, $J(b) = 2$, and $I(A) = 0$ along with the state ω that interprets $\omega(v) = 0$, $\omega(z) = 0$. Then, the train stands still in the beginning and cannot move forward to $I(m) = 20$ at all. In particular, formula (2.2) is not valid, because it does not evaluate to *true* for all I, η, v. □

What we notice in this example is that it is quite difficult and cumbersome to reason about $\mathsf{d}\mathscr{L}$ formulas and the dynamics of hybrid systems on the level of semantics. Especially, we have only analysed particular behaviours starting at specific initial values for all the variables. For validity, we are interested in analysing all possible initial values, all numbers of repetitions, and arbitrary durations of staying in the continuous evolution modes. To do this in an elegant and coherent way, we introduce a proof calculus for $\mathsf{d}\mathscr{L}$ in Sect. 2.5. After all, the semantics gives meaning to formulas and captures the intended meaning and behaviour in real systems. The semantics is intended to be intuitive to relate to the real world, not necessarily for being easy to use in meta-reasoning. Supporting simple analysis and proofs is the task of the proof calculus for $\mathsf{d}\mathscr{L}$ that we develop in Sect. 2.5. Still, a good semantics like the one we chose is compositional, which makes the proof calculus simpler.

Further note that, for control-feedback loops α with a discrete controller regulating a continuous plant, transition structures involve all safety-critical states; hence, $\psi \to [\alpha]\phi$ is a natural rendition of the safety property that ϕ holds at all states reachable by α from initial states that satisfy ψ. Otherwise, $\mathsf{d}\mathscr{L}$ can be augmented with temporal operators to refer to intermediate states or nonterminating traces. The corresponding calculus is compatible and reduces temporal properties to nontemporal properties at intermediate states of the hybrid program, as we illustrate in Chap. 4.

2.4 Collision Avoidance in Train Control

As a case study to illustrate how dℒ can be used for specifying and verifying hybrid systems, we examine a scenario of cooperating traffic agents in the European Train Control System (ETCS) [91]. The purpose of ETCS is to ensure that trains cannot crash into other trains or pass open gates. Its secondary objective is to maximise throughput and velocity without endangering safety. To achieve these objectives, ETCS discards the static partitioning of the track into fixed segments of mutually exclusive and physically separated access by trains, which has been used traditionally. Instead, permission to move is granted dynamically by decentralised Radio Block Controllers (RBCs) depending on the current track situation and movement of other traffic agents within the region of responsibility of the RBC; see Fig. 2.8.

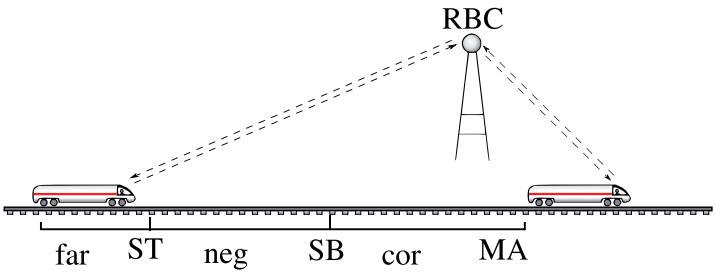


Fig. 2.8 ETCS train coordination protocol using dynamic movement authorities

Movement Authorities

This moving block principle is achieved by dynamically giving a *movement authority* (MA) to each traffic agent, within which it is obliged to remain. Before a train moves into a part of the track for which it does not have MA, it asks the RBC for an MA extension (during the negotiation phase indicated *neg* in Fig. 2.8). Depending on the MA that the RBC has currently given to other traffic agents or gates, the RBC will grant this extension and the train can move on. If the requested MA extension is still in the possession of another train that could possibly occupy the same part of the track, or if the MA is still consumed by an open gate, the RBC will deny the MA extension such that the requesting train needs to reduce speed or start braking in order to safely remain within its old MA. This is the correction phase *cor* in Fig. 2.8, which has to happen at the point SB (for start braking) at the latest. As the negotiation process with the RBC can take time because of possibly unreliable wireless communication and negotiation of the RBC with other agents, the train initiates negotiation well before reaching the end of its MA. This negotiation phase *neg* starts at the start talking point ST at the latest. Only if the train has a very large distance

to the end of its MA (phase *far* in Fig. 2.8) is it safe to drive freely and not yet necessary to request MA extensions. When the rear end of a train has safely left a part of a track, the train can give that part of its MA back to RBC control such that it can be used by other traffic agents, including trains or gates.

In addition to increased flexibility and throughput of this moving block principle, the underlying technical concept of movement authorities can be exploited for verifying ETCS. It can be shown that a system of arbitrarily many trains, gates, and RBCs, which communicate in the aforementioned manner, safely avoids collisions if each traffic agent always resides within its MA under all circumstances, provided that the RBCs grant MAs mutually exclusively so that the MAs dynamically partition the track (Chap. 7). This way, verification of a system of unboundedly many traffic agents can be reduced to an analysis of individual agents with respect to their specific MA.

Train Control Model

In trains, speed supervision and automatic train protection are responsible for locally controlling the movement of a train such that it always respects its MA [90]. Depending on the current driving situation, the train controller determines a point SB (for start braking) up to which driving is safe, and adjusts its acceleration a in accordance with SB. Before SB, speed can be regulated freely (to keep the desired speed and throughput of a track profile). Beyond SB (correcting phase *cor* in Fig. 2.8), the train starts braking in order to make sure it remains within its MA if the RBC does not grant an extension in time.

We assume that an MA has been granted up to some track position, which we call m, and the train is located at position z, heading with current speed v towards m. We represent the point SB as the safety distance s relative to the end m of the MA (i.e., $m-s=\text{SB}$). In this situation, dL can analyse the following crucial safety property of ETCS, which we state as a dL formula:

$$\begin{aligned} \psi &\rightarrow [(ctrl;drive)^*]z \leq m \qquad (2.7)\\ \text{where } ctrl &\equiv (?m-z \leq s; a:=-b) \cup (?m-z \geq s; a:=A),\\ drive &\equiv \tau:=0; (z'=v, v'=a, \tau'=1 \,\&\, v \geq 0 \wedge \tau \leq \varepsilon). \end{aligned}$$

It expresses that a train always ($[(ctrl;drive)^*]$) remains within its MA ($z \leq m$), assuming some constraint ψ for its parameters. The operational system model is a control-feedback loop of the digital controller *ctrl* and the plant *drive*. In *ctrl*, the train controller corrects its acceleration or brakes on the basis of the remaining distance ($m-z$). As a fail-safe recovery manoeuvre [90], it applies brakes with force b if the remaining MA is less than or equal to s. Otherwise, speed is regulated freely. The controller *ctrl* has a nondeterministic choice ($\cup$) where the left option starts with test $?m-z \leq s$ and the right option starts with test $?m-z \geq s$. The controller can try both options, but the left test will only succeed if $m-z \leq s$ holds for the current state, and the right test will only succeed if $m-z \geq s$ holds. In particular, if

$m-z<s$ holds the controller can only choose the left option, leading to braking by the assignment $a:=-b$. If $m-z>s$ holds the controller can only choose the right option, leading to acceleration by the assignment $a:=A$. If both tests could succeed, i.e., $m-z=s$, then either choice can be taken, nondeterministically. For simplicity, we assume the train uses a fixed acceleration A before passing s and does not choose any other accelerations than full braking b and full acceleration A (bang-bang control). The verification is quite similar when the controller can dynamically choose any acceleration $a \leq A$ instead, as we illustrate in Chap. 7.

After acceleration a has been set in *ctrl*, the second half of the sequential composition *ctrl*;*drive* executes, and the train continues moving in *drive*. There, the position z of the train evolves according to the differential equation system $z' = v, v' = a$ (i.e., $z'' = a$). The evolution in *drive* stops when the speed v drops below zero (or earlier), because the train would not drive backwards just by braking. Thus, $v \geq 0$ is in the maximum evolution domain of *drive*. Simultaneously, clock τ measures the duration of the current *drive* phase before the controllers react to situation changes again. Clock τ is reset to zero by $\tau:=0$ when entering *drive*, constantly evolves along $\tau' = 1$ together with the differential equations $z' = v, v' = a$, and is restricted by the evolution domain $\tau \leq \varepsilon$. Hence, the system can only follow *drive* for up to ε time units and at most as long as $v \geq 0$. The effect is that a *drive* phase is interrupted for reassessing the driving situation after at most ε seconds, and the *ctrl*;*drive* feedback loop repeats by the repetition operator $(^*)$. In particular, the continuous evolution cannot just be followed indefinitely without giving the controller *ctrl* a chance to react to situation changes. The corresponding transition structure $\rho_{I,\eta}((ctrl;drive)^*)$ is depicted in Fig. 2.9a. Essentially, we obtain the transition structure in Fig. 2.9a by gluing the elementary transition patterns from Fig. 2.5 together according to the structure of the hybrid program in (2.7).

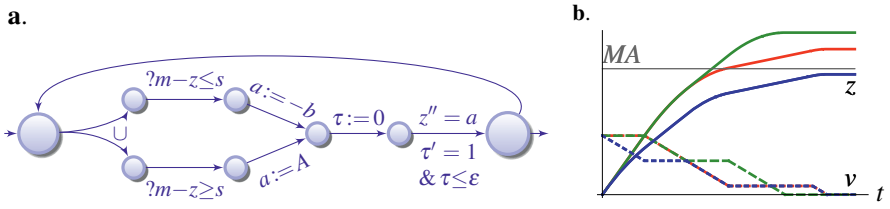


Fig. 2.9 ETCS transition structure and various choices of speed regulation for train speed control

Figure 2.9b shows possible runs of the train where speed regulation successively decreases velocity v because its MA has not been extended in time. Figure 2.9b shows three different runs (three upper position curves and three lower, partially overlapping velocity curves) which correspond to different choices of parameter s, where only the lowest velocity choice is safe. Finally, observe that the evolution domain $v \geq 0 \wedge \tau \leq \varepsilon$ needs to be true at *all times* during continuous evolutions of *drive*; otherwise there is no corresponding transition in $\rho_{I,\eta}(drive)$. This not only restricts the maximum duration of *drive*, but also imposes a constraint on permitted

initial states: The arithmetic constraint $v \geq 0$ expresses that the differential equation only applies for nonnegative speed. Hence, as in a test $?v \geq 0$, program *drive* allows no transitions at all when v is initially less than 0. In that case, $\rho_{I,\eta}((ctrl;drive)^*)$ collapses to the trivial identity transition where only zero repetitions are possible.

Discussion

Here, we explicitly take into account possibly delayed controller reactions to bridge the gap of continuous-time models and discrete-time control design. To get meaningful results, we need to assume a maximum reaction delay ε, because safety cannot otherwise be guaranteed (the system would not be safe if the controllers can never execute). Polling cycles of sensors and digital controllers as well as latencies of actuators such as brakes contribute to ε. Instead of using specific estimates for ε for a particular train, we accept ε as a fully symbolic parameter. Further, instead of manually choosing specific values for the free parameters of (2.7) as in model checking approaches [91], we will use our calculus to synthesise constraints on the relationship of parameters that are required for safe operation of train control. We do not model weather conditions, slope of track, wheel friction, or train mass, because these are less relevant for the cooperation layer of train control [90].

Because of its nonlinear behaviour and nontrivial reset relations, system (2.7) is beyond the modelling capabilities of linear hybrid automata [8, 156, 126] and beyond o-minimal automata [189]. Previous approaches need linear flows [8, 156], do not support the coupled dynamics caused by nontrivial resets [189], require polyhedral initial sets and discrete dynamics [70], only handle robust systems with bounded regions [125] although parametric systems are not robust uniformly for all parameter choices, or handle only bounded-time safety for systems with bounded switching [217]. Finally, in addition to general numerical limits [238], numerical approaches [70, 21] quickly become intractable due to the exponential impact of the number of variables (curse of dimensionality).

2.5 Free-Variable Proof Calculus for Differential Dynamic Logic

In this section, we introduce a sequent calculus for formally verifying hybrid systems by proving validity of corresponding $\mathsf{d}\mathscr{L}$ formulas. The basic idea is to symbolically compute the effects of hybrid programs and successively transform them into logical formulas describing these effects by structural symbolic decomposition. The calculus consists of standard propositional rules, rules for dynamic modalities that are generalised to hybrid programs, and novel quantifier rules that integrate real quantifier elimination (or, in fact, any other quantifier elimination procedure) into the modal calculus using free variables and Skolemisation.

2.5.1 Substitution

The $\mathsf{d}\mathscr{L}$ calculus uses substitutions that take effect within formulas and programs. The result of applying to a $\mathsf{d}\mathscr{L}$ formula ϕ the *substitution* that simultaneously replaces variable y_i by term θ_i (for $1 \leq i \leq m$) is defined as usual. Figure 2.10 shows

$$
\begin{array}{rcll}
\sigma(y_i) & = & \theta_i & \text{for } 1 \leq i \leq n \\
\sigma(z) & = & y & \text{if } z \notin \{y_1, \ldots, y_m\} \text{ is a variable} \\
\sigma(f(\theta_1, \ldots, \theta_n)) & = & f(\sigma(\theta_1), \ldots, \sigma(\theta_n)) & \text{if } f \text{ is a function symbol} \\
\\
\sigma(p(\theta_1, \ldots, \theta_n)) & = & p(\sigma(\theta_1), \ldots, \sigma(\theta_n)) & \text{if } p \text{ is a predicate symbol} \\
\sigma(\neg\phi) & = & \neg\sigma(\phi) & \\
\sigma(\phi \wedge \psi) & = & \sigma(\phi) \wedge \sigma(\psi) & \\
\sigma(\phi \vee \psi) & = & \sigma(\phi) \vee \sigma(\psi) & \\
\sigma(\phi \rightarrow \psi) & = & \sigma(\phi) \rightarrow \sigma(\psi) & \\
\sigma(\forall x\, \phi) & = & \forall x\, \sigma(\phi) & \text{if admissible} \\
\sigma(\exists x\, \phi) & = & \exists x\, \sigma(\phi) & \text{if admissible} \\
\sigma([\alpha]\phi) & = & [\sigma(\alpha)]\sigma(\phi) & \text{if admissible} \\
\sigma(\langle\alpha\rangle\phi) & = & \langle\sigma(\alpha)\rangle\sigma(\phi) & \text{if admissible} \\
\\
\sigma(x_1 := \theta_1, .., x_n := \theta_n) & = & x_1 := \sigma(\theta_1), .., x_n := \sigma(\theta_n) & \text{if admissible} \\
\sigma(x_1' = \theta_1, .., x_n' = \theta_n \,\&\, \chi) & = & x_1' = \sigma(\theta_1), .., x_n' = \sigma(\theta_n) \,\&\, \sigma(\chi) & \text{if admissible} \\
\sigma(?\chi) & = & ?\sigma(\chi) & \\
\sigma(\alpha;\beta) & = & \sigma(\alpha);\sigma(\beta) & \\
\sigma(\alpha \cup \beta) & = & \sigma(\alpha) \cup \sigma(\beta) & \\
\sigma(\alpha^*) & = & (\sigma(\alpha))^* &
\end{array}
$$

Fig. 2.10 Application of substitution σ that simultaneously replaces variable y_i by term θ_i (for $1 \leq i \leq m$)

how the substitution σ that replaces variable y_i by term θ_i (for each $1 \leq i \leq m$) can be applied to a term, $\mathsf{d}\mathscr{L}$ formula, or hybrid program, respectively. The first line in Fig. 2.10 represents that the substitution σ matches on the replaced (logical or state) variables y_i and replaces them by θ_i, respectively. The second line represents that no logical or state variable z other than $y_1, \ldots, y_n$ are affected by σ. The third line maps the substitution σ homomorphically over function applications by applying σ recursively to all argument terms. Similarly, the next block of cases in Fig. 2.10 maps substitutions homomorphically over all subformulas. Yet for quantifiers $(\forall, \exists)$ and modalities $([\alpha], \langle\alpha\rangle)$, the substitution is only applicable if admissible (as defined below) so that the bound variable x of the quantifier does not interfere with the substitution. We assume *bound variable renaming* (also known as α conversion) for renaming as needed: bound variables can be renamed to resolve conflicts, e.g., $\forall x\, \phi(x) \equiv \forall z\, \phi(z)$. Likewise, for applying the substitution homomorphically to hybrid programs (last block in Fig. 2.10) admissibility of the substitution is crucial in all cases. Admissibility implies, for instance, that the variables y_i replaced by the

substitution are different from the changed variables x_j on the left-hand sides of the assignments or differential equations of the hybrid program.

Definition 2.8 (Admissible substitution). An application of a substitution σ is *admissible* if no variable x that σ replaces by $\sigma(x)$ occurs in the scope of a quantifier or modality binding x or a (logical or state) variable of the replacement $\sigma(x)$. A modality *binds* a state variable x iff it contains a discrete jump set assigning to x (such as $x := \theta$) or a differential equation containing x' (such as $x' = \theta$).

In this book, only admissible substitutions are applicable, which is crucial for soundness. Admissible substitutions are denotation-preserving: They ensure that symbols still denote the same values after a substitution when they did so before.

Example 2.12 (Non-admissible substitution). It is important that only admissible substitutions are applicable. For the following formula, ϕ,

$$x = z \rightarrow \langle z := z+1 \rangle (z \geq x+1),$$

the substitution σ that replaces all occurrences of x by z is not admissible. This is due to the fact that for when we try to apply σ to ϕ forming

$$z = z \rightarrow \langle z := z+1 \rangle (z \geq z+1),$$

the substitution replaces x in postcondition $z \geq x+1$ by z, which is bound by modality $\langle z := z+1 \rangle$. Hence, within the scope of the modality, symbol z denotes a different value than outside the modality, thereby destroying the property of the occurrences of x—or, after the substitution, those of z—to share the same value throughout the formula. Instead, a substitution σ_2 of x by $y+1$ in ϕ to form $\sigma_2(\phi)$ is admissible for other symbols y, giving the formula $\sigma_2(\phi)$:

$$y+1 = z \rightarrow \langle z := z+1 \rangle (z \geq y+1+1).$$

□

More succinctly, we abbreviate the result of applying to ϕ the substitution σ that replaces variable y_i with term θ_i (for $1 \leq i \leq m$) by $\phi_{y_1}^{\theta_1} \ldots_{y_m}^{\theta_m}$. Thus $\phi_{y_1}^{\theta_1} \ldots_{y_m}^{\theta_m}$ is an abbreviation for $\sigma(\phi)$ defined according to Fig. 2.10. When no confusion arises, we also use implicit notation for substitutions to improve readability. Let $\phi(z)$ be a formula with a free variable z. Then for any term θ, we use $\phi(\theta)$ as an abbreviation for the formula $\phi(z)_z^\theta$ that results from $\phi(z)$ by substituting θ for z.

Example 2.13 (Admissible versus non-admissible substitutions). Consider the (valid) dL formula ϕ defined as

$$\phi \;\equiv\; x > 0 \land y > 1 \land z \geq x \;\rightarrow\; [z := z + xy] z > x.$$

Now the substitution that replaces x by $5a + x^2 - y$ is admissible for ϕ, giving the result $\phi_x^{5a+x^2-y}$:

$$5a+x^2-y>0 \land y>1 \land z \geq 5a+x^2-y \;\rightarrow\; [z:=z+(5a+x^2-y)y]\,z>5a+x^2-y.$$

This formula $\phi_x^{5a+x^2-y}$, which results by an admissible substitution from ϕ, is valid, just like ϕ.

However, the substitution that replaces x with the term az is not admissible for ϕ, because variable z occurs in the replacement az but is bound in ϕ, and could thus have a different value at its various occurrences. So we cannot apply this substitution to ϕ. Yet if we choose a fresh variable u and use bound variable renaming to rename all occurrences of bound variable z to u, we obtain the formula

$$\tilde{\phi} \;\equiv\; x>0 \land y>1 \land z \geq x \;\rightarrow\; [u:=z+xy]\,u>x.$$

This variant $\tilde{\phi}$ is equivalent to ϕ, because only bound variables have been renamed. After this bound variable renaming, the substitution replacing x by az becomes admissible and we obtain

$$\tilde{\phi}_x^{az} \;\equiv\; az>0 \land y>1 \land z \geq az \;\rightarrow\; [u:=z+(az)y]\,u>az.$$

This formula is valid (just like ϕ and $\tilde{\phi}$). But it is quite different from the formula we would obtain if we had just naïvely replaced every occurrence of x (admissible or not) by az, instead of using more careful admissible substitutions:

$$az>0 \land y>1 \land z \geq az \;\rightarrow\; [z:=z+(az)y]\,z>az.$$

The latter formula is clearly false for all I,η,ν with $val_{I,\eta}(\nu,a)=1$, because z cannot possibly be greater than az then. Contrast this with the validity of the original formula ϕ and its (admissible) substitution instance $\tilde{\phi}_x^{az}$.

Similarly, the substitution that replaces z with ax is not admissible for ϕ, because the replaced variable z is bound in ϕ, and could thus have a different value at its various occurrences. So we cannot apply this substitution. Yet if we again choose a fresh variable u and use bound variable renaming to rename all occurrences of bound variable z to u, we obtain the formula $\tilde{\phi}$ above. After this bound variable renaming, the substitution replacing z with ax becomes admissible and we obtain

$$\tilde{\phi}_z^{ax} \;\equiv\; x>0 \land y>1 \land ax \geq x \;\rightarrow\; [u:=ax+xy]\,u>x.$$

Again, this formula is valid and quite different from the formula we would obtain if we had just naïvely replaced every occurrence of z (admissible or not) by ax:

$$x>0 \land y>1 \land ax \geq x \;\rightarrow\; [z:=ax+xy]\,ax>x.$$

The latter formula is again false for all I,η,ν with $val_{I,\eta}(\nu,a)=1$, because x cannot possibly be greater than ax then. Contrast this with the validity of the original formula ϕ and the admissible substitution instance $\tilde{\phi}_x^{ax}$.

Thus, there is a close connection between the formula ϕ and its various substitution instances (if admissible!), which we will identify in the next lemma. As part of that, we will show that, since ϕ is valid, all of its (admissible) substitution instances

are valid. This close connection (and every other similarity) breaks when we naïvely replace terms in ϕ instead of obeying the requirements of admissible substitutions.

□

Example 2.14 (Non-admissibility in repetitions). The last example is prototypical for several dL formulas and works similarly for all dL formulas without repetitions or differential equations. Yet repetitions and differential equations themselves are more involved. Consider a (valid) dL formula with a repetition:

$$\psi \equiv x>0 \wedge y>1 \wedge z>x \rightarrow [(z:=z+xy)^*]z>x. \tag{2.8}$$

As with the formula ϕ from the last example, the substitution that replaces x by the term $5a+x^2-y$ is admissible for ψ, giving the result $\psi_x^{5a+x^2-y}$:

$$5a+x^2-y>0 \wedge y>1 \wedge z>5a+x^2-y \rightarrow [(z:=z+(5a+x^2-y)y)^*]z>5a+x^2-y$$

This formula $\psi_x^{5a+x^2-y}$, which results by an admissible substitution from ψ is valid, just like ψ.

Again, the substitution that replaces x by the term az is not admissible for ψ, because variable z occurs in the replacement az but is bound in ψ, and could have different values at its occurrences. Hence, we cannot apply this substitution. However, for repetitions, it is not so easy to do bound variable renaming to get around this! We cannot simply replace all bound occurrences of z by one fresh variable u, which would give

$$x>0 \wedge y>1 \wedge z>x \rightarrow [(u:=\check{u}+xy)^*]u>x.$$

But here the connection of u with the input z has been lost and the formula is no longer valid. The reason is that the occurrence of z on the right-hand side $z+xy$ of the jump (which corresponds to the occurrence of u we marked $\check{}$ in the last formula) is neither just free nor just bound. During the first iteration of the repetition, it would be free (because it receives its value from outside); during subsequent iterations, however, it would be bound (because it receives its value from the last assignment). The formula we would obtain if we had just naïvely replaced every occurrence of x (admissible or not) by az is also quite different and not valid:

$$x>0 \wedge y>1 \wedge z>x \rightarrow [(z:=z+(a\check{z})y)^*]z>az.$$

The reason is that the occurrence marked with $\check{}$ is neither just free nor just bound, because it depends on the number of iterations of the loop.

Likewise, the substitution that replaces z by ax is not admissible for ψ and cannot be applied, because the replaced variable z is bound in ψ. We thus cannot apply this substitution. Once more, it is not so easy to do bound variable renaming to get around this and we cannot just rename z to a fresh variable u to resolve this issue. The formula we would obtain if we had just naïvely replaced every occurrence of x (admissible or not) by az is also quite different and not valid:

$$x > 0 \land y > 1 \land ax > x \;\rightarrow\; [(z := \check{a}x + xy)^*]\, ax > x.$$

The reason is again that the occurrence of z (prior to replacing) at the position marked $\check{}$ is neither just free nor just bound. While it would be perfectly alright to replace the first dynamic occurrence of z (in the sequential execution order) by ax, subsequent occurrences (including those in repetitions) have a different operational value and cannot be replaced.

In these two cases, the substitutions are just not admissible for ψ and cannot be applied, because the modalities of ψ bind relevant replaced variables or variables in the replacements. Our proof calculus in Sect. 2.5.2 will use other ways that do not need substitution to prove formulas with repetitions like these. □

Example 2.15 (Non-admissibility in differential equations). The situation with differential equations is quite similar. In the $\mathsf{d}\mathscr{L}$ formula

$$\psi \;\equiv\; x > 0 \land y > 1 \land z > x \;\rightarrow\; [z' = z + xy]\, z > x \tag{2.9}$$

the occurrences of z in the differential equation are neither just free nor just bound: The value z affects the initial value z of the differential equation, but the value of z also evolves over time when following the differential equation to a new value. Thus, z is both a free initial value and bounded or updated during the evolution. The substitution that replaces x with $5a^2 + x^2 - y$ is still admissible for ψ, giving $\psi_x^{5a^2+x^2-y}$:

$$5a^2 + x^2 - y > 0 \land y > 1 \land z > 5a^2 + x^2 - y \rightarrow [z' = z + (5a^2 + x^2 - y)y]\, z > 5a^2 + x^2 - y$$

But we cannot substitute x with az, because the substitution is not admissible for ψ as bound variable z occurs in the replacement az. Nor can we substitute z with ax, because this substitution is not admissible for ψ either, as the replaced variable z is bound in ψ. The formula we would obtain if we had just naïvely replaced every occurrence of x (admissible or not) with az, is different and not valid:

$$az > 0 \land y > 1 \land z > x \;\rightarrow\; [z' = z + (\check{a}z)y]\, z > az.$$

The formula we would obtain, instead, if we had just naïvely replaced every occurrence of z (admissible or not) by ax, is also different and not valid:

$$x > 0 \land y > 1 \land ax > x \;\rightarrow\; [z' = \check{a}x + xy]\, ax > x.$$

In both cases, we marked the positions where the occurrences have been neither free nor bound with $\check{}$ once again.

In these two cases, the substitutions are not admissible for ψ and cannot be applied, because the modalities of ψ bind relevant replaced variables or variables in the replacements. Our proof calculus in Sect. 2.5.2 will prove such properties of differential equations differently. □

Example 2.16 (Bound variable renaming for repetitions and differential equations). On a side note, it would not be impossible to define bound variable renaming for

repetitions and differential equations. We decide not to use these extensions in our approach, because they are technically more involved and not necessary for our proof calculus. The purpose of this example is to show how bound variable renaming could be extended appropriately, nevertheless. When we add an extra discrete jump, we could define the following extended bound variable renaming variant of formula (2.8):

$$x > 0 \land y > 1 \land z > x \;\rightarrow\; [u := z;\; (u := u + xy)^*]\, u > x.$$

This formula separates the initial value assignment from the loop. Similarly when we add an extra discrete jump, we could define the following extended bound variable renaming variant of formula (2.9):

$$x > 0 \land y > 1 \land z > x \;\rightarrow\; [u := z;\; u' = u + xy]\, u > x.$$

Again, this formula separates the initial value assignment from the differential equation. For both variants, the substitution replacing x with az is admissible, and so is the substitution replacing z with ax. Essentially, the above two variants retain the initial value z explicitly before the repetition or differential equation. We have chosen not to use these extended bound variable renamings in this book and, instead, follow our choice that non-admissible substitutions are not applicable at all. □

There is a direct connection between a formula ϕ and its substitution instance $\sigma(\phi)$, provided that the substitution σ is admissible for ϕ. In fact, the valuation of ϕ and $\sigma(\phi)$ coincide if only we change the interpretation of the replaced symbols appropriately when evaluating ϕ. That is, semantically evaluating ϕ (after modifying the interpretation of the symbols replaced by σ in I, η, ν) is the same as semantically evaluating ϕ in the original I, η, ν after applying the substitution (resulting in $\sigma(\phi)$). Stated differently, we can show that, for admissible substitutions, syntactic substitution in the formula and semantic modification of I, η, ν have the same effect:

Lemma 2.2 (Substitution Lemma). *Let σ be an admissible substitution for the (term or) formula ϕ and let σ replace only logical variables; then*

$$\text{for each } I, \eta, \nu:\; val_{I,\eta}(\nu, \sigma(\phi)) = val_{I,\sigma^*(\eta)}(\nu, \phi),$$

where the semantic modification $\sigma^*(\eta)$ of assignment η is adjoint *to σ, i.e., $\sigma^*(\eta)$ is identical to η, except that $\sigma^*(\eta)(x) = val_{I,\eta}(\nu, \sigma(x))$ for all logical variables $x \in V$.*

Proof. In essence, the proof of this lemma is a simple corollary to the fact that both substitution and valuation are homomorphisms defined inductively on formulas from their effect on atomic symbols. The application of an admissible substitution σ is a homomorphic continuation of its effect on atomic symbols to all $\mathsf{d}\mathscr{L}$ formulas by way of Fig. 2.10. That is, the effect of an admissible(!) substitution on a compound formula is just defined by applying the substitution recursively to all

subformulas. Likewise, the valuation is a homomorphic continuation of the interpretation I, state ν, and assignment η on atomic symbols to all $\mathsf{d}\mathscr{L}$ formulas by way of Definition 2.6. That is, the valuation of a compound formula is just defined by using the valuation on all subformulas.

First we prove the substitution lemma applied to terms θ:

$$\text{for each } I,\eta,\nu:\ val_{I,\eta}(\nu,\sigma(\theta)) = val_{I,\sigma^*(\eta)}(\nu,\theta).$$

The proof is by induction on the structure of the term θ.

1. If θ is a logical variable $x \in V$, then, by definition of $\sigma^*(\eta)$:
$$val_{I,\eta}(\nu,\sigma(x)) = \sigma^*(\eta)(x) = val_{I,\sigma^*(\eta)}(\nu,x).$$
2. If θ is a state variable $x \in \Sigma$, then it is different from replaced logical variables $u \in V$ and $\sigma(x) = x$. Hence
$$val_{I,\eta}(\nu,\sigma(x)) = val_{I,\eta}(\nu,x) = \nu(x) = val_{I,\sigma^*(\eta)}(\nu,x).$$
3. If θ is of the form $f(\theta_1,\ldots,\theta_n)$ for a function symbol f of arity $n \geq 1$, then
$$\begin{aligned}
&val_{I,\eta}(\nu,\sigma(f(\theta_1,\ldots,\theta_n)))\\
&= val_{I,\eta}(\nu,f(\sigma(\theta_1),\ldots,\sigma(\theta_n)))\\
&= I(f)\big(val_{I,\eta}(\nu,\sigma(\theta_1)),\ldots,val_{I,\eta}(\nu,\sigma(\theta_n))\big)\\
&= I(f)\big(val_{I,\sigma^*(\eta)}(\nu,\theta_1),\ldots,val_{I,\sigma^*(\eta)}(\nu,\theta_n)\big)\\
&= val_{I,\sigma^*(\eta)}(\nu,f(\theta_1,\ldots,\theta_n))
\end{aligned}$$
because the θ_i are simpler than $f(\theta_1,\ldots,\theta_n)$ so that, by induction hypothesis, we have for each i:
$$val_{I,\eta}(\nu,\sigma(\theta_i)) = val_{I,\sigma^*(\eta)}(\nu,\theta_i).$$

Next we prove the substitution lemma applied to $\mathsf{d}\mathscr{L}$ formulas ϕ:

$$\text{for each } I,\eta,\nu:\ val_{I,\eta}(\nu,\sigma(\phi)) = val_{I,\sigma^*(\eta)}(\nu,\phi).$$

The proof is by induction on the structure of the formula ϕ.

1. If ϕ is of the form $p(\theta_1,\ldots,\theta_n)$ for a predicate symbol p of arity $n \geq 1$, then the proof is almost identical to that for function symbols above.
2. If ϕ is of the form $\phi_1 \vee \phi_2$, then we use the induction hypothesis on ϕ_1 and ϕ_2 to conclude
$$\begin{aligned}
&val_{I,\eta}(\nu,\sigma(\phi_1 \vee \phi_2))\\
&= val_{I,\eta}(\nu,\sigma(\phi_1) \vee \sigma(\phi_2)) = true\\
&\text{iff } val_{I,\eta}(\nu,\sigma(\phi_1)) = true \text{ or } val_{I,\eta}(\nu,\sigma(\phi_2)) = true
\end{aligned}$$

$$\begin{aligned}&\text{iff } val_{I,\sigma^*(\eta)}(\nu,\phi_1) = true \text{ or } val_{I,\sigma^*(\eta)}(\nu,\phi_2) = true\\ &\text{iff } val_{I,\sigma^*(\eta)}(\nu,\phi_1 \vee \phi_2) = true\end{aligned}$$

If ϕ is of the form $\phi_1 \wedge \phi_2$ or of the form $\phi_1 \rightarrow \phi_2$ or $\neg\phi_1$, then the proof is similar.

3. If ϕ is of the form $\exists x\,\psi$, then we use that σ was assumed to be admissible for ϕ. In particular (by bound variable renaming), x is not one of the replaced variables u and x does not occur in any of the replacements $\sigma(u)$. We use the induction hypothesis on ψ to conclude

$$\begin{aligned}&val_{I,\eta}(\nu,\sigma(\exists x\,\psi)) = val_{I,\eta}(\nu,\exists x\,\sigma(\psi)) = true\\ &\text{iff there is a } d \text{ such that } val_{I,\eta[x\mapsto d]}(\nu,\sigma(\psi)) = true\\ &\text{iff there is a } d \text{ such that } val_{I,\sigma^*(\eta[x\mapsto d])}(\nu,\psi) = true\\ &\text{iff there is a } d \text{ such that } val_{I,\sigma^*(\eta)[x\mapsto d]}(\nu,\psi) = true\\ &\text{iff } val_{I,\sigma^*(\eta)}(\nu,\exists x\,\psi) = true.\end{aligned}$$

Note that $\sigma^*(\eta[x\mapsto d]) = \sigma^*(\eta)[x\mapsto d]$, because x is not affected by the substitution σ (since admissible); hence x is not affected by adjoint assignments. If ϕ is of the form $\forall x\,\psi$, the proof is similar.

4. If ϕ is of the form $[\alpha]\psi$, then we use that the substitution σ is admissible by assumption. Hence, α does not bind any of the replaced variables nor any of the variables that occur in any of the replacements $\sigma(u)$. We use the induction hypothesis on ψ to conclude

$$\begin{aligned}&val_{I,\eta}(\nu,\sigma([\alpha]\psi)) = val_{I,\eta}(\nu,[\sigma(\alpha)]\sigma(\psi)) = true\\ &\text{iff for all } \omega \text{ with } (\nu,\omega)\in\rho_{I,\eta}(\sigma(\alpha)) : val_{I,\eta}(\omega,\sigma(\psi)) = true\\ &\text{iff for all } \omega \text{ with } (\nu,\omega)\in\rho_{I,\eta}(\sigma(\alpha)) : val_{I,\sigma^*(\eta)}(\omega,\psi) = true\\ &\text{iff}^{\star} \text{ for all } \omega \text{ with } (\nu,\omega)\in\rho_{I,\sigma^*(\eta)}(\alpha) : val_{I,\sigma^*(\eta)}(\omega,\psi) = true\\ &\text{iff } val_{I,\sigma^*(\eta)}(\nu,[\alpha]\psi) = true.\end{aligned}$$

For the middle step marked with $\star$, we still have to prove the substitution lemma for hybrid programs:

$$\rho_{I,\eta}(\sigma(\alpha)) = \rho_{I,\sigma^*(\eta)}(\alpha). \tag{2.10}$$

If α is of the form $\langle\alpha\rangle\psi$ then the proof is similar.

Finally we prove the substitution lemma for hybrid programs α as formulated in (2.10). The proof is by induction on the structure of hybrid program α.

1. If α is of the form $x_1 := \theta_1, \ldots x_n := \theta_n$, then we use the substitution lemma on the terms θ_i to show

$$
\begin{aligned}
&(\nu,\omega) \in \rho_{I,\eta}(\sigma(x_1 := \theta_1, \ldots x_n := \theta_n)) = \rho_{I,\eta}(x_1 := \sigma(\theta_1), \ldots x_n := \sigma(\theta_n)) \\
&\text{iff } \nu[x_1 \mapsto val_{I,\eta}(\nu, \sigma(\theta_1))] \ldots [x_n \mapsto val_{I,\eta}(\nu, \sigma(\theta_n))] = \omega \\
&\text{iff } \nu[x_1 \mapsto val_{I,\sigma^*(\eta)}(\nu, \theta_1)] \ldots [x_n \mapsto val_{I,\sigma^*(\eta)}(\nu, \theta_n)] = \omega \\
&\text{iff } (\nu,\omega) \in \rho_{I,\sigma^*(\eta)}(x_1 := \theta_1, \ldots x_n := \theta_n).
\end{aligned}
$$

2. If α is of the form $?\chi$ for a (first-order) $\mathsf{d}\mathscr{L}$ formula χ, then we use the substitution lemma on the (simpler and even first-order) $\mathsf{d}\mathscr{L}$ formula χ:

$$
\begin{aligned}
&(\nu,\omega) \in \rho_{I,\eta}(\sigma(?\chi)) = \rho_{I,\eta}(?\sigma(\chi)) \\
&\text{iff } \nu = \omega \text{ and } val_{I,\eta}(\nu, \sigma(\chi)) = \mathit{true} \\
&\text{iff } \nu = \omega \text{ and } val_{I,\sigma^*(\eta)}(\nu, \chi) = \mathit{true} \\
&\text{iff } (\nu,\omega) \in \rho_{I,\sigma^*(\eta)}(?\chi).
\end{aligned}
$$

3. If α is of the form $x_1' = \theta_1, \ldots x_n' = \theta_n \,\&\, \chi$, then we use the substitution lemma on terms and on the (first-order) $\mathsf{d}\mathscr{L}$ formula χ to conclude:

$$
\begin{aligned}
&(\nu,\omega) \in \rho_{I,\eta}(\sigma(x_1' = \theta_1, \ldots x_n' = \theta_n \,\&\, \chi)) \\
&\quad = \rho_{I,\eta}(x_1' = \sigma(\theta_1), \ldots x_n' = \sigma(\theta_n) \,\&\, \sigma(\chi)),
\end{aligned}
$$

which holds if and only if there is a continuous flow function $f : [0,r] \to \mathrm{Sta}(\Sigma)$ with $f(0) = \nu, f(r) = \omega$ and $val_{I,\eta}(f(\zeta), z) = val_{I,\eta}(\nu, z)$ for all $\zeta \in [0,r]$ and all $z \notin \{x_1, \ldots, x_n\}$ such that:

- for each x_i, $val_{I,\eta}(f(\zeta), x_i) = f(\zeta)(x_i)$ is continuous in ζ on $[0,r]$ and has a derivative of value $val_{I,\eta}(f(\zeta), \sigma(\theta_i))$ at each time $\zeta \in (0,r)$,
- and $val_{I,\eta}(f(\zeta), \sigma(\chi)) = \mathit{true}$ for each $\zeta \in [0,r]$.

By the substitution lemma for terms and formulas, respectively, these conditions are equivalent to

- for each x_i, $val_{I,\sigma^*(\eta)}(f(\zeta), x_i) = f(\zeta)(x_i)$ is continuous in ζ on $[0,r]$ and has a derivative of value $val_{I,\sigma^*(\eta)}(f(\zeta), \theta_i)$ at each time $\zeta \in (0,r)$,
- and $val_{I,\sigma^*(\eta)}(f(\zeta), \chi) = \mathit{true}$ for each $\zeta \in [0,r]$,

which hold if and only if

$$
(\nu,\omega) \in \rho_{I,\sigma^*(\eta)}(x_1' = \theta_1, \ldots x_n' = \theta_n \,\&\, \chi).
$$

4. If α is of the form $\beta \cup \gamma$, then we can use the induction hypothesis on β and γ to conclude

$$
\begin{aligned}
\rho_{I,\eta}(\sigma(\beta \cup \gamma)) = \rho_{I,\eta}(\sigma(\beta) \cup \sigma(\gamma)) &= \rho_{I,\eta}(\sigma(\beta)) \cup \rho_{I,\eta}(\sigma(\gamma)) \\
&= \rho_{I,\sigma^*(\eta)}(\beta) \cup \rho_{I,\sigma^*(\eta)}(\gamma) = \rho_{I,\sigma^*(\eta)}(\beta \cup \gamma).
\end{aligned}
$$

5. If α is of the form $\beta;\gamma$, then we use the induction hypothesis on β and on γ to conclude

$$
\begin{aligned}
&(\nu,\omega) \in \rho_{I,\eta}(\sigma(\beta;\gamma)) = \rho_{I,\eta}(\sigma(\beta);\sigma(\gamma)) \\
&\text{iff there is a } \mu \text{ with } (\nu,\mu) \in \rho_{I,\eta}(\sigma(\beta)) \text{ and } (\mu,\omega) \in \rho_{I,\eta}(\sigma(\gamma)) \\
&\text{iff there is a } \mu \text{ with } (\nu,\mu) \in \rho_{I,\sigma^*(\eta)}(\beta) \text{ and } (\mu,\omega) \in \rho_{I,\sigma^*(\eta)}(\gamma) \\
&\text{iff } (\nu,\omega) \in \rho_{I,\sigma^*(\eta)}(\beta;\gamma).
\end{aligned}
$$

6. The case where α is of the form β^* is similar, again using admissibility:

$$
\begin{aligned}
&(\nu,\omega) \in \rho_{I,\eta}(\sigma(\beta^*)) = \rho_{I,\eta}(\sigma(\beta)^*) \\
&\text{iff there are } n \in \mathbb{N}, \mu_0 = \nu, \mu_1, \ldots, \mu_n = \omega : (\mu_i,\mu_{i+1}) \in \rho_{I,\eta}(\sigma(\beta)) \\
&\text{iff there are } n \in \mathbb{N}, \mu_0 = \nu, \mu_1, \ldots, \mu_n = \omega : (\mu_i,\mu_{i+1}) \in \rho_{I,\sigma^*(\eta)}(\beta) \\
&\text{iff } (\nu,\omega) \in \rho_{I,\sigma^*(\eta)}(\beta^*).
\end{aligned}
$$

□

The substitution lemma implies a simple corollary for substituting program variables instead of (or in addition to) logical variables. The proof is an immediate consequence of a double application of the substitution lemma, so that, in the remainder of this book, we do not distinguish between Lemma 2.2 and the following corollary.

Corollary 2.1. *Let σ be an admissible substitution for the (term or) formula ϕ; then*

$$\text{for each } I,\eta,\nu: \ val_{I,\eta}(\nu,\sigma(\phi)) = val_{I,\sigma^*(\eta)}(\sigma^*(\nu),\phi),$$

where the semantic modification $\sigma^*(\nu)$ of state ν is adjoint *to σ. The adjoint $\sigma^*(\nu)$ is identical to ν, except that $\sigma^*(\nu)(x) = val_{I,\eta}(\nu,\sigma(x))$ for all state variables $x \in \Sigma$. The adjoint $\sigma^*(\eta)$ is defined as in Lemma 2.2.*

Proof. The proof is a simple corollary to Lemma 2.2, using fresh logical variables z_i to relate $\sigma(\phi)$ with ϕ for gluing two uses of Lemma 2.2 together. To simplify notation, assume that σ only replaces a single state variable x by θ and let us denote the result of applying this substitution to ϕ by ϕ_x^θ. Let z be a fresh logical variable. Since the substitution σ is admissible for ϕ, the replaced variable x and all variables in its replacement θ are not bound in ϕ. Thus, ϕ is of the form ψ_z^x for the formula ψ, which is like ϕ except that it has z in place of x everywhere. Now abbreviate $val_{I,\eta}(\nu,\theta)$ as e, and abbreviate $val_{I,\eta}(\nu[x\mapsto e],x)$ as d. Then, we use Lemma 2.2 at the positions indicated $\star$ to conclude:

$$
\begin{aligned}
val_{I,\eta}(\nu,\phi_x^\theta) &= val_{I,\eta}(\nu,\psi_{z\,x}^{x\theta}) = val_{I,\eta}(\nu,\psi_z^\theta) \overset{\star}{=} val_{I,\eta[z\mapsto e]}(\nu,\psi) \\
&= val_{I,\eta[z\mapsto d]}(\nu[x\mapsto e],\psi) \overset{\star}{=} val_{I,\eta}(\nu[x\mapsto e],\psi_z^x) = val_{I,\eta}(\nu[x\mapsto e],\phi).
\end{aligned}
$$

Note that the two lines are equal because the value of state variable x in the state does not matter for ψ, where x does not occur, and because $d = val_{I,\eta}(\nu[x\mapsto e],x) = e$.

□

Example 2.17. Again consider the formula ϕ, and an instance $\phi_x^{5a^2+b}$ under an admissible substitution:

$$\phi \equiv x = z \to \langle z := z+1 \rangle (z \geq x+1),$$
$$\phi_x^{5a^2+b} \equiv 5a^2+b = z \to \langle z := z+1 \rangle (z \geq 5a^2+b+1).$$

Using the substitution lemma, we can conclude that with respect to any I, η, ν, the formula ϕ and its instance $\phi_x^{5a^2+b}$ evaluate to the same truth-value when adapting the value of x appropriately. That is, let σ be the substitution that replaces x with $5a^2+b$, i.e., $\sigma(\phi) \equiv \phi_x^{5a^2+b}$; then (the corollary to) Lemma 2.2 implies:

$$val_{I,\eta}(\nu, \phi_x^{5a^2+b}) = val_{I,\sigma^*(\eta)}(\sigma^*(\nu), \phi).$$

Let us abbreviate the value $val_{I,\eta}(\nu, 5a^2+b)$ of the replacement $5a^2+b$ of x by e. Then if $x \in V$ is a logical variable, then $\sigma^*(\nu) = \nu$ and $\sigma^*(\eta) = \eta[x \mapsto e]$; hence

$$val_{I,\eta}(\nu, \phi_x^{5a^2+b}) = val_{I,\eta[x \mapsto e]}(\nu, \phi).$$

If, instead, $x \in \Sigma$ is a state variable, then $\sigma^*(\eta) = \eta$ and $\sigma^*(\nu) = \nu[x \mapsto e]$; hence

$$val_{I,\eta}(\nu, \phi_x^{5a^2+b}) = val_{I,\eta}(\nu[x \mapsto e], \phi).$$

In either case (either $x \in V$ or $x \in \Sigma$), if the value of x and its replacement $5a^2+b$ agree in the original I, η, ν already, i.e., if $val_{I,\eta}(\nu, x) = val_{I,\eta}(\nu, 5a^2+b)$, then their valuations agree according to the substitution lemma:

$$val_{I,\eta}(\nu, \phi_x^{5a^2+b}) = val_{I,\eta}(\nu, \phi).$$

□

The substitution lemma is a very powerful tool, because, among other things, we can use it to replace equals for equals without changing the valuation (substitution property). If we know that x and θ have the same value in I, η, ν, then we can substitute θ for x in a formula ϕ (if admissible) without changing the truth-value of ϕ, that is:

Lemma 2.3 (Substitution property). *If $I, \eta, \nu \models x = \theta$, then $I, \eta, \nu \models \phi \leftrightarrow \phi_x^\theta$ for any formula ϕ for which the substitution replacing x with θ is admissible.*

Proof. Consider any I, η, ν with $I, \eta, \nu \models x = \theta$. First, note that this assumption is equivalent to $val_{I,\eta}(\nu, x) = val_{I,\eta}(\nu, \theta)$. We have to show $I, \eta, \nu \models \phi \leftrightarrow \phi_x^\theta$, or, equivalently, $val_{I,\eta}(\nu, \phi) = val_{I,\eta}(\nu, \phi_x^\theta)$. This follows from the Substitution Lemma 2.2 when we choose σ to be the substitution that replaces x by θ since

$$val_{I,\eta}(\nu, \phi_x^\theta) = val_{I,\sigma^*(\eta)}(\sigma^*(\nu), \phi) = val_{I,\eta}(\nu, \phi).$$

The last step follows from the fact that I, η, ν equals $I, \sigma^*(\eta), \sigma^*(\nu)$, respectively, because the substitution σ only replaces x by θ, which already have the same value to begin with, as we assumed $val_{I,\eta}(\nu, x) = val_{I,\eta}(\nu, \theta)$. □

In addition, whenever a formula ϕ is valid (ϕ is true in all I, η, ν), the substitution lemma implies that all of its (admissible) substitution instances $\sigma(\phi)$ are valid too for any substitution σ that is admissible for ϕ.

Lemma 2.4 (Substitutions preserve validity). *If $\models \phi$, i.e., ϕ is valid, then $\models \sigma(\phi)$ for any substitution σ that is admissible for ϕ.*

Proof. Let ϕ be valid, i.e., $I, \eta, \nu \models \phi$ for all I, η, ν. Consider any I, η, ν and any substitution σ that is admissible for ϕ. Now the Substitution Lemma 2.2 implies

$$val_{I,\eta}(\nu, \sigma(\phi)) = val_{I,\sigma^*(\eta)}(\sigma^*(\nu), \phi) = true.$$

The last step holds because ϕ is valid and, in particular, holds for $I, \sigma^*(\eta), \sigma^*(\nu)$. □

Observe that, for soundness, the notion of bound variables in Definition 2.8 could in fact be *any* overapproximation of the set of variables that possibly change their value during a hybrid program. In vacuous identity changes like $x := x$ or $x' = 0$, variable x will not really change its value, but we still consider x as a bound variable for simplicity. For a hybrid program α, we denote by $\forall^\alpha \phi$ the *universal closure* of formula ϕ with respect to all state variables bound in α. Quantification over state variable x is definable as $\forall X\, [x := X]\Phi$ using an auxiliary logical variable X.

2.5.2 Rules of the Calculus for Differential Dynamic Logic

We present a proof calculus for $\mathsf{d}\mathscr{L}$ as a Gentzen-style sequent calculus [133]. Sequents are essentially a standard form for logical formulas that is convenient for proving. A *sequent* is of the form $\Gamma \vdash \Delta$, where the *antecedent* Γ and *succedent* Δ are finite sets of formulas. The semantics of $\Gamma \vdash \Delta$ is that of the formula $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$. For quantifier elimination rules, we make use of this fact by considering sequent $\Gamma \vdash \Delta$ as an abbreviation for the latter formula. The antecedent Γ can be thought of as the formulas we assume to be true, whereas the succedent Δ can be understood as formulas for which we want to show that at least one of them is true assuming all formulas of Γ are true. So for proving a sequent $\Gamma \vdash \Delta$, we assume all Γ and want to show that one of the Δ is true. For some simple sequents like $\Gamma, \phi \vdash \phi, \Delta$, we directly know that they are valid, because we can certainly show ϕ if we assume ϕ (in fact, we will use this as an axiom). For other sequents, it is more difficult to see whether they are valid (true under all circumstances) and it is the purpose of a proof calculus to provide a means to find out.

For handling quantifiers in the $\mathsf{d}\mathscr{L}$ calculus, we cannot use the standard proof rules [147, 122, 123], because these are for uninterpreted first-order logic and (ultimately) work by instantiating quantifiers, either eagerly as in ground tableaux or

lazily by unification as in free-variable tableaux [147, 122, 123]. Also, see App. A for an exposition of proving in uninterpreted first-order logic. The basis of $\mathsf{d}\mathscr{L}$, in contrast, is first-order logic interpreted over the reals or in the theory of real-closed fields [287, 288]. A formula like $\exists a \forall x (x^2 + a > 0)$ cannot be proven by instantiation-based quantifier rules but is valid in the theory of real-closed fields. Unfortunately, quantifier elimination (QE) over the reals [81, 288], which is the standard decision procedure for real arithmetic, cannot be applied to formulas with modalities either. Hence, we introduce new quantifier rules that integrate quantifier elimination in a way that is compatible with dynamic modalities (as we illustrate in Sect. 2.5.3).

Definition 2.9 (Quantifier elimination). A first-order theory admits *quantifier elimination* if, with each formula ϕ, a quantifier-free formula $\mathrm{QE}(\phi)$ can be associated effectively that is equivalent (i.e., $\phi \leftrightarrow \mathrm{QE}(\phi)$ is valid) and has no additional free variables or function symbols. The operation QE is further assumed to evaluate ground formulas (i.e., without variables), yielding a decision procedure for closed formulas of this theory (i.e., formulas without free variables).

Example 2.18. Quantifier elimination uses the special structure of real arithmetic to express quantified arithmetic formulas equivalently without quantifiers and without using more free variables. For instance, QE yields the following equivalence:

$$\mathrm{QE}(\exists x (ax^2 + bx + c = 0)) \;\equiv\; (a \neq 0 \land b^2 - 4ac \geq 0) \lor (a = 0 \land (b = 0 \rightarrow c = 0)).$$

In this particular case, the equivalence can be found by using the generic condition for solvability of quadratic equations over the reals plus special cases when coefficients are zero. For details on quantifier elimination in real-closed fields and an overview of decision procedures for real arithmetic, also see App. D.2. □

As usual in sequent calculus rules—although the direction of entailment in the proof rules is from *premises* (above rule bar) to *conclusion* (below)—the order of reasoning is *goal-directed*: Rules are applied backwards, i.e., starting from the desired conclusion at the bottom (*goal*) to the resulting premises (*subgoals*). To highlight the logical essence of the $\mathsf{d}\mathscr{L}$ calculus, Fig. 2.11 provides *rule schemata* with which the following definition associates the calculus rules that are applicable in $\mathsf{d}\mathscr{L}$ proofs. The calculus consists of propositional rules (¬r–*cut*), first-order quantifier rules (∀r–i∃), rules for dynamic modalities ($\langle;\rangle$–[′]), and global rules ([]*gen*–*con*). All substitutions in the rules in Fig. 2.11 need to be admissible for the rules to be applicable, including the substitution that inserts $s(X_1,..,X_n)$ into $\phi(s(X_1,..,X_n))$. Proof schemata come in three kinds with which the following definition associates proof rules: 1) sequent proof schemata that mention the sequent symbol $\vdash$, 2) symmetric proof schemata that do not mention the sequent symbol $\vdash$ and can be applied on either side of the sequent, 3) the special proof schema i∃ that merges multiple branches.

Definition 2.10 (Rules). The *rule schemata* in Fig. 2.11—in which *all* substitutions need to be admissible for the rules to be applicable, including the substitution that inserts $s(X_1,..,X_n)$ into $\phi(s(X_1,..,X_n))$—induce *calculus rules* by:

1. If
$$\frac{\Phi_1 \vdash \Psi_1 \quad \ldots \quad \Phi_n \vdash \Psi_n}{\Phi_0 \vdash \Psi_0} \tag{2.11}$$
is an instance of a rule schema in Fig. 2.11 (rules ∀r–∀l, i∀, and the propositional and global rule schemata have this form), then
$$\frac{\Gamma, \langle \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{J} \rangle \Psi_1, \Delta \quad \ldots \quad \Gamma, \langle \mathscr{J} \rangle \Phi_n \vdash \langle \mathscr{J} \rangle \Psi_n, \Delta}{\Gamma, \langle \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{J} \rangle \Psi_0, \Delta}$$
can be applied as a proof rule of the $\mathsf{d}\mathscr{L}$ calculus, where Γ, Δ are arbitrary finite sets of additional context formulas (including empty sets) and $\mathscr{J}$ is a discrete jump set (including the empty set). Hence, the rule context Γ, Δ and prefix $\langle \mathscr{J} \rangle$ remain unchanged during rule applications; only the formulas mentioned in (2.11) are affected.
2. Symmetric schemata can be applied on either side of the sequent: If
$$\frac{\phi_1}{\phi_0}$$
is an instance of one of the symmetric rule schemata (the dynamic rules) in Fig. 2.11, then
$$\frac{\Gamma \vdash \langle \mathscr{J} \rangle \phi_1, \Delta}{\Gamma \vdash \langle \mathscr{J} \rangle \phi_0, \Delta} \quad \text{and} \quad \frac{\Gamma, \langle \mathscr{J} \rangle \phi_1 \vdash \Delta}{\Gamma, \langle \mathscr{J} \rangle \phi_0 \vdash \Delta}$$
can both be applied as proof rules of the $\mathsf{d}\mathscr{L}$ calculus, where Γ, Δ are arbitrary finite sets of context formulas (including the empty set) and $\mathscr{J}$ is a discrete jump set (including empty sets). In particular, symmetric schemata yield equivalence transformations, because the same rule applies in the antecedent as in the succedent.
3. Schema i∃ applies to *all* goals containing X at once: If $\Phi_1 \vdash \Psi_1, \ldots, \Phi_n \vdash \Psi_n$ is the list of all open goals of the proof that contain free variable X, then an instance
$$\frac{\vdash \mathrm{QE}(\exists X \bigwedge_i (\Phi_i \vdash \Psi_i))}{\Phi_1 \vdash \Psi_1 \quad \ldots \quad \Phi_n \vdash \Psi_n}$$
of rule schema i∃ can be applied as a proof rule of the $\mathsf{d}\mathscr{L}$ calculus.

Propositional Rules

For propositional logic, standard propositional rules ¬r–*cut* with the cut rule are listed in the first block of Fig. 2.11. They decompose the propositional structure of formulas. Rules ¬r and ¬l use simple dualities caused by the implicative semantics of sequents. Essentially, instead of showing $\neg\phi$ in the succedent, we assume the contrary ϕ in the antecedent with rule ¬r. In rule ¬l, instead of assuming $\neg\phi$ in

$$(\neg\mathrm{r})\ \frac{\phi \vdash}{\vdash \neg\phi} \qquad (\vee\mathrm{r})\ \frac{\vdash \phi, \psi}{\vdash \phi \vee \psi} \qquad (\wedge\mathrm{r})\ \frac{\vdash \phi \quad \vdash \psi}{\vdash \phi \wedge \psi} \qquad (\rightarrow\mathrm{r})\ \frac{\phi \vdash \psi}{\vdash \phi \rightarrow \psi} \qquad (ax)\ \frac{}{\phi \vdash \phi}$$

$$(\neg\mathrm{l})\ \frac{\vdash \phi}{\neg\phi \vdash} \qquad (\vee\mathrm{l})\ \frac{\phi \vdash \quad \psi \vdash}{\phi \vee \psi \vdash} \qquad (\wedge\mathrm{l})\ \frac{\phi, \psi \vdash}{\phi \wedge \psi \vdash} \qquad (\rightarrow\mathrm{l})\ \frac{\vdash \phi \quad \psi \vdash}{\phi \rightarrow \psi \vdash} \qquad (cut)\ \frac{\vdash \phi \quad \phi \vdash}{\vdash}$$

$$(\langle;\rangle)\ \frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \qquad (\langle^{*n}\rangle)\ \frac{\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}{\langle\alpha^*\rangle\phi} \qquad (\langle:=\rangle)\ \frac{\phi^{\theta_1 \ldots \theta_n}_{x_1 \ldots x_n}}{\langle x_1 := \theta_1, .., x_n := \theta_n\rangle\phi}$$

$$([;])\ \frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi} \qquad ([^{*n}])\ \frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi} \qquad ([:=])\ \frac{\langle x_1 := \theta_1, .., x_n := \theta_n\rangle\phi}{[x_1 := \theta_1, .., x_n := \theta_n]\phi}$$

$$(\langle\cup\rangle)\ \frac{\langle\alpha\rangle\phi \vee \langle\beta\rangle\phi}{\langle\alpha \cup \beta\rangle\phi} \qquad (\langle?\rangle)\ \frac{\chi \wedge \psi}{\langle?\chi\rangle\psi} \qquad (\langle'\rangle)\ \frac{\exists t{\geq}0\,\big((\forall 0{\leq}\tilde{t}{\leq}t\ \langle\mathscr{S}_{\tilde{t}}\rangle\chi) \wedge \langle\mathscr{S}_t\rangle\phi\big)}{\langle x_1' = \theta_1, .., x_n' = \theta_n \,\&\, \chi\rangle\phi}\ 1$$

$$([\cup])\ \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} \qquad ([?])\ \frac{\chi \rightarrow \psi}{[?\chi]\psi} \qquad (['])\ \frac{\forall t{\geq}0\,\big((\forall 0{\leq}\tilde{t}{\leq}t\ \langle\mathscr{S}_{\tilde{t}}\rangle\chi) \rightarrow \langle\mathscr{S}_t\rangle\phi\big)}{[x_1' = \theta_1, .., x_n' = \theta_n \,\&\, \chi]\phi}\ 1$$

$$(\forall\mathrm{r})\ \frac{\vdash \phi(s(X_1, .., X_n))}{\vdash \forall x\,\phi(x)}\ 2 \qquad (\exists\mathrm{r})\ \frac{\vdash \phi(X)}{\vdash \exists x\,\phi(x)}\ 4$$

$$(\exists\mathrm{l})\ \frac{\phi(s(X_1, .., X_n)) \vdash}{\exists x\,\phi(x) \vdash}\ 2 \qquad (\forall\mathrm{l})\ \frac{\phi(X) \vdash}{\forall x\,\phi(x) \vdash}\ 4$$

$$(\mathrm{i}\forall)\ \frac{\vdash \mathrm{QE}(\forall X\,(\Phi(X) \vdash \Psi(X)))}{\Phi(s(X_1, .., X_n)) \vdash \Psi(s(X_1, .., X_n))}\ 3 \qquad (\mathrm{i}\exists)\ \frac{\vdash \mathrm{QE}(\exists X\ \bigwedge_i(\Phi_i \vdash \Psi_i))}{\Phi_1 \vdash \Psi_1 \quad \ldots \quad \Phi_n \vdash \Psi_n}\ 5$$

$$([]gen)\ \frac{\vdash \forall^{\alpha}(\phi \rightarrow \psi)}{[\alpha]\phi \vdash [\alpha]\psi} \qquad (\langle\rangle gen)\ \frac{\vdash \forall^{\alpha}(\phi \rightarrow \psi)}{\langle\alpha\rangle\phi \vdash \langle\alpha\rangle\psi}$$

$$(ind)\ \frac{\vdash \forall^{\alpha}(\phi \rightarrow [\alpha]\phi)}{\phi \vdash [\alpha^*]\phi} \qquad (con)\ \frac{\vdash \forall^{\alpha}\forall v{>}0\,(\varphi(v) \rightarrow \langle\alpha\rangle\varphi(v-1))}{\exists v\,\varphi(v) \vdash \langle\alpha^*\rangle\exists v{\leq}0\,\varphi(v)}\ 6$$

---

[1] t and $\tilde{t}$ are fresh logical variables and $\langle\mathscr{S}_t\rangle$ is the jump set $\langle x_1 := y_1(t), .., x_n := y_n(t)\rangle$ with simultaneous solutions $y_1, .., y_n$ of the respective differential equations with constant symbols x_i as symbolic initial values.

[2] s is a new (Skolem) function symbol and $X_1, .., X_n$ are all free logical variables of $\forall x\,\phi(x)$.

[3] X is a new logical variable. Further, QE needs to be defined for the formula in the premise.

[4] X is a new logical variable.

[5] Among all open branches, free logical variable X only occurs in the branches $\Phi_i \vdash \Psi_i$. Further, QE needs to be defined for the formula in the premise, especially, no Skolem dependencies on X can occur.

[6] Logical variable v does not occur in α.

Fig. 2.11 Rule schemata of the free-variable proof calculus for differential dynamic logic

the antecedent, we show the contrary ϕ in the succedent. Rule $\vee$r uses the fact that formulas are combined disjunctively in succedents, rule $\wedge$l that they are conjunctive in antecedents. The comma between formulas in an antecedent has the same effect as a conjunction, and the comma between formulas in the succedent has the same effect as a disjunction. Rules $\vee$l and $\wedge$r split the proof into two cases, because conjuncts in the succedent can be proven separately ($\wedge$r) and, dually, disjuncts of the antecedent can be assumed separately ($\vee$l). For $\wedge$r we want to show conjunction $\phi \wedge \psi$, so in the left branch we proceed to show $\Gamma \vdash \phi, \Delta$ and, in addition, in the right branch we

show $\Gamma \vdash \psi, \Delta$, which, together, entail $\Gamma \vdash \phi \wedge \psi, \Delta$. If, as in rule $\vee$l, we assume disjunction $\phi \vee \psi$ as part of the antecedent, then we do not know if we can assume ϕ to hold or if we can assume ψ to hold in the antecedent, but know only that one of them holds. Hence, as in a case distinction, $\vee$l considers both cases, the case where we assume ϕ in the antecedent, and the case where we assume ψ. If both subgoals can be proven, this entails $\Gamma, \phi \vee \psi \vdash \Delta$. Rules $\rightarrow$r and $\rightarrow$l can be derived from the equivalence of $\phi \rightarrow \psi$ and $\neg\phi \vee \psi$. Rule $\rightarrow$r uses the fact that implication $\rightarrow$ has the same meaning as the sequent arrow $\vdash$ of a sequent. Intuitively, to show implication $\phi \rightarrow \psi$, rule $\rightarrow$r assumes ϕ (in the antecedent) and shows ψ (in the succedent). Rule $\rightarrow$l assumes an implication $\phi \rightarrow \psi$ to hold in the antecedent, but we do not know if this implication holds because ϕ is false, or because ψ is true, so $\rightarrow$l splits into those two branches.

The axiom rule *ax* closes a goal (there are no further subgoals, which we sometimes mark $*$ explicitly), because assumption ϕ in the antecedent trivially entails ϕ in the succedent (sequent $\Gamma, \phi \vdash \phi, \Delta$ is a simple syntactic tautology). Rule *cut* is the *cut* rule that can be used for case distinctions: The right subgoal assumes any additional formula ϕ in the antecedent that the left subgoal shows in the succedent. Dually: regardless of whether ϕ is actually true or false, both cases are covered by proof branches. We only use cuts in an orderly fashion to derive simple rule dualities and to simplify meta-proofs. In practical applications, cuts are not usually needed and we conjecture that this is no coincidence.

According to the definition in Definition 2.10, all propositional rules can be applied with an additional context Γ, Δ. In particular, rules *ax* and *cut* can also be applied as:

$$ax\frac{}{\Gamma, \phi \vdash \phi, \Delta} \quad \text{and} \quad cut\frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta}$$

First-Order Quantifier Rules

The quantifier rules $\forall$r,$\exists$l,$\exists$r,$\forall$l,i$\forall$,i$\exists$ constitute a purely modular interface to arithmetic mathematical reasoning. They can use any theory that admits quantifier elimination and has a decidable ground theory (formulas without quantifiers or variables), including the theory of real arithmetic or real-closed fields [288, 81]. Rules $\forall$r,$\exists$l,$\exists$r,$\forall$l handle quantifiers and replace quantified variables by Skolem function terms ($\forall$r,$\exists$l) or free logical variables ($\exists$r,$\forall$l), respectively. Later in the proof, rules i$\forall$,i$\exists$ can reintroduce quantifiers for these previously quantified symbols and apply quantifier elimination in real-closed fields once the remaining formulas are first-order in the relevant symbols.

Rule $\exists$l, with which we want to show $\exists x\, \phi(x)$ in the succedent, introduces a new free logical variable X for an existentially quantified variable x. Essentially, free variable X can be thought of as a variable for which an appropriate value still needs to be found for the proof to close. This makes sense, because at the time of applying proof rule $\exists$l, it is mostly impossible to know which particular instance to choose for X that will help. But once we find such an X that proves the subgoal $\Gamma \vdash \phi(X), \Delta$

later, we have also proven the goal $\Gamma \vdash \exists x\, \phi(x), \Delta$, because X will be a witness for the existence. We have also proven the goal if, later during the proof, we prove the existence of an X satisfying the constraints indirectly, without directly instantiating a witness. This is what rule i$\exists$ is for.

The dual rule $\forall$l, which assumes $\forall x\, \phi(x)$ in the antecedent, introduces a new free logical variable X for the universally quantified variable x in the antecedent. If, later, we have found an instance of X that proves subgoal $\Gamma, \phi(X) \vdash \Delta$, then we have also proven goal $\Gamma, \forall x\, \phi(x) \vdash \Delta$, because if we can prove the subgoal just from assuming the particular $\phi(X)$ in the antecedent, then the goal also holds where we even assume $\phi(x)$ holds for all x. While this reasoning is perfectly good if it works, it is somewhat surprising why this should always work for all cases. Why should one instance be enough? Why should it not be necessary to assume two different instances $\phi(X)$ and $\phi(Y)$ during the proof? The fact that this is not necessary comes from proof rule i$\exists$, which can reintroduce quantifiers and eliminate them equivalently.

Rule $\forall$r, with which we want to show $\forall x\, \phi(x)$ in the succedent, introduces a new (Skolem) function symbol s for the previously quantified variable x and replaces x by a (Skolem) term $s(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are all the free logical variables of the original formula $\forall x\, \phi(x)$. This works like a proof in mathematics, where we want so show $\forall x\, \phi(x)$ in the succedent and do so by choosing a fresh symbol s for which we prove that $\phi(s(X_1, \ldots, X_n))$ holds. Because s was arbitrary and we did not assume anything special about the value of s, this implies that $\forall x\, \phi(x)$ holds. The free variables $X_1, \ldots, X_n$ of the Skolem terms keep track of the dependencies of the symbols for nested quantifiers. Having all free logical variables $X_1, \ldots, X_n$ in the Skolem term is important for soundness in order to prevent unsound rearrangements of quantifiers, as we elaborate in Sect. 2.5.3.

The dual rule $\exists$l is similar. When we assume $\exists x\, \phi(x)$ in the antecedent, then we only know that such an x exists, not what value it has. Hence, $\exists$l introduces a new name for this object in the form of a new (Skolem) function symbol s and replaces x by a (Skolem) term $s(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are all the free logical variables of the original formula $\exists x\, \phi(x)$. If we can prove the subgoal, the subgoal entails the goal, because we did not assume anything special about s. Having all free logical variables $X_1, \ldots, X_n$ in the Skolem term to track the dependencies of the symbols is again important for soundness to prevent unsound rearrangements of quantifiers. Intuitively, for a formula like $\forall x \exists y\, \phi(x, y)$ in the antecedent—which will yield $\exists y\, \phi(X, y)$ after applying $\forall$l—we need to track the dependency of y on X, which yields $\phi(s(X), X)$ when applying $\exists$l. We need to remember that the choice for s may depend on X, because the choice of y may depend on x.

With the rule i$\forall$, we can reintroduce a universal quantifier for a Skolem term $s(X_1, \ldots, X_n)$, which corresponds to a previously universally quantified variable in the succedent or a previously existentially quantified variable in the antecedent. The point of reintroducing the quantifier is that this makes sense when the remaining formulas are first-order in the quantified variable so that they can be handled equivalently by quantifier elimination in real-closed fields. When we have proven the subgoal (with for all X) then this entails the goal for the particular $s(X_1, \ldots, _n)$. In particular, when we remove a quantifier with $\forall$r,$\exists$l to obtain a Skolem term, we can

continue with other proof rules to handle the dynamic modalities and then reintroduce the quantifier for the Skolem term with i∀ once quantifier elimination for real arithmetic becomes applicable.

The dual rule i∃ can reintroduce an existential quantifier for a free logical variable that was previously existentially quantified in the succedent or previously universally quantified in the antecedent. Again, this makes sense when the resulting formula in the premise is first-order in the quantified variable X so that quantifier elimination can eliminate the quantifier equivalently. When we remove a quantifier with ∃r,∀l to obtain a free logical variable, we can continue using other proof rules to handle the dynamic modalities and then reintroduce the quantifier for the free logical variable with i∃ once quantifier elimination is applicable.

The quantifier rules ∀r and ∃l correspond to the liberalised δ^+-rule of Hähnle and Schmitt [147]. Rules ∃r and ∀l resemble the usual γ-rule but, unlike in [122, 123, 147, 134], they cannot be applied twice because the original formula is removed ($\exists x\, \phi(x)$ in ∃r). The calculus still has a complete handling of quantifiers due to i∀ and i∃, which can reconstruct and eliminate quantifiers once QE is applicable as the remaining constraints are first-order in the respective variables. In the premise of i∀ and i∃, we again consider sequents $\Phi \vdash \Psi$ as abbreviations for formulas. For closed formulas, we do not need other arithmetic rules. We defer illustrations and further discussion of quantifier rules to Sect. 2.5.3. For comparison, App. A gives a summary of the standard γ-rules and δ^+-rules that are used for handling quantifiers in uninterpreted first-order logic. In Sect. 3.5.5, we show an alternative way of handling real arithmetic in a modular way using deduction modulo by side deductions.

Dynamic Rules

The dynamic modality rules transform a hybrid program into structurally simpler logical formulas by symbolic decomposition. Rules $\langle;\rangle$,$[;]$,$\langle\cup\rangle$,$[\cup]$,$\langle^{*n}\rangle$,$[^{*n}]$,$\langle?\rangle$,$[?]$ are as in discrete dynamic logic [149, 37]. Also, see Fig. 2.12 for an illustration of the correspondence of a representative set of proof rules for dynamic modalities to the transition semantics of hybrid programs (from Definition 2.7).

Nondeterministic choices split into their alternatives ($\langle\cup\rangle$,$[\cup]$). For rule $[\cup]$: If all α transitions lead to states satisfying ϕ (i.e., $[\alpha]\phi$ holds) and all β transitions lead to states satisfying ϕ (i.e., $[\beta]\phi$ holds), then, all transitions of program $\alpha\cup\beta$ that choose between following α and following β also lead to states satisfying ϕ (i.e., $[\alpha\cup\beta]\phi$ holds). Dually for rule $\langle\cup\rangle$, if there is an α transition to a ϕ state ($\langle\alpha\rangle\phi$) or a β-transition to a ϕ state ($\langle\beta\rangle\phi$), then, in either case, there is a transition of $\alpha\cup\beta$ to ϕ ($\langle\alpha\cup\beta\rangle\phi$ holds), because $\alpha\cup\beta$ can choose which of those transitions to follow. A general principle behind the dL proof rules that is most noticeable in $\langle\cup\rangle$,$[\cup]$ is that these proof rules symbolically decompose the reasoning into two separate parts and analyse the fragments α and β separately, which is good for scalability. For these symbolic structural decompositions, it is very helpful that dL is a full logic that is closed under all logical operators, including disjunction and conjunction, for then the premises in $[\cup]$,$\langle\cup\rangle$ are dL formulas again (unlike in Hoare logic [161]).

$$([:=])\ \frac{\phi_x^\theta}{[x:=\theta]\phi}$$

$$(\langle' \rangle)\ \frac{\exists t{\geq}0\,\langle x:=y_x(t)\rangle\phi}{\langle x'=\theta\rangle\phi}$$

$$(\langle' \rangle)\ \frac{\exists t{\geq}0\,(\hat{\chi}\wedge\langle x:=y_x(t)\rangle\phi)}{\langle x'=\theta\,\&\,\chi\rangle\phi}\qquad \hat{\chi}\equiv\forall 0{\leq}s{\leq}t\,\langle x:=y_x(s)\rangle\chi$$

$$([\cup])\ \frac{[\alpha]\phi\wedge[\beta]\phi}{[\alpha\cup\beta]\phi}$$

$$([;])\ \frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi}$$

$$(ind)\ \frac{\vdash\forall^\alpha(\phi\to[\alpha]\phi)}{\phi\vdash[\alpha^*]\phi}$$

$$(con)\ \frac{\vdash\forall^\alpha\forall v{>}0\,(\varphi(v)\to\langle\alpha\rangle\varphi(v-1))}{\exists v\,\varphi(v)\vdash\langle\alpha^*\rangle\exists v{\leq}0\,\varphi(v)}$$

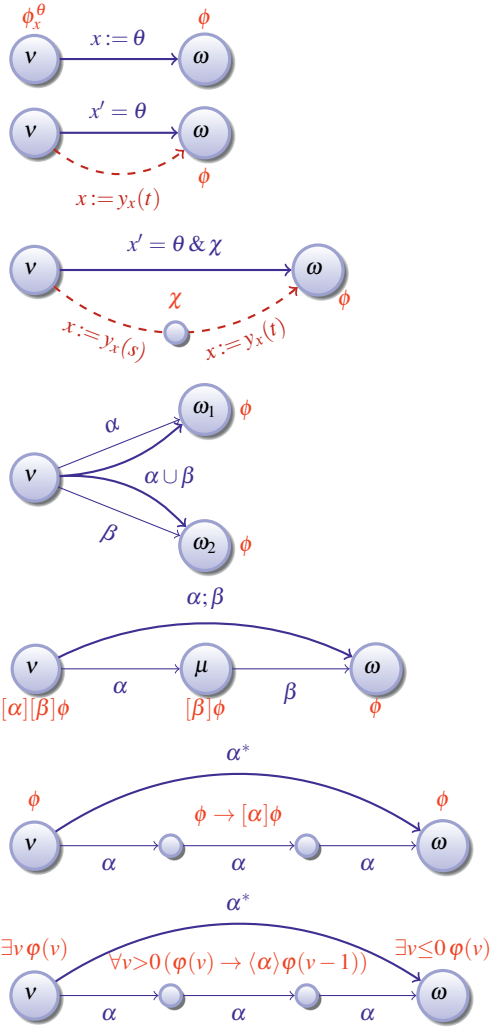


Fig. 2.12 Correspondence of dynamic proof rules and transition semantics

Sequential compositions are proven using nested modalities ($\langle;\rangle$,$[;]$). For rule $[;]$: If after all α-transitions, all β-transitions lead to states satisfying ϕ (i.e., $[\alpha][\beta]\phi$ holds), then also all transitions of the sequential composition $\alpha;\beta$ lead to states satisfying ϕ (i.e., $[\alpha;\beta]\phi$ holds). See, again, Fig. 2.12 for a graphical illustration of this proof principle. The dual rule $\langle;\rangle$ uses the fact that if there is an α-transition, after which there is a β-transition leading to ϕ (i.e., $\langle\alpha\rangle\langle\beta\rangle\phi$), then there is a transition of $\alpha;\beta$ leading to ϕ (that is, $\langle\alpha;\beta\rangle\phi$), because the transitions of $\alpha;\beta$ are just those that first do any α-transition, followed by any β-transition (Definition 2.7).

Rules $\langle{}^{*n}\rangle$,$[{}^{*n}]$ are the usual iteration rules, which partially unwind loops. Rule $\langle{}^{*n}\rangle$ uses the fact that ϕ holds after repeating α (i.e., $\langle\alpha^*\rangle\phi$), if ϕ holds at the beginning (for ϕ holds after zero repetitions then), or if, after one execution of α, ϕ holds after any number of repetitions of α, including zero repetitions (i.e., $\langle\alpha\rangle\langle\alpha^*\rangle\phi$). So rule $\langle{}^{*n}\rangle$ expresses that for $\langle\alpha^*\rangle\phi$ to hold, ϕ must hold either immediately or after one or more repetitions of α. Rule $[{}^{*n}]$ is the dual rule expressing that ϕ must hold after all of those combinations for $[\alpha^*]\phi$ to hold.

Tests are proven by showing (with a conjunction in rule $\langle?\rangle$) or assuming (with an implication in rule $[?]$) that the test succeeds, because test $?\chi$ can only make a transition when condition χ actually holds true (Definition 2.7). Thus, for dL formula $\langle?\chi\rangle\phi$ rule $\langle?\rangle$ is used to prove that χ holds true (otherwise there is no transition and thus the reachability property is false) and that ϕ holds after the resulting no-op. Rule $[?]$ for dL formula $[?\chi]\phi$, in contrast, assumes that χ holds true (otherwise there is no transition and thus nothing to show) and that ϕ holds after the resulting no-op.

Rule $\langle:=\rangle$ uses simultaneous substitutions from Fig. 2.10 for handling discrete jump sets. To show that ϕ is true after a discrete jump, $\langle:=\rangle$ shows that ϕ has been true before, when replacing the affected variables x_i with their new values θ_i in ϕ by an admissible substitution (Definition 2.8). Alternatively, the discrete jump set can also remain an unchanged prefix ($\mathscr{J}$ in Definition 2.10) for other dL rules applied to ϕ, until the substitution for rule $\langle:=\rangle$ becomes admissible. This is what our proof calculus uses instead of what we have shown in Example 2.16. Rule $[:=]$ uses the fact that discrete jump sets characterise a unique deterministic transition. Hence, its premise and conclusion are actually equivalent, because there is exactly one terminating transition for each discrete jump set. Assuming the presence of vacuous identity jumps $a:=a$ for variables a that do not otherwise change (vacuous identity jumps can be added as they do not change state), we can further use rule $\langle:=\rangle$ to *merge* subsequent discrete jumps into a single discrete jump set (see previous results [37] for a compatible calculus detailing jump set merging, which works without the need to add vacuous identity jumps $a:=a$):

$$\begin{array}{rl} & \vdash \langle z:=-\frac{b}{2}t^2+Vt, v:=V+1, a:=-b\rangle[\beta]\phi \\ \langle:=\rangle & \hline \vdash \langle a:=-b, v:=V\rangle\langle z:=\frac{a}{2}t^2+vt, v:=v+1, a:=a\rangle[\beta]\phi \\ [:=] & \hline \vdash \langle a:=-b, v:=V\rangle[z:=\frac{a}{2}t^2+vt, v:=v+1, a:=a][\beta]\phi \\ [;] & \hline \vdash \langle a:=-b, v:=V\rangle[z:=\frac{a}{2}t^2+vt, v:=v+1, a:=a;\beta]\phi \end{array}$$

More generally, $\langle x_1:=\theta_1,\ldots,x_n:=\theta_n\rangle\langle x_1:=\vartheta_1,\ldots,x_n:=\vartheta_n\rangle\phi$ can be merged by $\langle:=\rangle$ to $\langle x_1:=\vartheta_1{}^{\theta_1}_{x_1}\ldots{}^{\theta_n}_{x_n},\ldots,x_n:=\vartheta_n{}^{\theta_1}_{x_1}\ldots{}^{\theta_n}_{x_n}\rangle\phi$. Also see previous work [37] for more advanced and optimised merging techniques for state changes.

Given first-order definable flows for their differential equations, proof rules $\langle'\rangle$,$[']$ handle continuous evolutions (see [15, 189, 238] and App. B for flow approximation and solution techniques). These flows are combined in the discrete jump set $\mathscr{S}_t$. Given a solution $\mathscr{S}_t$ for the differential equation system with symbolic initial values $x_1,\ldots,x_n$, continuous evolution along differential equations can be replaced by a discrete jump $\langle\mathscr{S}_t\rangle$ with an additional quantifier for the evolution time t. The effect

of the constraint on χ is to restrict the continuous evolution such that its solution $\mathscr{S}_{\tilde{t}}$ remains in the evolution domain χ at all intermediate times $\tilde{t} \leq t$. This constraint simplifies to *true* if the evolution domain restriction χ is *true*, which makes sense, because there are no special constraints on the evolution (other than the differential equations) if the evolution domain region is described by *true*, hence the full space $\mathbb{R}^n$. A notable special case of rules $[']$ and $\langle' \rangle$ is when the evolution domain χ is *true*:

$$\frac{\forall t \geq 0 \, \langle \mathscr{S}_t \rangle \phi}{[x_1' = \theta_1, .., x_n' = \theta_n]\phi} \qquad \frac{\exists t \geq 0 \, \langle \mathscr{S}_t \rangle \phi}{\langle x_1' = \theta_1, .., x_n' = \theta_n \rangle \phi} \tag{2.12}$$

Similar simplifications can be made for convex invariant conditions (Sects. 2.9 and 3.8).

Global Rules

The last block of rules $[]gen, \langle\rangle gen, ind, con$ are *global rules*. They depend on the truth of their premises in all states reachable by hybrid program α, which is ensured by the universal closure $\forall^\alpha$ with respect to all bound state variables (Definition 2.8) of the respective hybrid program α. This universal closure overapproximates all possible change caused by α, because it comprises all bound variables. This universal closure is required for soundness in the presence of contexts Γ, Δ (Definition 2.10) or free variables. The global rules are given in a form that best displays their underlying logical principles. The general pattern for applying global rules to prove that the succedent of their conclusion holds is to prove that both their premise and the antecedent of their conclusion hold. In particular, the antecedent can be thought of as holding in the current state, whereas the premise can be thought of as holding in all reachable states because of the universal closure.

Rules $[]gen, \langle\rangle gen$ are generalisation rules and can be used to strengthen postconditions: antecedent $[\alpha]\phi$ is sufficient for proving succedent $[\alpha]\psi$ when postcondition ϕ entails ψ in all relevant states reachable by α, which are overapproximated by the universal closure $\forall^\alpha$ with respect to the bound variables of α. Clearly, for rule $[]gen$, if all states reachable by α satisfy ϕ ($[\alpha]\phi$) and ϕ implies ψ in all these states ($\forall^\alpha \phi \rightarrow \psi$), then ψ also holds in all states reachable by α ($[\alpha]\psi$). Similarly, for rule $\langle\rangle gen$, if some state reachable by α satisfies ϕ ($\langle\alpha\rangle\phi$) and ϕ implies ψ in all reachable states ($\forall^\alpha \phi \rightarrow \psi$), then ψ also holds in some state reachable by α ($\langle\alpha\rangle\psi$).

Rule *ind* is an induction schema with *inductive invariant* ϕ. Similarly, *con* is a generalisation of Harel's convergence rule [149] to the hybrid case with decreasing *variant* φ. Both rules are given in a form that best displays their underlying logical principles and similarity. Rule *ind* says that ϕ holds after any number of repetitions of α if it holds initially (antecedent) and, for all reachable states (as overapproximated by $\forall^\alpha$), invariant ϕ remains true after one iteration of α (premise). If ϕ is true after executing α whenever ϕ has been true before, then, if ϕ holds in the beginning, ϕ will continue to hold, no matter how often we repeat α in $[\alpha^*]\phi$; again, see Fig. 2.12 for an illustration. Rule *con* expresses that the variant $\varphi(v)$ holds for some real number $v \leq 0$ after repeating α sufficiently often if $\varphi(v)$ holds for some real

number at all in the beginning (antecedent) and, by premise, $\varphi(v)$ decreases after every execution of α by 1 (or another positive real constant). This rule can be used to show positive progress (by 1) with respect to $\varphi(v)$ by executing α.

For practical verification, rules *ind* or *con* can be combined with generalisation ($[]gen$,$\langle\rangle gen$) to prove a postcondition ψ of a loop α^* by showing that (*a*) the antecedents of the respective goals of *ind* and *con*, which represent the induction start, holds initially (*b*) their subgoals, which represent the induction step, hold and (*c*) the postcondition of the succedentd in their goals entails ψ. The corresponding variants of *ind* and *con* are *derived rules*. That is, these rules are non-essential, because they can be derived easily by chaining the proof rules from Fig. 2.11 together in an appropriate way.

$$(ind') \ \frac{\vdash \phi \quad \vdash \forall^{\alpha}(\phi \to [\alpha]\phi) \quad \vdash \forall^{\alpha}(\phi \to \psi)}{\vdash [\alpha^*]\psi}$$

$$(con') \ \frac{\vdash \exists v\, \varphi(v) \quad \vdash \forall^{\alpha}\forall v{>}0\,(\varphi(v) \to \langle\alpha\rangle\varphi(v-1)) \quad \vdash \forall^{\alpha}(\exists v{\leq}0\,\varphi(v) \to \psi)}{\vdash \langle\alpha^*\rangle\psi}$$

For example, using a cut with $\phi \to [\alpha^*]\phi$, rule *ind'* can be derived from *ind* and $[]gen$ as follows:

$$\dfrac{\dfrac{\dfrac{\vdash \forall^{\alpha}(\phi \to [\alpha]\phi)}{\phi \vdash [\alpha^*]\phi}\,ind}{\vdash \phi \to [\alpha^*]\phi}\,{\to}r \qquad \dfrac{\vdash \phi \qquad \dfrac{\vdash \forall^{\alpha}(\phi \to \psi)}{[\alpha^*]\phi \vdash [\alpha^*]\psi}\,[]gen}{\phi \to [\alpha^*]\phi \vdash [\alpha^*]\psi}\,{\to}l}{\vdash [\alpha^*]\psi}\,cut$$

These derived rules are not necessary in theory, but still useful in practise.

Derivability and Proofs

We call any formula ϕ provable or derivable (in the $\mathsf{d}\mathscr{L}$ calculus) if we can find a $\mathsf{d}\mathscr{L}$ proof for it that starts with axioms (rule *ax*) at the leaves and ends with a sequent $\vdash \phi$ at the bottom. While constructing proofs, however, we would start with the desired goal $\vdash \phi$ at the bottom and work our way backwards to the subgoals until they can be proven to be valid as axioms (*ax*). Once all subgoals have been proven to be valid axioms, they entail their consequences, which, recursively, entail the original goal $\vdash \phi$. This property of preserving truth or preserving entailment, which we prove in Sect. 2.6, is called soundness. Thus, while constructing proofs, we work bottom-up from the goal. When we have found a proof, we justify formulas from the axioms top-down to the original goal.

The notions of derivations and proofs for the $\mathsf{d}\mathscr{L}$ calculus are standard, except that i$\exists$ produces multiple conclusions. Hence, we define derivations as finite acyclic graphs instead of trees. We want proofs to be acyclic and not accept a formula that is used to prove itself.

Definition 2.11 (Provability). A *derivation* is a finite acyclic graph labelled with sequents such that, for every node, the (set of) labels of its children must be the (set

of) premises of an instance of one of the calculus rules (Definition 2.10) and the (set of) labels of the parents of these children must be the (set of) conclusions of that rule instance. A formula ψ is *provable* from a set Φ of formulas, denoted by $\Phi \vdash_{\mathsf{d}\mathscr{L}} \psi$, iff there is a finite subset $\Phi_0 \subseteq \Phi$ for which the sequent $\Phi_0 \vdash \psi$ is derivable, i.e., there is a derivation with a single root (i.e., node without parents) labelled $\Phi_0 \vdash \psi$.

Example 2.19. A very simple (in fact essentially propositional) proof of the formula

$$v^2 \leq 10 \land b > 0 \rightarrow b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10) \tag{2.13}$$

is shown in Fig. 2.13. The proof starts with the proof goal as a sequent at the bottom:

$$\vdash v^2 \leq 10 \land b > 0 \rightarrow b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10).$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{v^2 \leq 10, b > 0 \vdash b > 0}\,ax\,*}{v^2 \leq 10 \land b > 0 \vdash b > 0}\,\land\mathrm{l}}{v^2 \leq 10 \land b > 0 \vdash b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10)}\,\land\mathrm{r} \qquad \dfrac{\dfrac{\dfrac{}{v^2 \leq 10, b > 0 \vdash \neg(v \geq 0), v^2 \leq 10}\,ax\,*}{v^2 \leq 10 \land b > 0 \vdash \neg(v \geq 0), v^2 \leq 10}\,\land\mathrm{l}}{v^2 \leq 10 \land b > 0 \vdash \neg(v \geq 0) \lor v^2 \leq 10}\,\lor\mathrm{r}}{\vdash v^2 \leq 10 \land b > 0 \rightarrow b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10)}\,\rightarrow\mathrm{r}$$

Fig. 2.13 Simple propositional example proof

The first (i.e., bottom most) proof step applies proof rule $\rightarrow$r to turn the implication ($\rightarrow$) to the sequent level by moving the assumption into the antecedent. The next proof step applies rule $\land$r to split the proof into the left branch for showing that conjunct $b > 0$ follows from the assumptions in the antecedent and into the right branch for showing that conjunct $\neg(v \geq 0) \lor v^2 \leq 10$ follows from the antecedent also. On the left branch, the proof closes with an axiom *ax* after splitting the conjunction $\land$ on the antecedent with rule $\land$l. We mark closed proof goals with $*$. The right branch closes with an axiom *ax* after splitting the disjunction ($\lor$) in the succedent with rule $\lor$r and then splitting the conjunction ($\land$) in the antecedent with rule $\land$l. Now that all branches of the proof have closed (with *ax*), we know that all leaves at the top are valid, and, hence, since the premises are valid, each application of a proof rule ensures that their respective conclusions are valid also. By recursively following this derivation from the leaves at the top to the original root at the bottom, we see that the original goal is valid and formula (2.13) is, indeed, true under all circumstances (valid).

While this proof does not show anything particularly exciting, because it only uses propositional rules, it shows how a proof can be build systematically in the $\mathsf{d}\mathscr{L}$ calculus and gives an intuition about how validity is inherited from the premises to the conclusions. □

2.5.3 Deduction Modulo with Invertible Quantifiers and Real Quantifier Elimination

The first-order quantifier rules in Fig. 2.11 lift quantifier elimination to dℒ by following a generalised deduction modulo approach. They integrate decision procedures, e.g., for real quantifier elimination as a background prover [32], into the deductive proof system. Yet, unlike in the approaches of Dowek et al. [103] and Tinelli [290], the information given to the background prover is not restricted to ground formulas [290] or atomic formulas [103]. Further, real quantifier elimination is different from uninterpreted logic [147, 122, 134] in that the resulting formulas are not obtained by instantiation but by intricate arithmetic recombination. The quantifier rules can use any theory that admits quantifier elimination (see Definition 2.9) and has a decidable ground theory, for instance, the first-order theory of real arithmetic (which is equivalent to the theory of real-closed fields [288, 81]). A *formula of real arithmetic* is a first-order formula with $+,-,\cdot,/,=,\leq,<,\geq,>$ as the only function or predicate symbols besides constant symbols of Σ and logical variables of V. Also see App. D.2.

Integrating quantifier elimination to deal with statements about real quantities is quite challenging in the presence of modalities that influence the values of flexible symbols. In principle, quantifier elimination can be used to handle quantified constraints such as those arising for continuous evolutions. In dℒ, however, real quantifiers interact with modalities containing further discrete or continuous transitions, which is an effect nherent in the interacting nature of hybrid systems. A hybrid formula like $\exists z\,\langle z'' = -b; ?m-z \geq s; z'' = 0\rangle\, m-z < s$ is not first-order; hence quantifier elimination cannot be applied. Even more so, the effect of a modality depends on the solutions of the differential equations contained therein. The dynamics of a hybrid program depends on the values of its parameters (z, b, m, s in the above case), but, at the same time, the constraints on a quantified variable like z depend on the effect of the hybrid program. For instance, it is hard to know in advance, which first-order constraints need to be solved by QE for the above formula. To find out how z evolves from quantifier $\exists z$ to postcondition $m-z<s$, the system dynamics in the modality needs to be taken into account (as for repetitions). Hence, our calculus first unwraps the first-order structure before applying QE to the resulting arithmetic formulas.

2.5.3.1 Lifting Quantifier Elimination by Invertible Quantifier Rules

The purpose of the quantifier rules in Fig. 2.11 is to postpone QE until the actual arithmetic constraints become apparent. The idea is that ∀r,∃l,∃r, and ∀l temporarily remove quantifiers by introducing new auxiliary symbols for quantified variables such that the proof can be continued beyond the occurrence of the quantifier to further analyse the modalities contained therein. Later, when the actual first-order constraints for the auxiliary symbol have been discovered, the corresponding quantifier

$$\dfrac{\dfrac{\dfrac{}{v\geq 0, z<m \vdash v^2>2b(m-z)}}{\vdash v\geq 0\wedge z<m\rightarrow v^2>2b(m-z)}\,{\scriptstyle \rightarrow r,\wedge l} \quad \dfrac{\dfrac{\dfrac{}{v\geq 0, z<m \vdash -\frac{b}{2}T^2+vT+z>m}}{v\geq 0, z<m \vdash \langle z:=-\frac{b}{2}T^2+vT+z\rangle z>m}\,{\scriptstyle \langle:=\rangle}}{v\geq 0, z<m \vdash T\geq 0}\,{\scriptstyle i\exists}}{v\geq 0, z<m \vdash T\geq 0\wedge \langle z:=-\frac{b}{2}T^2+vT+z\rangle z>m}\,{\scriptstyle \wedge r}$$

$$\dfrac{v\geq 0, z<m \vdash \exists t{\geq}0\, \langle z:=-\frac{b}{2}t^2+vt+z\rangle z>m}{v\geq 0, z<m \vdash \langle z'=v, v'=-b\rangle z>m}\,{\scriptstyle \langle'\rangle}$$

$$\dfrac{v\geq 0, z<m \vdash \exists t{\geq}0\, \langle z:=-\frac{b}{2}t^2+vt+z\rangle z>m}{v\geq 0, z<m \vdash \langle z'=v, v'=-b\rangle z>m}\,{\scriptstyle \exists r}$$

$$\dfrac{v\geq 0, z<m \vdash \langle z'=v, v'=-b\rangle z>m}{\vdash v\geq 0\wedge z<m\rightarrow \langle z'=v, v'=-b\rangle z>m}\,{\scriptstyle \rightarrow r,\wedge l}$$

Fig. 2.14 Deduction modulo for analysis of MA violation in braking mode

can be reintroduced (i$\forall$, i$\exists$) and quantifier elimination QE is applied to reduce the sequents equivalently to a simpler formula with less (distinct) symbols. In $\exists$r,$\forall$l,i$\exists$, the respective auxiliary symbols are free logical variables. In $\forall$r,$\exists$l,i$\forall$, Skolem function terms are used instead for reasons that are crucial for soundness and will be illustrated in the remainder of this section. In this context, we think of *free* logical variables as being introduced by γ-rules ($\exists$r and $\forall$l), and hence implicitly existentially quantified.

To illustrate how quantifier and dynamic rules of $\mathsf{d}\mathscr{L}$ interact to combine arithmetic with dynamic reasoning in hybrid systems, we analyse the braking behaviour in train control. The proof in Fig. 2.14 can be used to analyse whether a train can violate its MA although it is braking. That is, if the train position z can leave m ($z>m$) although it starts inside ($z<m$) and is braking will full braking force all the time:

$$v\geq 0\wedge z<m\rightarrow \langle z'=v, v'=-b\rangle z>m.$$

As the proof reveals, the answer depends on the initial velocity v. The proof starts with the conjecture at the bottom and applies propositional transformation rules $\rightarrow$r,$\wedge$l to obtain a decomposed sequent form. Then it uses rule $\langle'\rangle$ to replace the differential equation with a quantified formula about its solution. For notational convenience, we use the simplified $\langle'\rangle$ rule from (2.12), as the differential equation is not restricted to an evolution domain. Now we have a quantified modal formula, $\exists t{\geq}0\, \langle z:=-\frac{b}{2}t^2+vt+z\rangle z>m$, which, unfortunately, cannot be handled by quantifier elimination in real-closed fields, because it is not first-order. Using rule $\exists$r, however, the proof can continue by introducing a new free variable T for the quantified variable t and postpone QE. After introducing T, the proof can continue by splitting a conjunction in the succedent into two branches (rule $\wedge$r) and applying the assignment with a substitution on the right branch (rule $\langle:=\rangle$). Finally, the previously quantified free variable T only occurs in first-order formulas on all open goals. Then rule i$\exists$ can be applied in Fig. 2.14 to merge all open proof goals mentioning T, reintroduce the quantifier for T, and apply quantifier elimination. The conjunction of the two goals can be handled by QE and simplification, yielding the resulting subgoal:

$$\mathrm{QE}\left(\exists T\left((v \geq 0 \wedge z < m \rightarrow T \geq 0) \wedge (v \geq 0 \wedge z < m \rightarrow -\frac{b}{2}T^2 + vT + z > m)\right)\right)$$
$$\equiv v \geq 0 \wedge z < m \rightarrow v^2 > 2b(m-z).$$

After applying rules →r,∧l for structural reasons again, the open branch with this formula reveals the speed limit and can be used to synthesise a corresponding parameter constraint. When $v^2 > 2b(m-z)$ holds initially, m can eventually be violated even in braking mode, as the velocity exceeds the braking force.

Similarly, the dual constraint $v^2 \leq 2b(m-z)$ guarantees that m can be respected by appropriate braking. The constraint so discovered thus forms a *controllability constraint* of ETCS, i.e., a constraint that characterises from which states control choices exist that guarantee safety. It is essentially equivalent to $[z'' = -b]z \leq m$ and $\exists a\,(-b \leq a \leq A \wedge [z'' = a]z \leq m)$. The resulting controllable region of the state space of ETCS is illustrated in Fig. 2.15.

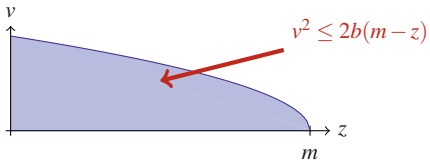


Fig. 2.15 Controllable region of ETCS dynamics

For comparison, the dual formula $v \geq 0 \wedge z < m \rightarrow [z' = v, v' = -b]\,z \leq m$ can be analysed as shown in Fig. 2.16 to study under which circumstances the MA is always respected ($[z'' = -b]z \leq m$) rather than under which it can fail ($\langle z'' = -b\rangle z > m$). The outcome again discovers the controllability constraint. The difference of the

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{v \geq 0, z < m \vdash v^2 \leq 2b(m-z)}{\vdash v \geq 0 \wedge z < m \rightarrow v^2 \leq 2b(m-z)}\,\rightarrow\mathrm{r},\wedge\mathrm{l}}{v \geq 0, z < m, s \geq 0 \vdash -\frac{b}{2}s^2 + vs + z \leq m}\,\mathrm{i}\forall}{v \geq 0, z < m, s \geq 0 \vdash \langle z := -\frac{b}{2}s^2 + vs + z\rangle z \leq m}\,\langle:=\rangle}{v \geq 0, z < m, s \geq 0 \vdash [z := -\frac{b}{2}s^2 + vs + z]z \leq m}\,[:=]}{v \geq 0, z < m \vdash s \geq 0 \rightarrow [z := -\frac{b}{2}s^2 + vs + z]z \leq m}\,\rightarrow\mathrm{r}}{v \geq 0, z < m \vdash \forall t{\geq}0\,[z := -\frac{b}{2}t^2 + vt + z]z \leq m}\,\forall\mathrm{r}}{v \geq 0, z < m \vdash [z' = v, v' = -b]z \leq m}\,[']}{\vdash v \geq 0 \wedge z < m \rightarrow [z' = v, v' = -b]z \leq m}\,\rightarrow\mathrm{r},\wedge\mathrm{l}$$

Fig. 2.16 Deduction modulo for analysis of MA-safety in braking mode

deduction in Fig. 2.16 compared to that in Fig. 2.14 is that we now use rule [′], which gives a universal quantifier for time t. With rule ∀r, the quantifier can be turned into a Skolem constant term s, which does not have any arguments, because no free logical variables occur. After applying the solution of the differential equation with

$[:=],\langle:=\rangle$, the resulting formula is first-order in the Skolem term s. Then rule i$\forall$ can be used to reintroduce a universal quantifier for the previously quantified variable, and to apply quantifier elimination:

$$\begin{aligned}&\mathrm{QE}\left(\forall s\,(v \geq 0 \land z < m \land s \geq 0 \rightarrow -\frac{b}{2}s^2 + vs + z \leq m)\right)\\ \equiv\ & v \geq 0 \land z < m \rightarrow v^2 \leq 2b(m-z).\end{aligned}$$

2.5.3.2 Admissibility in Invertible Quantifier Rules

The requirement that substitutions in i$\forall$ are admissible implies that no occurrence of $s(X_1,\ldots,X_n)$ is within the scope of a quantifier for any of these X_i. Admissibility makes sense, because variables in $s(X_1,\ldots,X_n)$ would otherwise be captured by quantifiers when substituting. The admissibility condition prevents i$\forall$ from rearranging the order of quantifiers from $\exists X_i\,\forall s$ to the weaker $\forall s\,\exists X_i$. Such a rearrangement would be unsound, because it is not sufficient to show the weak subgoal $\forall s\,\exists X_i$ (each s has an X_i) in order to prove the strong statement $\exists X_i\,\forall s$ saying that the same X_i works for all s. Because this is an important part of soundness, we illustrate in detail why unsound rearrangements are prevented.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{i}\forall \text{ is not applicable}}{\vdash \mathrm{QE}(\exists X\,(2X+1 < s(X)))}}{\vdash 2X+1 < s(X)}\,\text{i}\exists}{\vdash \langle x:=2X+1\rangle(x < s(X))}\,\langle:=\rangle}{\vdash \forall y\,\langle x:=2X+1\rangle(x < y)}\,\forall\text{r}}{\vdash \exists x \forall y\,\langle x:=2x+1\rangle(x < y)}\,\exists\text{r}$$

Fig. 2.17a Wrong rearrangement with deduction modulo by invertible quantifiers

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overbrace{\vdash \mathrm{QE}\,(\exists X\ \mathrm{QE}(\forall s\,(2X+1 < s)))}^{\text{false}}}{\vdash \mathrm{QE}(\forall s\,(2X+1 < s))}\,\text{i}\exists}{\vdash 2X+1 < s(X)}\,\text{i}\forall}{\vdash \langle x:=2X+1\rangle(x < s(X))}\,\langle:=\rangle}{\vdash \forall y\,\langle x:=2X+1\rangle(x < y)}\,\forall\text{r}}{\vdash \exists x \forall y\,\langle x:=2x+1\rangle(x < y)}\,\exists\text{r}$$

Fig. 2.17b Correct reintroduction order with deduction modulo by invertible quantifiers

For the moment, suppose the rules did not contain QE. The requirement for admissible substitutions (Definition 2.8) ensures that the proof attempt of an invalid formula in Fig. 2.17a cannot close in the $\mathsf{d}\mathscr{L}$ calculus. At the indicated position at the top, i$\forall$, which would unsoundly invert the quantifier order to $\forall S\,\exists X$, cannot be applied: In i$\forall$, the substitution inserting $s(X)$ gives $\exists Y\,(2Y+1 < s(X))$ by bound variable renaming instead of $\exists X\,(2X+1 < s(X))$, because the substitution would not otherwise be admissible. Thus, i$\forall$ is not applicable, because the quantified formula is not of the form $\Psi(s(X))$.

Now, we consider what happens in the presence of QE. The purpose of QE is to (equivalently) remove quantifiers like $\exists X$. Thus it is no longer obvious that the admissibility argument applies, because the blocking variable X would have disappeared after successful quantifier elimination. However, quantifier elimination over the reals is defined in the first-order theory of real arithmetic [288, 81]. Yet, when

eliminating X in Fig. 2.17a, the Skolem term $s(X)$ is no term of real arithmetic, as, unlike that of $+$, the interpretation of the Skolem function s is arbitrary. The truth-value of $\exists X\,(2X+1<s(X))$ depends on the interpretation of s. If $I(s)$ happens to be a constant function, the formula is true, if $I(s)(a)=2a$, however, it is false. In general, such cases cannot be distinguished without quantifiers, because two functions cannot be shown to be identical by evaluating them at finitely many points. Thus, in the presence of uninterpreted function terms, real arithmetic does not generally admit quantifier elimination. Consequently, i$\exists$ and i$\forall$ are only applicable if QE is defined. Yet, we show that QE can be lifted to formulas with Skolem functions, nevertheless, when these are instances of real arithmetic formulas:

Lemma 2.5 (Quantifier elimination lifting). *Quantifier elimination can be lifted to instances of formulas of first-order theories that admit quantifier elimination, i.e., to formulas that result from the base theory by substitution.*

Proof. Let formula ϕ be an instance of ψ, with ψ being a formula of the base theory, i.e., ϕ is $\psi_{z_1}^{\theta_1}\ldots_{z_n}^{\theta_n}$ for some variables z_i and arbitrary terms θ_i. As QE is defined for the base theory, let $\mathrm{QE}(\psi)$ be the quantifier-free formula belonging to ψ according to Definition 2.9. Then $\mathrm{QE}(\psi)_{z_1}^{\theta_1}\ldots_{z_n}^{\theta_n}$ satisfies the requirements of Definition 2.9 for ϕ, because $\vDash \psi_{z_1}^{\theta_1}\ldots_{z_n}^{\theta_n} \leftrightarrow \mathrm{QE}(\psi)_{z_1}^{\theta_1}\ldots_{z_n}^{\theta_n}$: For F defined as $\psi\leftrightarrow\mathrm{QE}(\psi)$, we have that $\vDash F$ implies $\vDash F_{z_1}^{\theta_1}\ldots_{z_n}^{\theta_n}$ by a standard consequence of the Substitution Lemma 2.2. And $\psi\leftrightarrow\mathrm{QE}(\psi)$ is indeed valid, by the properties of QE; see Definition 2.9. □

With this, consider again the example in Fig. 2.17a. By Lemma 2.5, QE is defined in the presence of Skolem terms that do not depend on quantified variables, e.g., for $\exists X\,(2X+1<t(Y,Z))$, which is an instance of the form $(\exists X\,(2X+1<z))_z^{t(Y,Z)}$. However, QE is *not defined* in the premise of i$\exists$ when Skolem dependencies on X occur. In Fig. 2.17a, $\exists X\,(2X+1<s(X))$ is no instance of first-order real arithmetic, because, by bound variable renaming $(\exists X\,(2X+1<z))_z^{s(X)}$ yields a different formula $\exists Y\,(2Y+1<s(X))$. An occurrence of $s(X)$, which corresponds to a quantifier nesting of $\exists X\forall s$, thus requires $s(X)$ to be eliminated by i$\forall$ before i$\exists$ can eliminate X; see Fig. 2.17b. Hence, inner universal quantifiers are enforced to be handled first and unsound quantifier rearrangements are prevented even in the presence of QE.

Finally, observe that i$\forall$ and i$\exists$ do not require quantifiers to be eliminated in the exact same order in which they occurred in the original formula. The elimination order within homogeneous quantifier blocks like $\forall x_1\forall x_2$ is not restricted as there are no Skolem dependencies among the corresponding auxiliary Skolem terms. Yet, eliminating such a quantifier block is sound in any order (accordingly for $\exists x_1\exists x_2$). Similarly, i$\exists$ and i$\forall$ could interchange the order of $\forall x\exists y$ to the stronger $\exists y\forall x$, because the resulting Skolem term s for x in the former formula does not depend on y. In this direction, however, the interchange is sound, as it amounts to proving a stronger statement. This quantifier rearrangement is not necessarily wise, because it requires proving a stronger formula, but it is at least sound.

2.5.3.3 Quantifier Elimination and Modalities

Quantifier elimination over the first-order theory of reals cannot handle modal formulas. Hence, the $\mathsf{d}\mathscr{L}$ calculus first reduces modalities to first-order constraints before applying QE. Yet, this is not necessary for all modalities. The modal subformula in the following example does not impose any constraints on X, but its truth-value only determines which first-order constraints are imposed on X:

$$\mathrm{QE}(\exists X\,(X<0\wedge((\langle y:=2y+1\rangle y>0)\rightarrow X>y)))\equiv(\langle y:=2y+1\rangle y>0)\rightarrow y<0.$$

Modal formulas not containing elimination variable X can be handled by propositional abstraction in QE and remain unchanged. Syntactically, the reason for this is that $\mathsf{d}\mathscr{L}$ rule applications on modal formulas that do not contain X will never produce formulas which do. The semantical reason for the same fact is a generalisation of the coincidence lemma to $\mathsf{d}\mathscr{L}$, which says that values of variables that do not occur will neither affect the transition structure of a hybrid program nor the truth-value of formulas.

Lemma 2.6 (Coincidence lemma). *If the interpretations (and assignments and states, respectively) I,η,ν and J,ε,ω agree on all symbols that occur freely in the formula ϕ, then $val_{I,\eta}(\nu,\phi)=val_{J,\varepsilon}(\omega,\phi)$.*

Proof. The proof is by a simple structural induction using the definitions of valuation $val_{I,\eta}(\nu,\cdot)$ and $\rho_{I,\eta}(\cdot)$ in Definitions 2.5–2.7. □

2.5.3.4 Global Invertible Quantifier Rules

Rules i∀ and i∃ display an asymmetry. While i∀ works locally on a branch, i∃ needs to inspect all branches that contain X. The reason for this is that branches are implicitly combined conjunctively in sequent calculus, as all branches have to close simultaneously for a proof to succeed (Definition 2.11). Universal quantifiers can be handled separately for conjunctions by $\forall x(\phi\wedge\psi)\equiv\forall x\,\phi\wedge\forall x\,\psi$. Existential quantifiers, however, can only be dealt with separately for disjunctions but not for conjunctions: $\exists x(\phi\vee\psi)\equiv\exists x\,\phi\vee\exists x\,\psi$. In calculi with a disjunctive proof structure, the roles of i∀ and i∃ would be interchanged but the phenomenon remains.

Rule i∃ can be applied to the full proof (i.e., all open goals) as a global closing substitution in the free-variable tableau calculus [122]; cf. App. A. By Lemma 2.6, however, rule i∃ only needs to consider the set of all open goals $\Phi_i\vdash\Psi_i$ that actually contain X. Rule i∃ resembles global closing substitutions in uninterpreted free-variable tableaux [134]. Both avoid the backtracking over closing substitutions that local closing substitutions require. Unlike closing substitutions, however, rule i∃ uses the fixed semantics of function and predicate symbols of real arithmetic such that variables can be eliminated equivalently by QE before the proof completes. Applying i∀ or i∃ early does not necessarily close the proof. Instead, equivalent constraints on the remaining variables will be revealed, which can simplify the proof or help in deriving parametric constraints or invariants.

2.5.4 Verification Example

As a simple example to prove, recall the bouncing ball system (*ball*) from Example 2.5 on p. 45 and its dL specification from Example 2.7 on p. 48. Consider the intuitive property that the bouncing ball never bounces higher than initial height H when the precondition of property (2.3) holds initially:

$$(v^2 \leq 2g(H-h) \land h \geq 0 \land g > 0 \land H \geq 0 \land 1 > c \geq 0) \rightarrow [ball](0 \leq h \leq H). \quad (2.3^*)$$

The bouncing ball is very simple, but shows some interesting aspects of proofs. In order to simplify the proof notation, let us discard clock variable τ. Clock τ is not necessary for the property, and only used to ensure natural switching during the bounce to prevent the ball from bouncing multiple times while still on the ground (which would be *superdense* switching with multiple discrete switches at the exact same point in time).

For the proof, we define some abbreviations. Let ψ denote the general assumptions in the precondition about parameters that do not change during bouncing ball runs, and let ϕ denote the state-dependent part of the precondition, that is:

$$\psi \equiv g > 0 \land H \geq 0 \land 1 > c \geq 0,$$
$$\phi \equiv v^2 \leq 2g(H-h) \land h \geq 0.$$

The dL proof for the bouncing ball property (2.3) is shown in Fig. 2.18.

The proof starts with the property (2.3) at the bottom (goal). After normalising to sequent form with rules $\rightarrow$r,$\land$l, the proof follows an induction (using the rule *ind′* from p. 86) with invariant ϕ. Rule *ind′* produces two other proof subgoals that are not shown in Fig. 2.18: the proof goal that the precondition $\psi \land \phi$ implies the invariant ϕ (i.e., $\psi, \phi \vdash \phi$) and the proof goal that the invariant ϕ implies the post-condition, which gives $\psi, \phi \vdash \forall h \forall v\,(\phi \rightarrow 0 \leq h \leq H)$. Both goals are trivial to prove by *ax* and $\forall$r,i$\forall$, respectively. The quantifiers $\forall h \forall v$ in the latter goal result from the universal closure $\forall^\alpha$ in rule *ind′*. Universal closures are not strictly necessary in this particular proof, because the premise only contains invariant ϕ and formula ψ about symbols that do not change during the hybrid program runs. Thus, the universal closure immediately disappears after applying $\forall$r. In general, however, universal closures in *ind* and the other global proof rules are critical for soundness; see Fig. 2.19a versus Fig. 2.19b.

After splitting the sequential composition by dL rule [;], the proof uses rule $[']$ with the solution $\langle h := h + vt - \frac{g}{2}t^2, v := v - gt \rangle$ of the differential equation system $h' = v, v' = -g$. We abbreviate this solution by $\langle \mathscr{S}_t \rangle$. Again, we use the simplified $[']$ rule from (2.12). QE cannot be applied to the result quantifier $\forall t \geq 0 \langle \mathscr{S}_t \rangle \ldots$, because the quantified variable t occurs in modalities to which QE is not applicable. Thus the proof uses $\forall$r to introduce a Skolem function s for the previously quantified variable t. But unlike for the proof in Fig. 2.14, we do not directly apply the resulting solution $\langle \mathscr{S}_s \rangle$ by rule $\langle := \rangle$. The reason is the different system structure of the bouncing ball. In the bouncing ball program, the differential equation comes

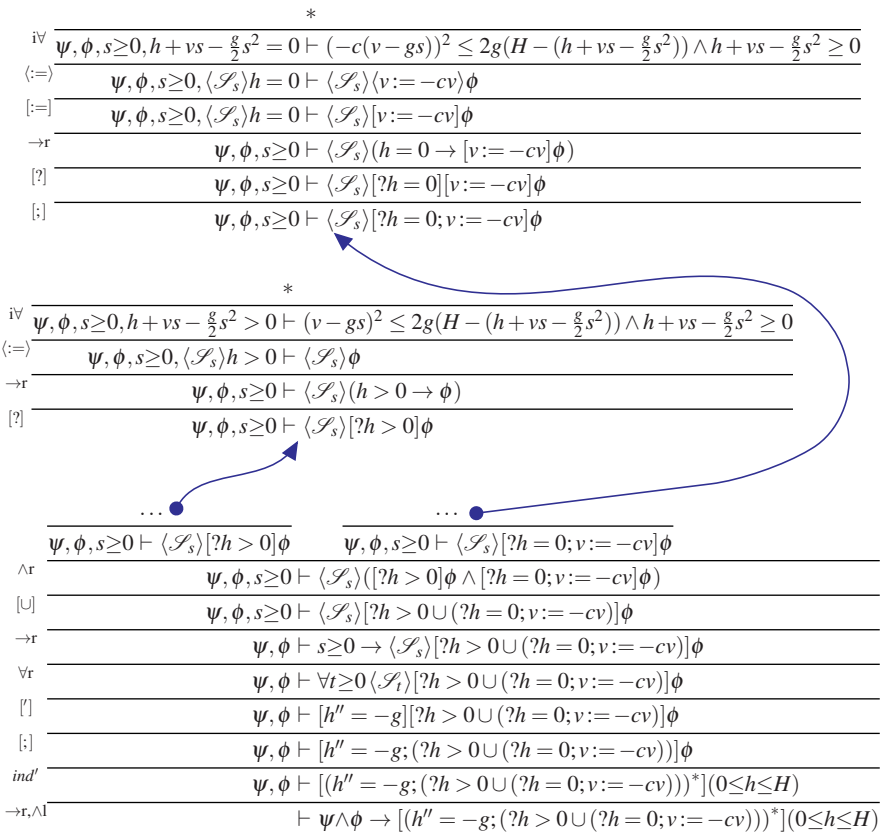


Fig. 2.18 Bouncing ball proof (no evolution domain)

unsound

$$\dfrac{\dfrac{\dfrac{x\leq 0, x\leq 1 \vdash x+1\leq 1}{x\leq 0, x\leq 1 \vdash [x:=x+1]x\leq 1}}{x\leq 0 \vdash x\leq 1\rightarrow[x:=x+1]x\leq 1}}{x\leq 0 \vdash [(x:=x+1)^*]x\leq 1}$$

Fig. 2.19a Unsound attempt of induction without universal closure $\forall^\alpha$

not provable

$$\dfrac{\dfrac{\dfrac{x\leq 0, y\leq 1 \vdash y+1\leq 1}{x\leq 0, y\leq 1 \vdash [y:=y+1]y\leq 1}\,[:=],\langle:=\rangle}{x\leq 0 \vdash y\leq 1\rightarrow[y:=y+1]y\leq 1}\,\rightarrow\mathrm{r}}{x\leq 0 \vdash \forall x\,(x\leq 1\rightarrow[x:=x+1]x\leq 1)}\,\forall\mathrm{r}}{x\leq 0 \vdash [(x:=x+1)^*]x\leq 1}\,ind$$

Fig. 2.19b Correct use of induction with universal closure $\forall^\alpha$, i.e., $\forall x$

first, and the discrete control equations are executed after that. Thus we keep the discrete jump set for the solution $\langle \mathscr{S}_s \rangle$ as an unmodified discrete jump prefix ($\langle \mathscr{J} \rangle$ in Definition 2.10) for the following rule applications and only apply the assignment with rule $\langle := \rangle$ to first-order formulas at the end of the proof.

On a side note: we could, in fact, just as well have used rule $\langle := \rangle$ here right away and substituted v, h inside the hybrid programs immediately, because there are no remaining loops or differential equations. That would clutter the notation, though, and we want to illustrate how discrete jump sets can be used as unmodified proof rule prefixes in Fig. 2.18.

Leaving prefix $\langle \mathscr{S}_s \rangle$ unchanged, the proof in Fig. 2.18 continues by splitting the choice between $h > 0$ and $h = 0$ with rule $[\cup]$ into two conjuncts, which split into two branches by rule $\wedge$r. On both branches, which are continued as indicated by the arrows, the test statements are turned into implications by rule $[?]$ and, ultimately, the accumulated discrete jump sets are applied (with rules $[:=], \langle := \rangle$) when the remaining formulas are simple. The last step of the proof is to reintroduce quantifiers for Skolem term s by rule i$\forall$ and apply quantifier elimination to the resulting first-order formulas on the left and right branches respectively:

$$\mathrm{QE}\Big(\forall s\,\big(\psi \wedge \phi \wedge s{\geq}0 \wedge h + vs - \frac{g}{2}s^2 > 0 \\ \rightarrow (v - gs)^2 \leq 2g(H - (h + vs - \frac{g}{2}s^2)) \wedge h + vs - \frac{g}{2}s^2 \geq 0\big)\Big) \equiv \mathit{true};$$

$$\mathrm{QE}\Big(\forall s\,\big(\psi \wedge \phi \wedge s{\geq}0 \wedge h + vs - \frac{g}{2}s^2 = 0 \\ \rightarrow (-c(v - gs))^2 \leq 2g(H - (h + vs - \frac{g}{2}s^2)) \wedge h + vs - \frac{g}{2}s^2 \geq 0\big)\Big) \equiv \mathit{true}.$$

In the proof of Fig. 2.18, we have used a bouncing ball without an evolution domain restriction. The bouncing ball property can also be proven with its evolution domain restricted to $h \geq 0$ on the differential equation system as in Fig. 2.2; see Fig. 2.20 for a proof. The proof is slightly more involved compared to Fig. 2.18, because of the extra constraints from the non-simplified rule $[']$. This time, for a change, we simply choose the full precondition as invariant, although the part marked in grey is still unaffected by the dynamics:

$$\phi \equiv v^2 \leq 2g(H - h) \wedge h \geq 0 \wedge g > 0 \wedge H \geq 0 \wedge 1 > c \geq 0.$$

Note, in particular, that there are many invariants that can be used to prove the same property. We just need to find one invariant that works.

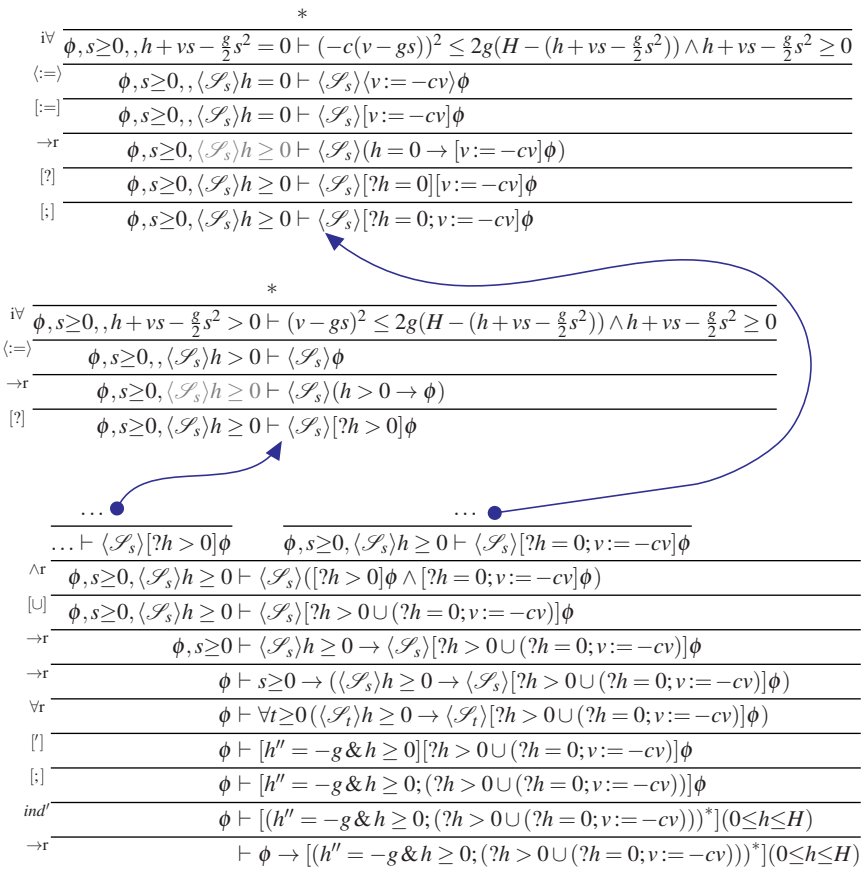


Fig. 2.20 Bouncing ball proof (with evolution domain)

2.6 Soundness

In this section, we prove that the $\mathsf{d}\mathscr{L}$ calculus is a sound axiomatisation of the transition behaviour of hybrid systems. Whatever we can prove in the $\mathsf{d}\mathscr{L}$ calculus is actually true.

The proof calculus for $\mathsf{d}\mathscr{L}$ in Fig. 2.11 needs to fit to the semantics of differential dynamic logic from Sect. 2.3; otherwise, the proof rules would not be meaningful. Fortunately, every differential dynamic logic formula that can be derived in the $\mathsf{d}\mathscr{L}$ calculus from Fig. 2.11 really is a valid formula! This property of the calculus is called *soundness* and is crucial, because it would be disastrous if a formula would be called "proven" when it is actually not valid, since we could not trust our proofs then. A calculus is sound iff every formula that can be derived in the calculus is also valid according to the semantics.

We prove that a successful deduction in the dℒ calculus always produces correct verification results about hybrid systems: The dℒ calculus is *sound*, i.e., all provable (closed) formulas are valid in all states of all interpretations. We can restrict our attention to closed formulas, i.e., formulas without free variables to begin with, because we can start with the universal closure of the formula for validity just as well. To reflect the interaction of free variables and Skolem terms, we adapt the notion of soundness for the liberalised δ^+-rule in free-variable tableau calculi [147] to sequent calculus.

A formula ϕ *has a model* [147] if there is an interpretation I and a state ν such that *for all* variable assignments η we have $I,\eta,\nu \models \phi$. Closed tableaux prove the unsatisfiability of the negated goal [147]. Sequent calculi work dually and show validity of the original proof goal. Consequently, we use the dual notion and say that formula ψ is a *consequence* of ϕ iff, for every I,ν *there is* an assignment η such that $I,\eta,\nu \models \psi$ provided that, for every I,ν, *there is* an assignment η such that $I,\eta,\nu \models \phi$. A proof rule that concludes Ψ from the premises Φ is *sound* if Ψ is, indeed, a consequence of Φ in the sense just defined. As usual, multiple branches in Ψ or Φ are combined conjunctively.

In this context, we think of free logical variables as being introduced by γ-rules, i.e., $\exists$r and $\forall$l (hence the implicit existential quantification of free logical variables by η). For closed formulas (without free logical variables), validity corresponds to being a consequence from an empty set of open goals. Hence, closed formulas that are provable with a sound deduction are *valid* (true in all states of all interpretations).

Theorem 2.1 (Soundness of dℒ). *The dℒ calculus is sound.*

Proof. The calculus is sound if each rule instance is sound. All rules of the dℒ calculus except $\forall$r,$\exists$l and i$\exists$ are also *locally sound*, i.e., their conclusion is true at I,η,ν if all its premises are true in I,η,ν, which implies soundness. It is also easy to show that locally sound rules remain sound when adding contexts $\Gamma,\Delta,\langle\mathscr{J}\rangle$ as in Definition 2.10, since a discrete jump set $\langle\mathscr{J}\rangle$ characterises a unique state transition. Local soundness proofs of $\langle;\rangle$,[;],$\langle\cup\rangle$,[$\cup$],$\langle^{*n}\rangle$,[^{*n}],$\langle?\rangle$,[?] and propositional rules are as usual. Note that, for symmetric rules, local soundness implies that the premise and conclusion are *equivalent*, i.e., true in the same states. For an illustration of the dynamics behind the dynamic proof rules, we recall Fig. 2.12 from p. 83.

$\langle:=\rangle$ The rule $\langle:=\rangle$ is locally sound. Assume that the premise holds in I,η,ν, i.e., $I,\eta,\nu \models \phi^{\theta_1\ldots\theta_n}_{x_1\ldots x_n}$. We have to show that $I,\eta,\nu \models \langle x_1:=\theta_1,\ldots,x_n:=\theta_n\rangle\phi$, i.e., $I,\eta,\omega \models \phi$ for a state ω with $(\nu,\omega) \in \rho_{I,\eta}(x_1:=\theta_1,\ldots,x_n:=\theta_n)$. This follows directly from the Substitution Lemma 2.2 for admissible substitutions (Definition 2.8). The proof for rule [:=] uses the fact that discrete jumps are deterministic.

$\langle;\rangle$ Rule $\langle;\rangle$ is locally sound. Assume that the premise holds in I,η,ν, i.e., $I,\eta,\nu \models \langle\alpha\rangle\langle\beta\rangle\phi$. We have to show that the conclusion holds in I,η,ν, i.e., $I,\eta,\nu \models \langle\alpha;\beta\rangle\phi$. By premise, $I,\eta,\nu \models \langle\alpha\rangle\langle\beta\rangle\phi$, we know that there is a state μ such that $(\nu,\mu) \in \rho_{I,\eta}(\alpha)$ and $I,\eta,\mu \models \langle\beta\rangle\phi$. Hence, there is a state ω such that $(\mu,\omega) \in \rho_{I,\eta}(\beta)$ and $I,\eta,\omega \models \phi$. Now, by the semantics

of $\alpha;\beta$ (Definition 2.7), there is a transition from ν to ω (via intermediate state μ) along $\alpha;\beta$. Thus, $(\nu,\omega)\in\rho_{I,\eta}(\alpha;\beta)$ and $I,\eta,\omega\models\phi$, which implies $I,\eta,\nu\models\langle\alpha;\beta\rangle\phi$. The converse direction can be proven similarly to show equivalence and the local soundness of the dual rule $[;]$.

$\langle\cup\rangle$ Rule $\langle\cup\rangle$ is locally sound. Assume that the premise holds in I,η,ν, i.e., $I,\eta,\nu\models\langle\alpha\rangle\phi\vee\langle\beta\rangle\phi$. We have to show that the conclusion holds in I,η,ν, i.e., $I,\eta,\nu\models\langle\alpha\cup\beta\rangle\phi$. If the disjunction in the premise is true, then one of its disjuncts must hold in I,η,ν. Consider the case where $I,\eta,\nu\models\langle\alpha\rangle\phi$. Then there is a state ω such that $(\nu,\omega)\in\rho_{I,\eta}(\alpha)$ and $I,\eta,\omega\models\phi$. By the semantics of $\alpha\cup\beta$ in Definition 2.7, every transition of α is a transition of $\alpha\cup\beta$. Hence $(\nu,\omega)\in\rho_{I,\eta}(\alpha\cup\beta)$ and $I,\eta,\omega\models\phi$, which imply $I,\eta,\nu\models\langle\alpha\cup\beta\rangle\phi$. If, instead, the second disjunct $I,\eta,\nu\models\langle\beta\rangle\phi$ holds, then the proof is similar. Either way, we have $I,\eta,\nu\models\langle\alpha\cup\beta\rangle\phi$. The converse direction can be proven accordingly to show equivalence and the local soundness of the dual rule $[\cup]$.

$\langle{}^{*n}\rangle$ Rule $\langle{}^{*n}\rangle$ is locally sound. Assume that the premise holds in I,η,ν, i.e., assume $I,\eta,\nu\models\phi\vee\langle\alpha\rangle\langle\alpha^*\rangle\phi$. We have to show that the conclusion holds in I,η,ν, i.e., $I,\eta,\nu\models\langle\alpha^*\rangle\phi$. The disjunction in the premise holds; hence, one of the disjuncts holds. Consider the case where $I,\eta,\nu\models\phi$; then $I,\eta,\nu\models\langle\alpha^*\rangle\phi$ already holds with zero repetitions α^* for ϕ is true in the beginning. Consider the case where $I,\eta,\nu\models\langle\alpha\rangle\langle\alpha^*\rangle\phi$. Thus, there is an α-transition to a state μ such that $(\nu,\mu)\in\rho_{I,\eta}(\alpha)$ with $I,\eta,\mu\models\langle\alpha^*\rangle\phi$. Consequently, there is an α^*-transition to a state ω with $(\mu,\omega)\in\rho_{I,\eta}(\alpha^*)$ and $I,\eta,\omega\models\phi$. Obviously, every α-transition also is an α^*-transition, because repetitions may choose to repeat only once. In particular, by chaining the α-transition $(\nu,\mu)\in\rho_{I,\eta}(\alpha)\subset\rho_{I,\eta}(\alpha^*)$ with the α^*-transition $(\mu,\omega)\in\rho_{I,\eta}(\alpha^*)$, we obtain a longer α^*-transition $(\nu,\omega)\in\rho_{I,\eta}(\alpha^*)$ by the transition semantics in Definition 2.7. Hence, in either case, we conclude $I,\eta,\nu\models\langle\alpha^*\rangle\phi$. The converse direction can be proven accordingly to show equivalence and the local soundness of the dual rule $[{}^{*n}]$.

$\langle?\rangle$ Rule $\langle?\rangle$ is locally sound. Assume that the premise holds in I,η,ν, i.e., $I,\eta,\nu\models\chi\wedge\phi$. We have to show that the conclusion holds in I,η,ν, i.e., $I,\eta,\nu\models\langle?\chi\rangle\phi$. We have to show that there is a transition along $?\chi$ to a state where ϕ holds. By the semantics in Definition 2.7, there is only a transition along hybrid program $?\chi$ if $I,\eta,\nu\models\chi$ and the state is not changed by $?\chi$ transitions. Now the premise implies $I,\eta,\nu\models\chi$ and $I,\eta,\nu\models\phi$, which, together, imply $I,\eta,\nu\models\langle?\chi\rangle\phi$. Since this is the only case where $?\chi$ can make a transition to a state satisfying ϕ, it shows equivalence. Local soundness of the dual rule $[?]$ follows from this.

$\langle'\rangle$ The rule $\langle'\rangle$ is locally sound. Let $y_1,\ldots,y_n$ be a solution for the differential equation system $x_1'=\theta_1,\ldots,x_n'=\theta_n$ with symbolic initial values $x_1,\ldots,x_n$. Let further $\langle\mathscr{S}_t\rangle$ be the jump set $\langle x_1:=y_1(t),\ldots,x_n:=y_n(t)\rangle$. Assume I,η,ν are such that the premise is true: $I,\eta,\nu\models\exists t{\geq}0\,(\bar{\chi}\wedge\langle\mathscr{S}_t\rangle\phi)$ with $\forall 0{\leq}\tilde{t}{\leq}t\,\langle\mathscr{S}_{\tilde{t}}\rangle\chi$ abbreviated as $\bar{\chi}$. For any $\zeta\in\mathbb{R}$, we denote by η^{ζ}_{t} the assignment that agrees with η except that it assigns ζ to t. Then, by as-

sumption, there is a real value $r \geq 0$ such that $I, \eta^r_t, \nu \models \bar{\chi} \wedge \langle \mathscr{S}_t \rangle \phi$. Abbreviate $x'_1 = \theta_1, .., x'_n = \theta_n \,\&\, \chi$ by $\mathscr{D}$. We have to show that $I, \eta, \nu \models \langle \mathscr{D} \rangle \phi$. Equivalently, by Lemma 2.6, we show $I, \eta^r_t, \nu \models \langle \mathscr{D} \rangle \phi$, because t is a fresh variable that does not occur in $\mathscr{D}$ or ϕ. Let function $f : [0, r] \to \mathrm{Sta}(\Sigma)$ be defined such that $(\nu, f(\zeta)) \in \rho_{I,\eta^\zeta_t}(\mathscr{S}_t)$ for all $\zeta \in [0, r]$. By premise, $f(0)$ is identical to ν and ϕ holds at $f(r)$. Thus it only remains to be shown that f respects the constraints of Definition 2.7 for $\mathscr{D}$. In fact, f obeys the continuity and differentiability properties of Definition 2.7 by the corresponding properties of the y_i. Moreover, $val_{I,\eta^r_t}(f(\zeta), x_i) = val_{I,\eta^r_t}(\nu, y_i(t))$ has a derivative of value $val_{I,\eta^r_t}(f(\zeta), \theta_i)$, because y_i is a solution of the differential equation $x'_i = \theta_i$ with corresponding initial value $\nu(x_i)$. Further, it can be shown that the evolution domain χ is respected along f as follows: By premise, $I, \eta^r_t, \nu \models \bar{\chi}$ holds for the initial state ν; thus $val_{I,\eta^r_t}(f(\zeta), \chi) = \mathit{true}$ for all $\zeta \in [0, r]$. Combining these results, we can conclude that f is a witness for $I, \eta, \nu \models \langle \mathscr{D} \rangle \phi$. The converse direction can be shown accordingly to prove the dual rule [′] using Lemma 2.1.

∀r The proof is a sequent calculus adaptation of that in [147]. By contraposition, assume that there are I, ν such that for all η it is the case that $I, \eta, \nu \not\models \forall x \phi(x)$; hence $I, \eta, \nu \models \exists x \neg\phi(x)$. We construct an interpretation I' that agrees with I except for the new function symbol s. Let $b_1, \ldots, b_n \in \mathbb{R}$ be arbitrary elements and let η^b assign b_i to the respective X_i for $1 \leq i \leq n$. As $I, \eta, \nu \models \exists x \neg\phi(x)$ holds for all η, we pick a witness d for $I, \eta^b, \nu \models \exists x \neg\phi(x)$ and choose $I'(s)(b_1, \ldots, b_n) = d$. For this interpretation I' and state ν we have $I', \eta, \nu \not\models \phi(s(X_1, \ldots, X_n))$ for all assignments η by Lemma 2.6, as $X_1, .., X_n$ are all free variables determining the truth-value of $\phi(s(X_1, \ldots, X_n))$. To see that the contexts Γ, Δ of Definition 2.10 can be added to instantiate this rule, consider the following. Since s is new and does not occur in the context Γ, Δ, the latter do not change their truth-value by passing from I to I'. Likewise, s is rigid so that it does not change its value by adding jump prefix $\langle \mathscr{J} \rangle$ which concludes the proof. The proof of ∃l is dual.

i∀ i∀ is locally sound. Assume that $I, \eta, \nu \models \mathrm{QE}(\forall X\, (\Phi(X) \vdash \Psi(X)))$. Since QE yields an equivalence, we can conclude $I, \eta, \nu \models \forall X\, (\Phi(X) \vdash \Psi(X))$. Then if the antecedent of the conclusion is true, $I, \eta, \nu \models \Phi(s(X_1, \ldots, X_n))$, we conclude $I, \eta, \nu \models \Psi(s(X_1, \ldots, X_n))$ by choosing $val_{I,\eta}(\nu, s(X_1, \ldots, X_n))$ for X in the premise. By admissibility of substitutions, variables $X_1, \ldots, X_n$ are free at all occurrences of $s(X_1, \ldots, X_n)$, and hence their value is the same in all occurrences.

∃r ∃r is locally sound by a simplified version of the proof in [147]. For any I, η, ν with $I, \eta, \nu \models \phi(X)$ we can conclude $I, \eta, \nu \models \exists x \phi(x)$ according to the witness $\eta(X)$. The proof of ∀l is dual.

i∃ For any I, ν let η be such that $I, \eta, \nu \models \mathrm{QE}(\exists X \bigwedge_i (\Phi_i \vdash \Psi_i))$. Again, this implies $I, \eta, \nu \models \exists X \bigwedge_i (\Phi_i \vdash \Psi_i)$, because quantifier elimination yields an equivalence. We pick a witness $d \in \mathbb{R}$ for this existential quantifier. As X does not occur anywhere else in the proof, it disappears from all

open premises of the proof by applying i$\exists$. Hence, by the Coincidence Lemma 2.6, the value of X does not change the truth-value of the premise of i$\exists$. Consequently, η can be extended to η' by changing the interpretation of X to the witness d such that $I, \eta', \nu \models \bigwedge_i (\Phi_i \vdash \Psi_i)$. Thus, η' extends I, η, ν to a simultaneous model of all conclusions.

$\langle\rangle gen$ Rules $[]gen$–con are locally sound by a variation of the usual proofs [149] using universal closures for local soundness. $[]gen$,$\langle\rangle gen$ are simple refinements of Lemma 2.6 using the fact that the universal closure $\forall^\alpha$ comprises all variables that change in α. Let $I, \eta, \nu \models \langle\alpha\rangle\phi$, i.e., let $(\nu, \nu') \in \rho_{I,\eta}(\alpha)$ with $I, \eta, \nu' \models \phi$. As α can only change its bound variables, which are quantified universally in the universal closure $\forall^\alpha$, the premise implies $I, \eta, \nu' \models \phi \to \psi$; thus $I, \eta, \nu' \models \psi$ and $I, \eta, \nu \models \langle\alpha\rangle\psi$. The proof of $[]gen$ is similar.

ind For any I, η, ν with $I, \eta, \nu \models \forall^\alpha(\phi \to [\alpha]\phi)$, we know $I, \eta, \nu' \models \phi \to [\alpha]\phi$ for all ν' with $(\nu, \nu') \in \rho_{I,\eta}(\alpha)$. As these share the same η, we can further conclude $I, \eta, \nu \models \phi \to [\alpha^*]\phi$ by induction along the series of states ν' reached from ν by repeating α. The universal closure is necessary as, otherwise, the premise may yield different η in different states ν'.

con Assume that the antecedent and premise hold in I, η, ν. By premise, we have $I, \eta[v \mapsto d], \nu' \models v > 0 \wedge \varphi(v) \to \langle\alpha\rangle\varphi(v-1)$ for all $d \in \mathbb{R}$ and all states ν' that are reachable by α^* from ν, because $\forall^\alpha$ comprises all variables that are bound by α, which are the same as those bound by α^*. By antecedent, there is a $d \in \mathbb{R}$ such that $I, \eta[v \mapsto d], \nu \models \varphi(v)$. Now, the proof is a well-founded induction on d. If $d \leq 0$, we directly have $I, \eta, \nu \models \langle\alpha^*\rangle\exists v{\leq}0\, \varphi(v)$ for zero repetitions. Otherwise, if $d > 0$, we have, by premise, that

$$I, \eta[v \mapsto d], \nu \models v > 0 \wedge \varphi(v) \to \langle\alpha\rangle\varphi(v-1).$$

As $v > 0 \wedge \varphi(v)$ holds true at $I, \eta[v \mapsto d], \nu$, we have for some ν' with $(\nu, \nu') \in \rho_{I,\eta[v\mapsto d]}(\alpha)$ that $I, \eta[v \mapsto d], \nu' \models \varphi(v-1)$. In particular, we can conclude that $I, \eta[v \mapsto d-1], \nu' \models \varphi(v)$ satisfies the induction hypothesis for a smaller d and a reachable ν', because $(\nu, \nu') \in \rho_{I,\eta}(\alpha)$ as v does not occur in α. The induction is well-founded, because d decreases by 1 up to the base case $d \leq 0$. □

With this soundness theorem, we now know that everything we prove in the $\mathsf{d}\mathscr{L}$ calculus accurately reflects reality, because the syntactic proofs built with Fig. 2.11 fit to the semantics defined in Sect. 2.3.

2.7 Completeness

In this section, we prove that the $\mathsf{d}\mathscr{L}$ calculus is a sound and complete axiomatisation of the transition behaviour of hybrid systems relative to differential equations.

With Soundness Theorem 2.1, we have shown that all provable formulas are valid. So we know that will never prove something that does not even hold (is not valid). The converse question is whether all valid formulas are also provable, i.e., whether we will always be able to prove all formulas that are "true" (valid). Have we just been lucky with the successful proofs that we managed to show so far? Or is there a deeper reason for which we can know that, in principle, we could also find proofs for all other valid formulas?

2.7.1 Incompleteness

Theorem 2.1 shows that all provable closed dℒ formulas are valid. The converse question is whether the dℒ calculus is *complete*, i.e., all valid dℒ formulas are provable. Combining completeness for first-order logic [147] and decidability of real arithmetic [81], it is easy to see that our calculus is complete for closed formulas of first-order real arithmetic by chaining the quantifier rules ∀r,∃l,∃r,∀l with the respective inverse rules i∀,i∃, using propositional rules as needed to unfold the propositional structure. In the presence of modalities, however, dℒ is not axiomatisable and, unlike its basis of first-order *real* arithmetic, dℒ is undecidable. Both unbounded repetition in the discrete fragment and unbounded evolution in the continuous fragment cause incompleteness. Beyond hybrid dynamics, where reachability is known to be undecidable [156], we show that even the purely discrete and purely continuous parts of dℒ are not effectively axiomatisable. Hence, valid dℒ formulas are not always provable.

Theorem 2.2 (Incompleteness of dℒ). *Both the discrete fragment and the continuous fragment of* dℒ *are* not effectively axiomatisable*, i.e., they have no sound and complete effective calculus, because natural numbers are definable in both fragments.*

Proof. We prove that natural numbers are definable among the real numbers of dℒ interpretations in both fragments. Then these fragments extend first-order *integer* arithmetic such that the incompleteness theorem of Gödel [137] applies. Gödel's incompleteness theorem shows that no logic extending first-order integer arithmetic can have a sound and complete effective calculus. Natural numbers are definable in the discrete fragment without continuous evolutions using repetitive additions:

$$nat(n) \;\leftrightarrow\; \langle x:=0;(x:=x+1)^*\rangle\, x=n.$$

In the continuous fragment, an isomorphic copy of the natural numbers is definable using linear differential equations:

$$nat(n) \;\leftrightarrow\; \exists s\,\exists c\,\exists \tau\,(s=0\wedge c=1\wedge \tau=0\wedge\langle s'=c,c'=-s,\tau'=1\rangle(s=0\wedge\tau=n)).$$

These differential equations characterise sin and cos as unique solutions for s and c,

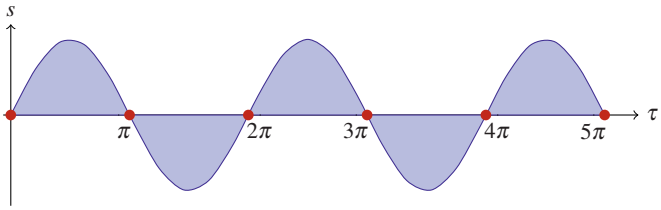


Fig. 2.21 Characterisation of $\mathbb{N}$ as zeros of solutions of differential equations

respectively. Their zeros, as detected by τ, correspond to an isomorphic copy of natural numbers, scaled by π, i.e., $nat(n)$ holds iff n is of the form $k\pi$ for a $k \in \mathbb{N}$; see Fig. 2.21. The initial values for s and c prevent the trivial solution identical to 0. □

In this context, note that hybrid programs contain a computationally complete sublanguage and that reachability of hybrid systems is undecidable [156].

2.7.2 Relative Completeness

The standard approach for showing adequacy of a calculus when its logic is not effectively axiomatisable is to analyse the deductive power of the calculus relative to a base logic or to an ineffective oracle rule for the base logic [87, 148, 149]. In calculi for discrete programs, completeness is proven relative to the handling of data [87, 148, 149]. For hybrid systems, this is inadequate: By Theorem 2.2, no sound calculus for $\mathsf{d}\mathscr{L}$ can be complete relative to its data (the reals), because its basis, first-order real arithmetic, is a perfectly decidable and axiomatisable theory [288]. If the $\mathsf{d}\mathscr{L}$ calculus itself would be complete relative to the data of first-order real arithmetic, then, since this is a decidable logic, the $\mathsf{d}\mathscr{L}$ calculus would be complete altogether, which would contradict Theorem 2.2. Thus, we need a different basis for a relative completeness argument. Unlike in classical discrete programs, the data is not where the complexity comes from. In hybrid dynamical systems, the complexity truly originates from the actual dynamics.

According to Theorem 2.2, both continuous evolutions and repetitive discrete transitions, as well as their interaction, cause non-axiomatisability of $\mathsf{d}\mathscr{L}$. Discrete transitions and repetition do not supersede the complexity of continuous transitions. Even relative to an oracle for handling properties of discrete jumps and repetition, the $\mathsf{d}\mathscr{L}$ calculus is not complete, simply because not all differential equations have solutions that are definable in first-order arithmetic so that rule $[']$ can be used. For instance, the solutions of $s' = c, c' = -s$ are trigonometric functions (like sin and cos), which are not first-order definable. The question is whether the converse is true, i.e., whether hybrid programs can be verified given that all required differential equations can be handled.

To calibrate the deductive power of the dℒ calculus in light of its inherent incompleteness, we analyse the quotient of reasoning about hybrid systems modulo differential equation handling. Using generalisations of the usual notions of relative completeness for discrete systems [87, 148, 149] to the hybrid case, we show that the dℒ calculus completely axiomatises dℒ relative to one single additional axiom about valid first-order properties of differential equations. Essentially, we drop the effectiveness requirement for one oracle axiom and show that the resulting dℒ calculus is sound and complete. We thus show that the dℒ calculus would be complete if only we had a complete replacement for $[']$,$\langle' \rangle$. Although repetitions and interactions of hybrid programs are more involved than purely continuous systems, this results emphasises the importance of studying approximations of this continuous oracle for the analysis of hybrid systems, as we do in Chap. 3.

As a basis, we define FOD as the *first-order logic of differential equations*, i.e., first-order real arithmetic augmented with formulas expressing properties of differential equations, that is, dℒ formulas of the form $[x_1' = \theta_1, \ldots, x_n' = \theta_n]F$ with a first-order formula F. Dually, the diamond formula $\langle x_1' = \theta_1, \ldots, x_n' = \theta_n \rangle F$ is expressible as $\neg[x_1' = \theta_1, \ldots, x_n' = \theta_n]\neg F$.

Theorem 2.3 (Relative completeness of dℒ). *The* dℒ *calculus is* complete relative to *FOD, i.e., every valid* dℒ *formula can be derived from FOD tautologies.*

Proof (Outline). The (constructive) proof, which, in full, is contained in the remainder of this section, adapts the techniques of Cook [87] and Harel [148, 149] to the hybrid case. The decisive step is to show that every valid property of a repetition α^* can be proven by rules *ind* or *con*, respectively, with a sufficiently strong invariant or variant that is expressible in dℒ. For this, we show that dℒ formulas can be expressed equivalently in FOD, and that valid dℒ formulas can be derived from corresponding FOD axioms in the dℒ calculus. In turn, the crucial step is to construct a finite FOD formula that characterises the effect of unboundedly many repetitive hybrid transitions and just uses finitely many real variables. □

This main result completely aligns hybrid and continuous verification proof-theoretically. It gives a formal justification that reasoning about hybrid systems is possible to *exactly* the same extent to which it is possible to show properties of solutions of differential equations. Theorem 2.3 shows that superpositions or combinations of discrete jumps, continuous evolutions, and repetitions of hybrid processes can be verified whenever corresponding (intermediate) properties of differential equations are provable. Moreover, in a proof-theoretical sense, our calculus completely lifts all verification techniques for dynamical systems to hybrid systems perfectly. Summarising Theorems 2.1 and 2.3:

The dℒ calculus axiomatises the transition behaviour of hybrid systems completely relative to the handling of differential equations!

In the following subsections, we present a fully constructive proof of Theorem 2.3, which generalises the techniques of Harel [148, 149] and Cook [87] to

the hybrid case. It shows that for every valid $\mathsf{d}\mathscr{L}$ formula, there is a finite set of valid FOD formulas from which it can be derived in the $\mathsf{d}\mathscr{L}$ calculus. Recall the proof outline of Theorem 2.3 for a road map of the proof.

Natural numbers are definable in FOD by Theorem 2.2. In this section, we abbreviate quantifiers over natural numbers, e.g., $\forall x\,(nat(x) \to \phi)$ by $\forall x{:}\mathbb{N}\ \phi$ and $\exists x\,(nat(x) \wedge \phi)$ by $\exists x{:}\mathbb{N}\ \phi$. Likewise, we abbreviate quantifiers over integers, e.g., $\forall x\,((nat(x) \vee nat(-x)) \to \phi)$ by $\forall x{:}\mathbb{Z}\ \phi$.

2.7.3 Characterising Real Gödel Encodings

As the central device for constructing a FOD formula that captures the effect of unboundedly many repetitive hybrid transitions and just uses finitely many real variables, we prove that a real version of Gödel encoding is definable in FOD. That is, we give a FOD formula that reversibly packs finite sequences of real values into a single real number.

Observe that a single differential equation system is *not* sufficient for defining these pairing functions as their solutions are differentiable, and yet, as a consequence of Morayne's theorem [213], there is no differentiable surjection $\mathbb{R} \to \mathbb{R}^2$, nor to any part of $\mathbb{R}^2$ of positive measure. We show that real sequences can be encoded nevertheless by chaining the effects of solutions of multiple differential equations and quantifiers.

Lemma 2.7 ($\mathbb{R}$-Gödel encoding). *The formula $at(Z,n,j,z)$, which holds iff Z is a real number that represents a Gödel encoding of a sequence of n real numbers with real value z at position j (for $1 \le j \le m$), is definable in FOD. For a formula $\phi(z)$ we abbreviate $\exists z\,(\mathrm{at}(Z,n,j,z) \wedge \phi(z))$ by $\phi(Z_j^{(n)})$.*

$$\sum_{i=0}^{\infty} \frac{a_i}{2^i} = a_0.a_1a_2\ldots \qquad \sum_{i=0}^{\infty} \frac{b_i}{2^i} = b_0.b_1b_2\ldots \qquad \Rightarrow \qquad \sum_{i=0}^{\infty} \left(\frac{a_i}{2^{2i-1}} + \frac{b_i}{2^{2i}} \right) = a_0b_0.a_1b_1a_2b_2\ldots$$

Fig. 2.22 Fractional encoding principle of $\mathbb{R}$-Gödel encoding by bit interleaving

Proof. The basic idea of the $\mathbb{R}$-Gödel encoding is to interleave the bits of real numbers as depicted in Fig. 2.22 (for a pairing of $n = 2$ numbers a and b). For defining $\mathrm{at}(Z,n,j,z)$, we use several auxiliary functions to improve readability; see Fig. 2.23. Note that these definitions need no recursion. Hence, as in the notation $\phi(Z_j^{(n)})$, we can consider occurrences of the function symbols as syntactic abbreviations for quantified variables satisfying the respective definitions.

$$\begin{aligned}
\mathrm{at}(Z,n,j,z) &\leftrightarrow \forall i{:}\mathbb{Z}\ \mathrm{digit}(z,i) = \mathrm{digit}(Z,n(i-1)+j) \wedge nat(n) \wedge nat(j) \wedge n > 0\\
\mathrm{digit}(a,i) &= \mathrm{intpart}(2\,\mathrm{frac}(2^{i-1}a))\\
\mathrm{intpart}(a) &= a - \mathrm{frac}(a)\\
\mathrm{frac}(a) = z &\leftrightarrow \exists i{:}\mathbb{Z}\ z = a - i \wedge -1 < z \wedge z < 1 \wedge az \geq 0\\
2^i = z &\leftrightarrow i \geq 0 \wedge \exists x \exists t\,(x = 1 \wedge t = 0 \wedge \langle x' = x\ln 2, t' = 1\rangle (t = i \wedge x = z))\\
&\quad \vee\ i < 0 \wedge \exists x \exists t\,(x = 1 \wedge t = 0 \wedge \langle x' = -x\ln 2, t' = -1\rangle (t = i \wedge x = z))\\
\ln 2 = z &\leftrightarrow \exists x \exists t\,(x = 1 \wedge t = 0 \wedge \langle x' = x, t' = 1\rangle (x = 2 \wedge t = z))
\end{aligned}$$

Fig. 2.23 FOD definition characterising Gödel encoding of $\mathbb{R}$-sequences in one real number

The function symbol $\mathrm{digit}(a,i)$ gives the ith bit of $a \in \mathbb{R}$ when represented with basis 2. For $i > 0$, $\mathrm{digit}(a,i)$ yields fractional bits, and, for $i \leq 0$, it yields bits of the integer part. For instance, $\mathrm{digit}(a,1)$ yields the first fractional bit, $\mathrm{digit}(a,0)$ is the least-significant bit of the integer part of a. The function $\mathrm{intpart}(a)$ represents the integer part of $a \in \mathbb{R}$. The function $\mathrm{frac}(a)$ represents the fractional part of $a \in \mathbb{R}$, which drops all integer bits. The last constraint in its definition implies that $\mathrm{frac}(a)$ keeps the sign of a (or 0). Consequently, $\mathrm{intpart}(a)$ and $\mathrm{digit}(a,i)$ also keep the sign of a (or 0). Exponentiation 2^i is definable using differential equations, using an auxiliary characterisation of the natural logarithm $\ln 2$. The definition of 2^i splits into the case of exponential growth when $i \geq 0$ and a symmetric case of exponential decay when $i < 0$. □

2.7.4 Expressibility and Rendition of Hybrid Program Semantics

In order to show that dL is sufficiently expressive to state the invariants and variants that are needed for proving valid statements about loops with rules *ind* and *con*, we prove an expressibility result. We give a constructive proof that the state transition relation of hybrid programs is definable in FOD, i.e., there is a FOD formula $\mathscr{S}_\alpha(\vec{x},\vec{v})$ characterising the state transitions of hybrid program α from the state characterised by the vector $\vec{x}$ of variables to the state characterised by vector $\vec{v}$.

For this, we need to characterise hybrid processes equivalently by differential equations in FOD. Observe that the existence of such characterisations does *not* follow from results embedding Turing machines into differential equations [57, 140], because, unlike Turing machines, hybrid processes are not restricted to discrete values on a grid (such as $\mathbb{N}^k$) but work with continuous real values. Furthermore, Turing machines only have repetitions of discrete transitions on discrete data (e.g., $\mathbb{N}$). For hybrid programs, in contrast, we have to characterise repetitive interactions of discrete and continuous transitions in continuous space (some $\mathbb{R}^k$).

Lemma 2.8 (Hybrid program rendition). *For every hybrid program α with variables among $\vec{x} = x_1,\ldots,x_k$, there is a FOD formula $\mathscr{S}_\alpha(\vec{x},\vec{v})$ with variables among the $2k$ distinct variables $\vec{x} = x_1,\ldots,x_k$ and $\vec{v} = v_1,\ldots,v_k$ such that*

$$\vDash \mathscr{S}_\alpha(\vec{x},\vec{v}) \leftrightarrow \langle\alpha\rangle \vec{x} = \vec{v}$$

or, equivalently, for every I, η, ν,

$$I, \eta, \nu \models \mathscr{S}_\alpha(\vec{x}, \vec{v}) \text{ iff } (\nu, \nu[\vec{x} \mapsto val_{I,\eta}(\nu, \vec{v})]) \in \rho_{I,\eta}(\alpha).$$

$$\begin{aligned}
\mathscr{S}_{x_1:=\theta_1,\ldots,x_k:=\theta_k}(\vec{x},\vec{v}) &\equiv \bigwedge_{i=1}^{k}(v_i = \theta_i)\\
\mathscr{S}_{x_1'=\theta_1,\ldots,x_k'=\theta_k}(\vec{x},\vec{v}) &\equiv \langle x_1' = \theta_1, .., x_k' = \theta_k\rangle \vec{v} = \vec{x}\\
\mathscr{S}_{x_1'=\theta_1,\ldots,x_k'=\theta_k \,\&\, \chi}(\vec{x},\vec{v}) &\equiv \exists t \big(t = 0 \wedge \langle x_1' = \theta_1, .., x_k' = \theta_k, t' = 1\rangle \big(\vec{v} = \vec{x}\\
&\qquad \wedge [x_1' = -\theta_1, .., x_k' = -\theta_k, t' = -1](t \geq 0 \rightarrow \chi)\big)\big)\\
\mathscr{S}_{?\chi}(\vec{x},\vec{v}) &\equiv \vec{v} = \vec{x} \wedge \chi\\
\mathscr{S}_{\beta\cup\gamma}(\vec{x},\vec{v}) &\equiv \mathscr{S}_\beta(\vec{x},\vec{v}) \vee \mathscr{S}_\gamma(\vec{x},\vec{v})\\
\mathscr{S}_{\beta;\gamma}(\vec{x},\vec{v}) &\equiv \exists \vec{z}(\mathscr{S}_\beta(\vec{x},\vec{z}) \wedge \mathscr{S}_\gamma(\vec{z},\vec{v}))\\
\mathscr{S}_{\beta^*}(\vec{x},\vec{v}) &\equiv \exists Z \exists n{:}\mathbb{N}\, \big(Z_1^{(n)} = \vec{x} \wedge Z_n^{(n)} = \vec{v}\\
&\qquad \wedge \forall i{:}\mathbb{N}\, (1 \leq i < n \rightarrow \mathscr{S}_\beta(Z_i^{(n)}, Z_{i+1}^{(n)}))\big)
\end{aligned}$$

Fig. 2.24 Explicit rendition of hybrid program transition semantics in FOD

Proof. By Lemma 2.6, interpretations of the vectors $\vec{x}$ and $\vec{v}$ characterise the input and output states, respectively, as far as α is concerned. These vectors are finite because α is finite. Vectorial equalities like $\vec{x} = \vec{v}$ or quantifiers $\exists \vec{v}$ are to be understood componentwise. The program rendition is defined inductively in Fig. 2.24. To simplify the notation, we assume that all variables $x_1, \ldots, x_k$ are affected in discrete jumps and differential equations by adding vacuous $x_i := x_i$, or $x_i' = 0$ if x_i does not change in the respective statement.

Differential equations give FOD formulas; no further reduction is needed. Evolution along differential equations with evolution domain restrictions is definable by following the unique flow (Lemma 2.1) backwards. Continuous evolution is reversible, i.e., the transitions of $x_i' = -\theta$ are inverse to those of $x_i' = \theta$. Consequently, with an auxiliary variable t, all evolutions of $[x_1' = -\theta_1, .., x_k' = -\theta_k, t' = -1]$ follow the same flow as $\langle x_1' = \theta_1, .., x_k' = \theta_k, t' = 1\rangle$, but backwards. By also reversing clock t, we ensure that, along the reverse flow, χ has been true at all times (because of the box modality) until starting time $t = 0$; see Fig. 2.25.

To show reversibility, let $(\nu, \omega) \in \rho_{I,\eta}(x_1' = \theta_1, .., x_k' = \theta_k)$, that is, let $f : [0, r] \to \mathrm{Sta}(\Sigma)$ be a solution of $x_1' = \theta_1, .., x_k' = \theta_k$ starting in state ν and ending in ω. Then $g : [0, r] \to \mathrm{Sta}(\Sigma)$, defined as $g(\zeta) = f(r - \zeta)$, starts in ω and ends in ν. Thus, it only remains to show that g is a solution of $x_1' = -\theta_1, .., x_k' = -\theta_k$, which can be seen for $1 \leq i \leq k$ as follows:

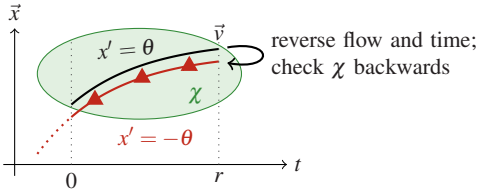


Fig. 2.25 Evolution domain checks along backwards flow over time t

$$\frac{\mathrm{d}g(t)(x_i)}{\mathrm{d}t}(\zeta) = \frac{\mathrm{d}f(r-t)(x_i)}{\mathrm{d}t}(\zeta) = \frac{\mathrm{d}f(u)(x_i)}{\mathrm{d}u}\frac{\mathrm{d}(r-t)}{\mathrm{d}t}(\zeta) = -\frac{\mathrm{d}f(u)(x_i)}{\mathrm{d}u}(\zeta)$$
$$= -val_{I,\eta}(f(\zeta),\theta_i) = val_{I,\eta}(f(\zeta),-\theta_i).$$

Unlike all other cases, case $\mathscr{S}_{x_1'=\theta_1,\ldots,x_k'=\theta_k\,\&\,\chi}(\vec{x},\vec{v})$ in Fig. 2.24 uses nested FOD modalities. Nested modalities can be avoided in $\mathscr{S}_{\alpha}(\vec{x},\vec{v})$ using an equivalent FOD formula without them; see Fig. 2.25:

$$\exists t\,\exists r\,\big(t=0\wedge\langle x_1'=\theta_1,\ldots,x_k'=\theta_k,t'=1\rangle(\vec{v}=\vec{x}\wedge r=t)\wedge$$
$$\forall\vec{x}\forall t\,(\vec{x}=\vec{v}\wedge t=r\rightarrow[x_1'=-\theta_1,\ldots,x_k'=-\theta_k,t'=-1](t\geq 0\rightarrow\chi))\big).$$

With a finite formula, the characterisation of repetition $\mathscr{S}_{\beta^*}(\vec{x},\vec{v})$ in FOD needs to capture arbitrarily long sequences of intermediate real-valued states and the correct transition between successive states of such a sequence. To achieve this with first-order quantifiers, we use the real Gödel encoding from Lemma 2.7 in Fig. 2.24 to map unbounded sequences of real-valued states reversibly to a single real number Z, which can be quantified over in first-order logic. □

Using the program rendition from Lemma 2.8 to characterise modalities, we prove that every dL formula can be expressed equivalently in FOD by structural induction.

Lemma 2.9 (dL Expressibility). *Logic* dL *is* expressible *in FOD: for all* dL *formulas* $\phi\in\mathrm{Fml}(\Sigma,V)$ *there is a FOD formula* $\phi^{\#}\in\mathrm{Fml}_{FOD}(\Sigma,V)$ *that is equivalent, i.e.,* $\vDash\phi\leftrightarrow\phi^{\#}$. *The converse holds trivially.*

Proof. The proof follows an induction on the structure of formula ϕ for which it is imperative to find an equivalent $\phi^{\#}$ in FOD. Observe that the construction of $\phi^{\#}$ from ϕ is effective.

0. If ϕ is a first-order formula, then $\phi^{\#}:=\phi$ already is a FOD formula such that nothing has to be shown.
1. If ϕ is of the form $\varphi\vee\psi$, then by the induction hypothesis there are FOD formulas $\varphi^{\#},\psi^{\#}$ such that $\vDash\varphi\leftrightarrow\varphi^{\#}$ and $\vDash\psi\leftrightarrow\psi^{\#}$, from which we can conclude by congruence that $\vDash(\varphi\vee\psi)\leftrightarrow(\varphi^{\#}\vee\psi^{\#})$, giving $\vDash\phi\leftrightarrow\phi^{\#}$ by choosing $\varphi^{\#}\vee\psi^{\#}$ for $\phi^{\#}$. Similar reasoning addresses the other propositional connectives or quantifiers.
2. The case where ϕ is of the form $\langle\alpha\rangle\psi$ is a consequence of the characterisation of the semantics of hybrid programs in FOD. The expressibility conjecture holds

by the induction hypothesis using the equivalence of explicit hybrid program renditions from Lemma 2.8:

$$\vDash \quad \langle\alpha\rangle\psi \leftrightarrow \exists\vec{v}\,(\mathscr{S}_\alpha(\vec{x},\vec{v}) \wedge \psi^{\#\vec{v}}_{\vec{x}}).$$

3. The case where ϕ is $[\alpha]\psi$ is again a consequence of Lemma 2.8:

$$\vDash \quad [\alpha]\psi \leftrightarrow \forall\vec{v}\,(\mathscr{S}_\alpha(\vec{x},\vec{v}) \rightarrow \psi^{\#\vec{v}}_{\vec{x}})$$

□

The above proofs directly carry over to rich test $\mathsf{d}\mathscr{L}$, i.e., the logic where $\mathsf{d}\mathscr{L}$ formulas are allowed in tests $?\chi$ of hybrid programs and evolution domain restrictions χ of differential equations, when using $\chi^\#$ in place of χ in Fig. 2.24. Accordingly, nested modalities can be avoided in FOD by using the following formula for $\mathscr{S}_{x'_1=\theta_1,\ldots,x'_k=\theta_k\,\&\,\chi}(\vec{x},\vec{v})$:

$$\begin{aligned}\exists t\,\exists r\,\big(&t=0\wedge\langle x'_1=\theta_1,..,x'_k=\theta_k,t'=1\rangle(\vec{v}=\vec{x}\wedge r=t)\wedge\\ &\forall\vec{z}\,\big(\exists\vec{x}\exists t\,(\vec{x}=\vec{v}\wedge t=r\wedge\langle x'_1=-\theta_1,..,x'_k=-\theta_k,t'=-1\rangle(t\geq 0\wedge\vec{z}=\vec{x}))\\ &\quad\rightarrow\chi^{\#\vec{z}}_{\vec{x}}\big)\big).\end{aligned}$$

2.7.5 Relative Completeness of First-Order Assertions

As special cases of Theorem 2.3, we first prove relative completeness for first-order assertions about hybrid programs. These first-order cases constitute the basis for the general completeness proof for arbitrary formulas of differential dynamic logic.

In the following relative completeness proofs, we use the notation $\vdash_{\mathscr{D}}\phi$ to indicate that a $\mathsf{d}\mathscr{L}$ formula ϕ is derivable (Definition 2.11) from a set of FOD tautologies, which is equivalent to saying that ϕ is derivable in the $\mathsf{d}\mathscr{L}$ calculus augmented with a single *oracle axiom* $\mathscr{D}$ that gives all valid FOD instances. Likewise, we use the notation $\Gamma\vdash_{\mathscr{D}}\Delta$ to indicate that the sequent $\Gamma\vdash\Delta$ is derivable from $\mathscr{D}$.

For the completeness proof, we use several simplifications. For uniform proofs, we assume formulas to use a simplified vocabulary. A formula ϕ is valid iff it is true in *all* I,η,ν. In particular, we can assume valid ϕ to use Skolem constants (or state variables) instead of free logical variables. Existential quantifiers can be represented as modalities: $\exists x\,\phi \equiv \langle x'=1\rangle\phi\vee\langle x'=-1\rangle\phi$. For simplicity, we use cut (*cut*) and weakening to glue together subproofs propositionally. Weakening (i.e., from $\phi\vdash\psi$ infer $\phi_1,\phi\vdash\psi,\psi_1$) can be emulated using contexts Γ,Δ from Definition 2.10, and we use it implicitly together with rule *cut* in the following. Derivability of sequents and derivability of corresponding formulas are equivalent by the following lemma.

Lemma 2.10 (Derivability of sequents). $\vdash_{\mathscr{D}}\phi\rightarrow\psi$ *iff* $\phi\vdash_{\mathscr{D}}\psi$.

Proof. When we consider sequents as abbreviations for formulas, both sides are identical. Otherwise, let $\vdash_{\mathscr{D}} \phi \to \psi$ be derivable from $\mathscr{D}$. Using *cut* (and weakening) with $\phi \to \psi$, this derivation can be extended to one of $\phi \vdash_{\mathscr{D}} \psi$:

$$\dfrac{\dfrac{*}{\phi \vdash \phi \to \psi, \psi} \quad \dfrac{\dfrac{*}{\phi \vdash \phi, \psi}\,ax \qquad \dfrac{*}{\psi, \phi \vdash \psi}\,ax}{\phi, \phi \to \psi \vdash \psi}\,{\to}\mathrm{l}}{\phi \vdash \psi}\,cut$$

The converse direction is by an application of →r. □

Lemma 2.11 (Generalisation). *If $\vdash_{\mathscr{D}} \phi$ is provable without free logical variables, then so are $\vdash_{\mathscr{D}} \forall x\, \phi$ and $\vdash_{\mathscr{D}} \langle x_1 := \theta_1, \ldots x_n := \theta_n \rangle \phi$.*

Proof. For the second conjecture, let $\langle \mathscr{A} \rangle$ abbreviate $\langle x_1 := \theta_1, \ldots x_n := \theta_n \rangle$. We prefix each formula in the proof of ϕ with $\langle \mathscr{A} \rangle$ and show that this gives a proof of $\langle \mathscr{A} \rangle \phi$. i∃ is not needed in the proof due to the absence of free logical variables. As an intermediate step, we first show that prefixing with $\langle \mathscr{A} \rangle$ gives an (extended) proof with rule applications generalised to allow for nested jump prefixes $\langle \mathscr{A} \rangle \langle \mathscr{J} \rangle$: By the argument in Theorem 4.1, it is easy to see for discrete jump sets $\langle \mathscr{A} \rangle$ and $\langle \mathscr{J} \rangle$ that the dL rules remain sound with nested jump prefix $\langle \mathscr{A} \rangle \langle \mathscr{J} \rangle$ in place of only a single prefix $\langle \mathscr{J} \rangle$ from Definition 2.10. Applicability conditions of rules do not depend on jump prefixes, as Definition 2.10 allows adding *any* jump prefix. Thus, we obtain a sound (extended) proof of $\langle \mathscr{A} \rangle \phi$ when replacing—with arbitrary unchanged context $\Gamma, \Delta, \langle \mathscr{J} \rangle$—every rule application of the form

$$\dfrac{\Gamma, \langle \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{J} \rangle \Psi_1, \Delta \quad \ldots \quad \Gamma, \langle \mathscr{J} \rangle \Phi_n \vdash \langle \mathscr{J} \rangle \Psi_n, \Delta}{\Gamma, \langle \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{J} \rangle \Psi_0, \Delta}$$

in the proof of ϕ by a rule application with the additional unchanged prefix $\langle \mathscr{A} \rangle$ for corresponding $\Gamma, \Delta, \langle \mathscr{J} \rangle$:

$$\dfrac{\Gamma, \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Psi_1, \Delta \quad \ldots \quad \Gamma, \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_n \vdash \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Psi_n, \Delta}{\Gamma, \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Psi_0, \Delta} \tag{2.14}$$

Next, we show that these nested jump prefixes can be reduced to a single jump prefix as Definition 2.10 allows: Let $\langle \mathscr{A}\, \mathscr{J} \rangle$ denote the discrete jump set obtained by merging $\langle \mathscr{A} \rangle$ and $\langle \mathscr{J} \rangle$ using $\langle := \rangle$ as in Sect. 2.5.2. We replace each rule application (with nested prefixes) of the form (2.14) by the following derivation with only a single prefix (assuming $n = 1$ for notational convenience):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\ldots}{\Gamma, \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{A}\, \mathscr{J} \rangle \Psi_1, \Delta} \quad \dfrac{*}{\Gamma, \langle \mathscr{A}\, \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{A}\, \mathscr{J} \rangle \Phi_1, \Delta}\,ax}{\Gamma, \langle \mathscr{A}\, \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_1, \Delta}\,\langle := \rangle}{\Gamma, \langle \mathscr{A}\, \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{A}\, \mathscr{J} \rangle \Psi_1, \Delta}\,cut}{\Gamma, \langle \mathscr{A}\, \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{A}\, \mathscr{J} \rangle \Psi_0, \Delta}}{\Gamma, \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{A} \rangle \langle \mathscr{J} \rangle \Psi_0, \Delta}\,\langle := \rangle, \langle := \rangle$$

The bottom most $\langle:=\rangle$ applications merge $\langle\mathscr{A}\rangle$ into $\langle\mathscr{J}\rangle$ in the antecedent and succedent, respectively. The unmarked rule applies the same rule that has been used in (2.14), which is applicable on $\Phi_0 \vdash \Psi_0$ for *any* context by Definition 2.10, including $\Gamma, \Delta, \langle\mathscr{A}\,\mathscr{J}\rangle$. The subsequent cut with $\langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Phi_1$ restores the form of the premise in (2.14). The left branch continues using a dual argument to turn succedent $\langle\mathscr{A}\,\mathscr{J}\rangle\Psi_1$ into $\langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Psi_1$, thereby yielding a set of non-extended rule applications with the same conclusions and premises as the extended rule application (2.14):

$$\dfrac{\dfrac{\Gamma, \langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Phi_1 \vdash \langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Psi_1, \Delta}{\Gamma, \langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Phi_1 \vdash \langle\mathscr{A}\,\mathscr{J}\rangle\Psi_1, \Delta}\langle:=\rangle \qquad \dfrac{*}{\Gamma, \langle\mathscr{A}\,\mathscr{J}\rangle\Psi_1 \vdash \langle\mathscr{A}\,\mathscr{J}\rangle\Psi_1, \Delta}ax}{\Gamma, \langle\mathscr{A}\rangle\langle\mathscr{J}\rangle\Phi_1 \vdash \langle\mathscr{A}\,\mathscr{J}\rangle\Psi_1, \Delta}cut$$

For reducing the first conjecture of this lemma to the second, let s be a Skolem constant for state variable x. By the above proof, we derive $\vdash_{\mathscr{D}} \langle x:=s\rangle\phi$. Using $\forall$r, we continue this derivation to a proof of $\forall X\,\langle x:=X\rangle\phi$, which we abbreviate as $\forall x\,\phi$ (see the text below Definition 2.8). Rule $\forall$r is applicable for Skolem constant s as no free logical variables occur in the proof. □

Now we prove two special cases of Theorem 2.3 for formulas of a special form.

Proposition 2.1 (Relative completeness of first-order safety). *For every hybrid program $\alpha \in \mathrm{HP}(\Sigma, V)$ and all FOD formulas $F, G \in \mathrm{Fml}_{FOD}(\Sigma, V)$*

$$\vDash F \to [\alpha]G \text{ implies } \vdash_{\mathscr{D}} F \to [\alpha]G \text{ (and } F \vdash_{\mathscr{D}} [\alpha]G \text{ by Lemma 2.10)}.$$

Proof. We generalise the relative completeness proof by Cook [87] to $\mathsf{d}\mathscr{L}$ and follow an induction on the structure of program α. In the following, *IH* is short for the induction hypothesis.

1. The cases where α is of the form $x_1 := \theta_1, \ldots, x_n := \theta_n$, $?\chi$, $\beta \cup \gamma$, or $\beta;\gamma$ are consequences of the soundness of the symmetric rules $[;],[\cup],[?],\langle:=\rangle,[:=]$. Since these rules are symmetric, they perform equivalent transformations. Consequently, whenever their conclusion is valid, their premise is valid and of smaller complexity (the programs get simpler), and hence derivable by IH. Thus, we can derive $F \to [\alpha]G$ by applying the respective rule. We explicitly show the proof for $\beta;\gamma$ as it contains an extra twist.
2. $\vDash F \to [\beta;\gamma]G$, which implies $\vDash F \to [\beta][\gamma]G$. By Lemma 2.9, there is a FOD formula $G^\#$ such that $\vDash G^\# \leftrightarrow [\gamma]G$. From the validity of $\vDash F \to [\beta]G^\#$, we can conclude by IH that $F \vdash_{\mathscr{D}} [\beta]G^\#$ is derivable. Similarly, due to $\vDash G^\# \to [\gamma]G$, we conclude $\vdash_{\mathscr{D}} G^\# \to [\gamma]G$ by IH. Using Lemma 2.11, we conclude that also $\vdash_{\mathscr{D}} \forall^\beta(G^\# \to [\gamma]G)$. With an application of $[]gen$, the latter derivation can be extended to a derivation of $[\beta]G^\# \vdash_{\mathscr{D}} [\beta][\gamma]G$. Combining the above derivations propositionally by a cut with $[\beta]G^\#$, we can derive $F \vdash_{\mathscr{D}} [\beta][\gamma]G$, from which $[;]$ yields $F \vdash_{\mathscr{D}} [\beta;\gamma]G$ as desired (and Lemma 2.10 or $\to$r yield $\vdash_{\mathscr{D}} F \to [\beta;\gamma]G$).

3. $\vDash F \to [x_1' = \theta_1, \ldots, x_n' = \theta_n]G$ is a FOD formula and hence derivable as a $\mathscr{D}$ axiom. Continuous evolution $x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi$ with evolution domain restrictions is definable in FOD by Lemma 2.8, which we consider as an abbreviation in this proof.
4. $\vDash F \to [\beta^*]G$ can be derived by induction. For this, we define the invariant as a FOD encoding of the statement that all potential post-states of β^* satisfy G according to Lemma 2.9:

$$\phi \equiv ([\beta^*]G)^{\#} \equiv \forall \vec{v}\,(\mathscr{S}_{\beta^*}(\vec{x},\vec{v}) \to G_{\vec{x}}^{\vec{v}}).$$

Since $F \to \phi$ and $\phi \to G$ are valid FOD formulas, they are derivable by $\mathscr{D}$; so is $F \vdash_{\mathscr{D}} \phi$ derivable by Lemma 2.10. By Lemma 2.11 and $[]gen$, $[\beta^*]\phi \vdash_{\mathscr{D}} [\beta^*]G$ is derivable. Likewise, $\phi \to [\beta]\phi$ is valid according to the semantics of repetition, and thus derivable by IH, since β is less complex. Using Lemma 2.11, we can derive $\vdash_{\mathscr{D}} \forall^{\beta}(\phi \to [\beta]\phi)$, from which *ind* yields $\phi \vdash_{\mathscr{D}} [\beta^*]\phi$. Combining the above derivations propositionally by a cut with $[\beta^*]\phi$ and ϕ yields $F \vdash_{\mathscr{D}} [\beta^*]G$. □

Proposition 2.2 (Relative completeness of first-order liveness). *For each hybrid program* $\alpha \in \mathrm{HP}(\Sigma, V)$ *and all FOD formulas* $F, G \in \mathrm{Fml}_{FOD}(\Sigma, V)$

$$\vDash F \to \langle\alpha\rangle G \textit{ implies } \vdash_{\mathscr{D}} F \to \langle\alpha\rangle G \textit{ (and } F \vdash_{\mathscr{D}} \langle\alpha\rangle G \textit{ by Lemma 2.10)}.$$

Proof. We generalise the arithmetic completeness proof by Harel [148] to the hybrid case. Most cases of the proof are simple adaptations of the corresponding cases in Proposition 2.1. What remains to be shown is the case of repetitions. Assume that $\vDash F \to \langle\beta^*\rangle G$. To derive this formula by *con*, we use a FOD formula $\varphi(n)$ as a variant expressing that, after n iterations, β can lead to a state satisfying G. This formula is obtained from Lemmas 2.8 and 2.9 as $(\langle\beta^*\rangle G)^{\#} \equiv \exists \vec{v}\,(\mathscr{S}_{\beta^*}(\vec{x},\vec{v}) \wedge G_{\vec{x}}^{\vec{v}})$, *except* that the quantifier on the repetition count n is removed such that n becomes a free variable (plus index shifting to count repetitions):

$$\varphi(n-1) \equiv \exists \vec{v} \exists Z \left(Z_1^{(n)} = \vec{x} \wedge Z_n^{(n)} = \vec{v} \wedge \forall i{:}\mathbb{N}\ (1 \le i < n \to \mathscr{S}_{\beta}(Z_i^{(n)}, Z_{i+1}^{(n)})) \wedge G_{\vec{x}}^{\vec{v}}\right).$$

By Lemma 2.7, $\varphi(n)$ can only hold true if n is a natural number.

According to the loop semantics, $\vDash n > 0 \wedge \varphi(n) \to \langle\beta\rangle\varphi(n-1)$ is valid by construction: If $n > 0$ is a natural number then so is $n-1$, and if β reaches G after n repetitions, then, after executing β once, $n-1$ repetitions of β reach G. By IH, this formula is derivable, since β contains less loops. By Lemma 2.11, we extend this derivation to $\vdash_{\mathscr{D}} \forall^{\beta} \forall n{>}0\,(\varphi(n) \to \langle\beta\rangle\varphi(n-1))$. Thus $\exists v\,\varphi(v) \vdash_{\mathscr{D}} \langle\beta^*\rangle \exists v{\le}0\,\varphi(v)$ by *con*. It only remains to show that the antecedent is derivable from F and $\langle\beta^*\rangle G$ is derivable from the succedent. From our assumption, we conclude that the following are valid FOD formulas, hence $\mathscr{D}$ axioms:

- $\vDash F \to \exists v\,\varphi(v)$, because $\vDash F \to \langle\beta^*\rangle G$, and

- $\vDash (\exists v{\leq}0\, \varphi(v)) \rightarrow G$, because $v \leq 0$, and the fact, that by Lemma 2.7, $\varphi(v)$ only holds true for natural numbers, imply $\varphi(0)$. Further, $\varphi(0)$ entails G, because zero repetitions of β have no effect.

From the latter we derive $\vdash_{\mathscr{D}} \forall^{\beta}(\exists v{\leq}0\, \varphi(v) \rightarrow G)$ by Lemma 2.11 and extend the derivation to $\langle\beta^*\rangle\exists v{\leq}0\, \varphi(v) \vdash_{\mathscr{D}} \langle\beta^*\rangle G$ by $\langle\rangle gen$. From $\vdash_{\mathscr{D}} F \rightarrow \exists v\, \varphi(v)$ we conclude $F \vdash_{\mathscr{D}} \exists v\, \varphi(v)$ by Lemma 2.10. Now, the above derivations can be combined propositionally by a cut with $\langle\beta^*\rangle\exists v{\leq}0\, \varphi(v)$ and with $\exists v\, \varphi(v)$ to yield $F \vdash_{\mathscr{D}} \langle\beta^*\rangle G$. □

2.7.6 Relative Completeness of the Differential Logic Calculus

Having succeeded with the proofs of the above statements we can finish the proof of Theorem 2.3, which is the central theoretical result of this chapter.

Proof (of Theorem 2.3). The proof follows a basic structure analogous to that of Harel's proof for the discrete case [148, Theorem 3.1]. We have to show that every valid $\mathsf{d}\mathscr{L}$ formula ϕ can be proven from FOD axioms within the $\mathsf{d}\mathscr{L}$ calculus: from $\vDash \phi$ we have to prove $\vdash_{\mathscr{D}} \phi$. The proof proceeds as follows: By propositional recombination, we inductively identify fragments of ϕ that correspond to $\phi_1 \rightarrow [\alpha]\phi_2$ or $\phi_1 \rightarrow \langle\alpha\rangle\phi_2$ logically. Next, we express subformulas ϕ_i equivalently in FOD by Lemma 2.9, and use Propositions 2.1 and 2.2 to resolve these first-order safety or liveness assertions. Finally, we prove that the original $\mathsf{d}\mathscr{L}$ formula can be re-derived from the subproofs.

We can assume ϕ to be given in conjunctive normal form by appropriate propositional reasoning. In particular, we assume that negations are pushed inside over modalities using the dualities $\neg[\alpha]\phi \equiv \langle\alpha\rangle\neg\phi$ and $\neg\langle\alpha\rangle\phi \equiv [\alpha]\neg\phi$. The remainder of the proof follows an induction on a measure $|\phi|$ defined as the number of modalities in ϕ. For a simple and uniform proof, we assume quantifiers to be abbreviations for modal formulas: $\exists x\, \phi \equiv \langle x' = 1\rangle\phi \vee \langle x' = -1\rangle\phi$ and $\forall x\, \phi \equiv [x' = 1]\phi \wedge [x' = -1]\phi$.

0. $|\phi| = 0$; then ϕ is a first-order formula; hence derivable by $\mathscr{D}$.
1. ϕ is of the form $\neg\phi_1$; then ϕ_1 is first-order, as we assumed negations to be pushed inside. Hence, $|\phi| = 0$ and Case 0 applies.
2. ϕ is of the form $\phi_1 \wedge \phi_2$, then individually deduce the simpler proofs for $\vdash_{\mathscr{D}} \phi_1$ and $\vdash_{\mathscr{D}} \phi_2$ by IH, which can be combined by rule $\wedge$r.
3. ϕ is a disjunction and—without loss of generality—has one of the following forms (otherwise use associativity and commutativity to select a different order for the disjunction):

$$\phi_1 \vee [\alpha]\phi_2$$
$$\phi_1 \vee \langle\alpha\rangle\phi_2$$

As a unified notation for those cases we use $\phi_1 \vee \langle\!\langle\alpha\rangle\!\rangle\phi_2$. Then, $|\phi_2| < |\phi|$, since ϕ_2 has less modalities. Likewise, $|\phi_1| < |\phi|$ because $\langle\!\langle\alpha\rangle\!\rangle\phi_2$ contributes one modality to $|\phi|$ that is not part of ϕ_1.
According to Lemma 2.9 there are FOD formulas $\phi_1^\#, \phi_2^\#$ with $\vDash \phi_i \leftrightarrow \phi_i^\#$ for $i = 1,2$. By congruence, the validity $\vDash \phi$ yields $\vDash \phi_1^\# \vee \langle\!\langle\alpha\rangle\!\rangle\phi_2^\#$, which directly implies $\vDash \neg\phi_1^\# \rightarrow \langle\!\langle\alpha\rangle\!\rangle\phi_2^\#$. Then by Propositions 2.1 or 2.2, respectively, we can derive

$$\neg\phi_1^\# \vdash_{\mathscr{D}} \langle\!\langle\alpha\rangle\!\rangle\phi_2^\#. \tag{2.15}$$

Further $\vDash \phi_1 \leftrightarrow \phi_1^\#$ implies $\vDash \neg\phi_1 \rightarrow \neg\phi_1^\#$, which is derivable by IH, because $|\phi_1| < |\phi|$. By Lemma 2.10, we obtain $\neg\phi_1 \vdash_{\mathscr{D}} \neg\phi_1^\#$, which we combine with (2.15) by a cut with $\neg\phi_1^\#$ to

$$\neg\phi_1 \vdash_{\mathscr{D}} \langle\!\langle\alpha\rangle\!\rangle\phi_2^\#. \tag{2.16}$$

Likewise $\vDash \phi_2 \leftrightarrow \phi_2^\#$ implies $\vDash \phi_2^\# \rightarrow \phi_2$, which is derivable by IH, as $|\phi_2| < |\phi|$. We can extend the derivation of $\vdash_{\mathscr{D}} \phi_2^\# \rightarrow \phi_2$ to one of $\vdash_{\mathscr{D}} \forall^\alpha(\phi_2^\# \rightarrow \phi_2)$ by Lemma 2.11 and conclude $\langle\!\langle\alpha\rangle\!\rangle\phi_2^\# \vdash_{\mathscr{D}} \langle\!\langle\alpha\rangle\!\rangle\phi_2$ by $[]gen$–$\langle\rangle gen$. Finally we combine the latter derivation propositionally with (2.16) by a cut with $\langle\!\langle\alpha\rangle\!\rangle\phi_2^\#$ to derive $\neg\phi_1 \vdash_{\mathscr{D}} \langle\!\langle\alpha\rangle\!\rangle\phi_2$, from which $\vdash_{\mathscr{D}} \phi_1 \vee \langle\!\langle\alpha\rangle\!\rangle\phi_2$ can be obtained, again using *cut*, to complete the proof. □

This concludes the main theoretical proof of relative completeness of the $\mathsf{d}\mathscr{L}$ calculus, i.e., of Theorem 2.3.

2.8 Relatively Semidecidable Fragments

To strengthen the completeness result from Theorem 2.3, we consider fragments of $\mathsf{d}\mathscr{L}$ where the required FOD tautologies are sufficiently simple as differential equations have first-order definable flows and the required loop invariants (or variants) are expressible in first-order logic over the reals. In these fragments, the only difficulty is to find the required invariants and variants for the proof. Relative to an (ineffective) oracle that provides first-order invariants and variants for repetitions, the $\mathsf{d}\mathscr{L}$ calculus can be used as a semidecision procedure. That is, when we assume the oracle to provide suitable (in)variants, validity of formulas can be proven in the $\mathsf{d}\mathscr{L}$ calculus. If an imperfect oracle chooses inadequate (in)variants, applying the $\mathsf{d}\mathscr{L}$ calculus rules results in goals that are not valid, which is again decidable by quantifier elimination in the $\mathsf{d}\mathscr{L}$ calculus.

Theorem 2.4 (Relatively semidecidable fragment). *Relative to an oracle generating first-order invariants and variants, the $\mathsf{d}\mathscr{L}$ calculus gives a backtracking-free semidecision procedure for (closed) $\mathsf{d}\mathscr{L}$ formulas with differential equations having first-order definable flows.*

Proof (Outline). The (constructive) proof, which, in full, can be found in the remainder of this section, shows that there are always applicable dℒ rules that transform the formulas equivalently and that formulas in this dℒ proof descend along a well-founded order. For loops, we assume that suitable (in)variants are obtained from the oracle and we can guarantee termination when these (in)variants are first-order (or contain fewer loops). □

As a consequence, enumerating first-order invariants or variants gives a semidecision procedure for the fragment of Theorem 2.4. As a corollary to Theorems 2.2 and 2.4, there are valid dℒ formulas that need proper dℒ (or FOD) invariants to be provable and cannot be proven just using (in)variants of first-order real arithmetic. Similarly, the fragment with first-order definable flows and bounded loops is decidable: When loops α^* are annotated with natural numbers indicating the maximum number of repetitions of α, an effective oracle for Theorem 2.4 can be obtained by unrolling, e.g., by rule $\langle{*n}\rangle$.

As an auxiliary result for proving Theorem 2.4, we show that, in dℒ proofs, Skolem symbols occur in a uniform way, i.e., a Skolem symbol s always occurs with the same list of arguments.

Lemma 2.12 (Uniform Skolem symbols). *Let ϕ be a dℒ formula without Skolem symbols. In any derivation of ϕ, Skolem symbols only occur with a unique list of free logical variables as arguments, provided that the formulas in cuts (rule cut) obey this restriction.*

Proof. The proof is by induction on the structure of proofs in the dℒ calculus. For derivations of length zero, the conjecture holds, because ϕ does not contain Skolem symbols. We show that the conjectured Skolem occurrence property is preserved in all subgoals when applying a rule to a goal that satisfies the conjecture.

- ∀r The symbols $s(X_1,\ldots,X_n)$ introduced by rules ∀r,∃l are of the required form as the X_i are precisely the free logical variables. In addition, the symbol $s(X_1,\ldots,X_n)$ does not occur nested in other Skolem terms, because, by the induction hypothesis, the bound variable x does not occur in Skolem terms of the goal.
- i∀ Rules i∀ and i∃ are only applicable to instances of first-order real arithmetic (Lemma 2.5), for which the equivalence transformations of quantifier elimination preserve the Skolem occurrence property, because they never introduce quantifiers to bind free variables.
- $\langle'\rangle$ Rule $\langle'\rangle$ preserves the property, as it only substitutes state variables $x_i \in \Sigma$, not logical variables $X_i \in V$.
- *cut* Cuts preserve the Skolem occurrence property, as we assumed the formulas that *cut* introduces to adhere to the Skolem occurrence property.
- – The other rules of the dℒ calculus preserve the property as they never replace arguments of Skolem function symbols (which are free variables by induction hypothesis). □

Proof (of Theorem 2.4). The proof is by well-founded induction. We prove that there is a well-founded strict partial order $\prec$ such that:

IH: For all non-atomic formulas occurring in the sequents during a proof, there is an applicable series of $\mathsf{d}\mathscr{L}$ rules such that all resulting subgoals are simpler with respect to $\prec$ and have no additional free variables or function symbols, and their conjunction is equivalent to the conclusion (for suitable oracle choices).

By applying these $\mathsf{d}\mathscr{L}$ rules exhaustively, we obtain a decision procedure relative to the oracle, because the subgoals descend along the well-founded order $\prec$, which has no infinite descending chain. Finally, validity of the remaining sequents with atomic formulas is decidable by evaluating ground instances (Definition 2.9), because, by IH, the resulting formulas have no free variables when the initial formula is closed (open formulas, in contrast, yield equivalent parameter constraints as results). We use the derived rules *ind′* and *con′* from p. 86 in place of *ind* and *con*; see Sect. 2.5.2. To obtain a backtracking-free procedure, we remove rules $\langle^{*n}\rangle$,$[^{*n}]$, $[]gen$,$\langle\rangle gen$,*ind*,*con* and *cut* from the calculus: If a calculus with less rules gives a decision procedure, then so does the full calculus.

We define the order $\prec$ as the lexicographical order of, respectively, the number of: loops, differential equations, sequential compositions, choices, modalities, quantifiers, number of different variables and Skolem function symbols, and the number of logical connectives. As a lexicographical order of natural numbers, $\prec$ is well-founded [99]. It lifts to sequents in rule applications (Definition 2.10) when all subgoals of all rule schemata are simpler than their goals with respect to $\prec$, which can be shown to retain well-foundedness as a multiset ordering [99].

Now the proof of IH is by induction along $\prec$. Let ϕ be a non-atomic formula of a sequent in an open branch of the proof. We assume ϕ to occur in the succedent; the respective proofs for the antecedent are dual. Hence, we consider the sequent to be of the form $\Gamma \vdash \phi, \Delta$.

1. If ϕ is of the form $\psi_1 \wedge \psi_2$, then rule $\wedge$r is applicable, yielding smaller sequents (with less logical connectives) that are equivalent. Other logical connectives are handled likewise using rules $\neg$r,$\vee$r,$\wedge$r,$\rightarrow$r, respectively.
2. If ϕ is of the form $[\alpha]\psi$ or $\langle\alpha\rangle\psi$ and α is of the form $?\chi$, $\beta;\gamma$ or $\beta\cup\gamma$, the corresponding rule $\langle;\rangle$,$[;]$,$\langle\cup\rangle$,$[\cup]$ or $\langle?\rangle$,$[?]$ is applicable, yielding a simpler yet equivalent formula.
3. If ϕ is of the form $[x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi]\psi$, then rule $[']$ is applicable, as we assumed differential equations to have first-order definable flows. The resulting formula is equivalent and simpler, because it contains fewer differential equations. It involves additional bound variables but not free variables. Case $\langle x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, \chi\rangle\psi$ is similar, by rule $\langle'\rangle$.
4. If ϕ is of the form $[\alpha^*]\psi$, then rule *ind′* is applicable with a first-order invariant F obtained from the oracle. The resulting subgoals are simpler according to $\prec$, because they contain less loops (F does not contain loops). The resulting subgoals do not have additional free variables as all bound variables of α^* remain bound by the universal closure $\forall^\alpha$ in the respective premises. Finally, we assume the oracle to give an invariant such that the conjunction of the resulting subgoals is equivalent to the goal (otherwise we have nothing to show for

inadequate choices by the oracle). The case $\langle \alpha^* \rangle \psi$ is similar, using rule *con*′ instead.

5. If ϕ is of the form $\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle \psi$, there are two cases. If rule $\langle := \rangle$ is applicable, it yields equivalent simpler sequents. Otherwise, we have

$$\psi \prec \langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle \psi.$$

Thus, by IH, there is a finite sequence of rule applications on ψ yielding equivalent sequents with atomic formulas. Prefixing the resulting proof with $\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle$ yields a corresponding proof for deriving $\Gamma \vdash \phi, \Delta$ by Lemma 2.11. The formulas of the open branches of this proof resulting from ϕ are of the form $\langle x_1 := \theta_1, \ldots, x_n := \theta_n \rangle G$ for atomic formulas G, where, at the latest, rule $\langle := \rangle$ is applicable, as substitutions are admissible on atomic formulas. Case $[x_1 := \theta_1, \ldots, x_n := \theta_n] \psi$ is similar, using rule $[:=]$ first.

6. If ϕ is of the form $\forall x \, \psi(x)$, we can apply rule $\forall$r giving $\psi(s(X_1, \ldots, X_n))$. Now, we have $\psi(s(X_1, \ldots, X_n)) \prec \forall x \, \psi(x)$; hence, by IH, $\psi(s(X_1, \ldots, X_n))$ can be transformed equivalently to a set of sequents of the form

$$\Phi_i(s(X_1, \ldots, X_n)) \vdash \Psi_i(s(X_1, \ldots, X_n))$$

with atomic formulas (without loss of generality, we can assume $s(X_1, \ldots, X_n)$ to occur in all branches). Hence, QE is defined for these atomic formulas and rule i$\forall$ can be applied on each branch, yielding $\mathrm{QE}(\forall s \, (\Phi_i(s) \vdash \Psi_i(s)))$. Consequently, the original sequent $\Gamma \vdash \forall x \, \psi(x), \Delta$ is equivalent to

$$\bigwedge_i \mathrm{QE}(\forall s \, (\Phi_i(s) \vdash \Psi_i(s)))$$

for the following reason: $\Gamma \vdash \psi(s(X_1, \ldots, X_n)), \Delta$ is equivalent to

$$\bigwedge_i (\Phi_i(s(X_1, \ldots, X_n)) \vdash \Psi_i(s(X_1, \ldots, X_n)))$$

by IH, using the equivalence $\mathrm{QE}(\forall s \, (F \wedge G)) \equiv \mathrm{QE}(\forall s \, F) \wedge \mathrm{QE}(\forall s \, G)$ and the fact that s does not occur in Γ, Δ. After applying rule i$\forall$, the result has no additional free symbols, although intermediate formulas do.

7. If ϕ is of the form $\exists x \, \psi(x)$, then rule $\exists$r is applicable giving $\psi(X)$ for a fresh logical variable X. Then $\psi(X) \prec \exists x \, \psi(x)$; hence, by IH, $\psi(X)$ can be transformed equivalently to a set of sequents $\Phi_i \vdash \Psi_i$ with atomic formulas. If no Skolem dependency on X occurs in $\Phi_i \vdash \Psi_i$, then QE is defined and rule i$\exists$ applicable, giving $\mathrm{QE}(\exists X \bigwedge_i (\Phi_i \vdash \Psi_i))$, which is equivalent to $\exists X \bigwedge_i (\Phi_i \vdash \Psi_i)$. By IH, this is equivalent to $\Gamma \vdash \exists X \, \psi(X), \Delta$, because X does not occur in Γ, Δ. Otherwise, if a Skolem term $s(X_1, \ldots, X, \ldots, X_n)$ occurs in a $\Phi_i \vdash \Psi_i$, then, by IH, the Skolem function s already occurred in $\psi(X)$. By Lemma 2.12, the Skolem term $s(X_1, \ldots, X, \ldots, X_n)$ itself must already have occurred in $\psi(X)$, which contradicts the fact that X is fresh and that bound variable x does not occur in Skolem

terms of $\exists x\, \psi(x)$, again by Lemma 2.12. After applying rule i$\exists$ the additional free variable X disappears. □

This completes the proof of Theorem 2.4, showing that the dℒ calculus can be used as a semidecision procedure for a particular set of (in)variants provided by an oracle. Consequently, these results show that, in a certain sense, finding (in)variants is the only challenge in hybrid systems' verification, because the dℒ calculus takes care of everything else. In Chap. 3 we revisit and strengthen this result, because we show that properties of differential equations can be proven by appropriate generalisations of (in)variants that we call differential invariants. Furthermore, we turn to the challenge of finding these (differential) invariants in Chap. 6.

2.9 Train Control Verification

In this section, we verify collision avoidance of the train control system presented in Sect. 2.4. Especially, we identify the constraints required for the free parameters of the system and discover the preconditions for safe driving.

2.9.1 Finding Inductive Candidates

Recall the dℒ formula from Sect. 2.4 that expresses that the simplified ETCS train control system ensures that trains always stay inside their movement authority m to ensure collision-freedom:

$$\psi \rightarrow [(ctrl;drive)^*]z \leq m \tag{2.7*}$$

We want to prove safety statement (2.7) of the simplified version of the European Train Control System. Note that this is a significantly simplified version showing only the true essentials of ETCS. We consider the ETCS cooperation protocol in more detail in Chap. 7.

Using parametric extraction techniques, we identify both the requirement ψ for safe driving and the induction hypothesis ϕ that is required for the proof. Similar to the proof in Fig. 2.16, which is dual to the proof in Fig. 2.14, an unwinding of the loop in (2.7) by rule $[^{*n}]$ can be used to extract a candidate for a parametric inductive hypothesis. It expresses that there is sufficient braking distance at current speed v, which basically corresponds to the controllability constraint for ETCS (as illustrated in Fig. 2.15 on p. 90):

$$\phi \equiv v^2 \leq 2b(m-z) \wedge b > 0 \wedge A \geq 0 \ . \tag{2.17}$$

2.9.2 Inductive Verification

Using proof rule *ind* to prove $\mathsf{d}\mathscr{L}$ formula (2.7) by induction, we show that (a) invariant ϕ holds initially, i.e., $\psi \vdash \phi$ (implying the antecedent of the conclusion of *ind*), (b) the invariant is sustained after each execution of $ctrl;drive$, and (c) invariant ϕ implies postcondition $z \leq m$. Case (c) holds by QE, as $0 \leq v^2 \leq 2b(m-z)$ and $b > 0$. The induction start (a) will be examined after the full proof, since we want to identify the prerequisite ψ for safe driving by proof analysis. In the proof of the induction step $\phi \to [ctrl;drive]\phi$, we omit condition $m - z \leq s$ from *ctrl*, because it is not used in the proof (braking remains safe with respect to $z \leq m$). The induction is provable in $\mathsf{d}\mathscr{L}$ as follows (for notational convenience, we assume rule $\forall$r calls the Skolem constant for m again m, and so on, as there are no free logical variables):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\phi \vdash \langle a := -b\rangle [drive]\phi} \quad \dfrac{\dots}{\phi, m - z \geq s \vdash \langle a := A\rangle [drive]\phi}}{\phi \vdash [?m - z \geq s; a := A][drive]\phi}\,[?],{\to}\mathrm{r}}}{\phi \vdash [ctrl][drive]\phi}\,[\cup],{\wedge}\mathrm{r}}}{\phi \vdash [ctrl;drive]\phi}\,[;]}}{\vdash \phi \to [ctrl;drive]\phi}\,{\to}\mathrm{r}}}{\vdash \forall^{\alpha}(\phi \to [ctrl;drive]\phi)}\,\forall\mathrm{r}}}{\phi \vdash [(ctrl;drive)^*]\phi}\,ind
$$

The differential equation system in *drive* is linear with a constant coefficient matrix M. Its solution can be obtained by symbolically computing the exponential series $e^{Mt}\eta$ with symbolic initial value $\eta = (z, v)$ and similar symbolic integration of the inhomogeneous part [297, §18.VI]; also see App. B.4. We abbreviate the solution $\langle z := -\frac{b}{2}t^2 + vt + z, v := -bt + v\rangle$ thus obtained by $\langle \mathscr{S}_t \rangle$. See Example B.3 in App. B for an explanation of why this is a solution of the differential equations. In this example, the evolution domain restrictions are convex; hence the constraint $\forall 0 \leq \tilde{t} \leq t \langle \mathscr{S}_{\tilde{t}} \rangle \chi$ of rule [′] can be simplified to $\langle \mathscr{S}_t \rangle \chi$ as in (2.12) to save space. Further, we leave out conditions which are unnecessary for closing the above proof. In the left branch, the constrained evolution of clock τ is irrelevant and will be left out to save space (braking is the safest operation and can be continued indefinitely without extra risk). The left branch closes (marked $*$):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\phi, t \geq 0, -bt + v \geq 0 \vdash \langle \mathscr{S}_t \rangle \phi}\,\langle:=\rangle,\mathrm{i}\forall}{\phi, t \geq 0, \langle v := -bt + v\rangle v \geq 0 \vdash \langle \mathscr{S}_t \rangle \phi}\,\langle:=\rangle}{\phi \vdash t \geq 0 \to (\langle v := -bt + v\rangle v \geq 0 \to \langle \mathscr{S}_t \rangle \phi)}\,{\to}\mathrm{r},{\to}\mathrm{r}}{\phi \vdash \forall t{\geq}0\,(\langle v := -bt + v\rangle v \geq 0 \to \langle \mathscr{S}_t \rangle \phi)}\,\forall\mathrm{r}}{\phi \vdash [z' = v, v' = -b\,\&\,v \geq 0]\phi}\,[']}{\phi \vdash \langle a := -b\rangle [drive]\phi}\,\langle:=\rangle}{\phi \vdash [a := -b][drive]\phi}\,[:=]
$$

The right branch does not need the evolution domain constraint $v \geq 0$, because v does not decrease when accelerating. We again use $\langle \mathscr{S}_t \rangle$ as an abbreviation for the solution $\langle z := \frac{A}{2}t^2 + vt + z, v := At + v \rangle$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\phi, m-z \geq s \vdash s \geq \frac{v^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right)}}{\phi, m-z \geq s, 0 \leq t \leq \varepsilon \vdash \langle \mathscr{S}_t \rangle \phi}\,\langle:=\rangle,\mathrm{i}\forall}{\phi, m-z \geq s \vdash t \geq 0 \rightarrow (\langle \tau := t \rangle \tau \leq \varepsilon \rightarrow \langle \mathscr{S}_t \rangle \phi)}\,{\rightarrow}\mathrm{r},\langle:=\rangle}{\phi, m-z \geq s \vdash \forall t{\geq}0\,(\langle \tau := t \rangle \tau \leq \varepsilon \rightarrow \langle \mathscr{S}_t \rangle \phi)}\,\forall\mathrm{r}}{\phi, m-z \geq s \vdash \langle \tau := 0 \rangle \forall t{\geq}0\,(\langle \tau := t+\tau \rangle \tau \leq \varepsilon \rightarrow \langle \mathscr{S}_t \rangle \phi)}\,\langle:=\rangle}{\phi, m-z \geq s \vdash \langle \tau := 0 \rangle [z' = v, v' = A, \tau' = 1 \,\&\, \tau \leq \varepsilon] \phi}\,[']}{\phi, m-z \geq s \vdash [\tau := 0][z' = v, v' = A, \tau' = 1 \,\&\, \tau \leq \varepsilon] \phi}\,[:=]}{\phi, m-z \geq s \vdash \langle a := A \rangle [\tau := 0][z' = v, v' = a, \tau' = 1 \,\&\, \tau \leq \varepsilon] \phi}\,\langle:=\rangle}{\phi, m-z \geq s \vdash \langle a := A \rangle [drive] \phi}\,[;]}{\phi, m-z \geq s \vdash [a := A][drive] \phi}\,[:=]
$$

2.9.3 Parameter Constraint Discovery

The right branch only closes when the succedent of its open goal is guaranteed. That formula expresses that there will still be sufficient braking distance even after accelerating by $\leq A$ for up to ε seconds:

$$s \geq \frac{v^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right). \tag{2.18}$$

This constraint can be discovered automatically in the above proof by the indicated application of rule i$\forall$ using quantifier elimination with some simplifications. Constraint (2.18) is required to make sure invariant (2.17) still holds after accelerating. In fact, augmenting the case study with (2.18) makes the argument inductive, and the whole proof of the safety statement (2.7) closes when ψ is chosen identical to ϕ. Here, the conditions of ψ cannot be removed without leaving the proof open due to a counterexample, as the invariant (2.17) is a controllability constraint; see Sect. 2.5.3.1.

Quite unlike in the acceleration-free case [231], constraint (2.18) needs to be enforced dynamically as the affected variables change over time. That is, at the beginning of each *ctrl* cycle, s needs to be updated in accordance with (2.18), which admits complex behaviour as in Fig. 2.9b on p. 63. Further, this constraint can be used to find out how densely a track can be packed with trains in order to maximise ETCS throughput without compromising safety. The resulting provably safe train control system can be summarised as follows:

$$v^2 \leq 2b(m-z) \wedge b > 0 \wedge A \geq 0 \;\rightarrow [(ctrl; drive)^*] z \leq m \tag{2.19}$$

$$\begin{aligned} \text{where } ctrl &\equiv s := \frac{v^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v\right); \\ &\quad (?m - z \le s; a := -b) \cup (?m - z \ge s; a := A) \\ drive &\equiv \tau := 0; (z' = v, v' = a, \tau' = 1 \,\&\, v \ge 0 \wedge \tau \le \varepsilon). \end{aligned}$$

Using the $\mathsf{d}\mathscr{L}$ calculus, similar constraints can be derived (Sect. 4.8) to find out how early a train needs to start negotiation in order to minimise the risk of having to reduce speed when the MA is not extensible in time, which is the ST parameter of Fig. 2.8.

For the resulting ETCS system, liveness can be proven in the $\mathsf{d}\mathscr{L}$ calculus by showing that the train can pass every point p by an appropriate choice of m by the RBC:

$$z = z_0 \wedge v = v_0 > 0 \wedge \varepsilon > 0 \wedge b > 0 \wedge A \ge 0 \rightarrow \forall p \, \exists m \, \langle (ctrl; drive)^* \rangle z \ge p. \quad (2.20)$$

For $A = 0$, the proof of property (2.20) uses the variant $\varphi(n) \equiv z + n\varepsilon v_0 \ge p \wedge v = v_0$ for rule *con*, which expresses that the speed does not decrease (until $n < 0$) and that the remaining distance from z to target p can be covered after at most n iteration cycles. This directly proves the property even when $A = 0$ for appropriate acceleration choices. For general $A \ge 0$, the following variant proves property (2.20) by *con*:

$$\varphi(n) \equiv ((z + n\varepsilon v_0 \ge p \wedge z_0 \le z \wedge v^2 \le v_0^2 + 2A(z - z_0) \wedge v \ge v_0 \wedge z \le p) \vee z \ge p) \wedge v \ge 0. \quad (2.21)$$

It expresses that, when $z \le p$, the remaining distance can be covered after at most n iterations while the train position and velocity increase, yet the velocity is bounded depending on the initial velocity v_0, acceleration A, and distance $z - z_0$. The appropriate choice of m to prove property (2.20) with this variant is

$$m \ge p + \frac{v_o^2 + 2A(p - z_0)}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2\right) + \varepsilon\sqrt{v_0^2 + 2A(p - z_0)},$$

which can be obtained by overapproximating braking condition (2.18) with the speed limit $v^2 \le v_0^2 + 2A(z - z_o)$ from the variant. We will analyse ETCS in more detail in Chap. 7.

In this example, we can see the effect of the $\mathsf{d}\mathscr{L}$ calculus. It takes a specification of a hybrid system and successively identifies constraints on the parameters which are needed for correctness. These constraints can then be handled in a purely modular way by rules i$\forall$ and i$\exists$. As a typical characteristic of hybrid systems, further observe that intermediate formulas are significantly more complex than the original proof obligation, which can be expressed succinctly in the expressive language of $\mathsf{d}\mathscr{L}$. This reflects the fact that the actual complexity of hybrid systems originates from hybrid interaction, not from a single transition. Still, using appropriate proof strategies (Chap. 5) for the $\mathsf{d}\mathscr{L}$ calculus, the safety statement (2.7) with invariant (2.17) can be verified automatically in a theorem prover that invokes Mathem-

atica for rules $\langle' \rangle$,[′], i∀, and i∃. In fact, using the invariant computation techniques that we introduce in Chap. 6, the overall safety property (2.7) can be verified fully automatically even without providing an invariant manually.

2.10 Summary

We have introduced a first-order dynamic logic for hybrid programs, which are uniform operational models for hybrid systems with interacting discrete jumps and continuous evolutions along differential equations. For this differential dynamic logic, dℒ, we have presented a concise generalised free-variable proof calculus over the reals.

Our sequent calculus for dℒ is a generalisation of classical calculi for discrete dynamic logic [35, 37, 149, 148] to the hybrid case. It is a compositional verification calculus for verifying properties of hybrid programs by decomposing them into properties of their parts. In order to handle interacting hybrid dynamics, we lift real quantifier elimination to the deductive calculus in a new modular way that is suitable for automation, using real-valued free variables, Skolem terms, and invertible quantifier rules over the reals.

As a fundamental result aligning hybrid and continuous reasoning proof-theoretically, we have proven our calculus to axiomatise the transition behaviour of hybrid systems completely relative to the handling of differential equations. To the best of our knowledge, this is the first relative completeness result for hybrid systems' verification. Moreover, we have demonstrated that our calculus is well suited for practical automatic verification in a realistic case study of a fully parametric version of the European Train Control System.

Dynamic logic can be augmented [37] to support reasoning about dynamically reconfiguring system structures, which we want to extend to hybrid systems in future work. While the dℒ calculus is complete relative to the continuous fragment, it is a subtle open problem whether a converse calculus can exist that is complete relative to various discrete fragments.

http://www.springer.com/978-3-642-14508-7

Logical Analysis of Hybrid Systems
Proving Theorems for Complex Dynamics
Platzer, A.
2010, XXX, 426 p., Hardcover
ISBN: 978-3-642-14508-7