A Temporal Dynamic Logic for Verifying Hybrid System Invariants*

André Platzer

University of Oldenburg, Department of Computing Science, Germany
Carnegie Mellon University, Computer Science Department, Pittsburgh, PA
platzer@informatik.uni-oldenburg.de

Abstract. We combine first-order dynamic logic for reasoning about possible behaviour of hybrid systems with temporal logic for reasoning about the temporal behaviour during their operation. Our logic supports verification of hybrid programs with first-order definable flows and provides a uniform treatment of discrete and continuous evolution. For our combined logic, we generalise the semantics of dynamic modalities to refer to hybrid traces instead of final states. Further, we prove that this gives a conservative extension of dynamic logic. On this basis, we provide a modular verification calculus that reduces correctness of temporal behaviour of hybrid systems to non-temporal reasoning. Using this calculus, we analyse safety invariants in a train control system and symbolically synthesise parametric safety constraints.

Keywords: dynamic logic, temporal logic, sequent calculus, logic for hybrid systems, deductive verification of embedded systems

1 Introduction

Correctness of real-time and hybrid systems depends on a safe operation throughout *all* states of all possible trajectories, and the behaviour at intermediate states is highly relevant [1, 7, 9, 12, 14, 23].

Temporal logics (TL) use temporal operators to talk about intermediate states [1, 10, 11, 24]. They have been used successfully in model checking [1, 6, 14, 15, 18] of finite-state system abstractions. Continuous state spaces of hybrid systems, however, often do not admit equivalent finite-state abstractions [14,18]. Instead of model checking, TL can also be used deductively to prove validity of formulas in calculi [8, 9]. Valid TL formulas, however, only express very generic facts that are true for all systems, regardless of their actual behaviour. Hence, the behaviour of a specific system first needs to be axiomatised declaratively to obtain meaningful results. Then, however, the correspondence between actual system operations and a declarative temporal representation may be questioned.

* This research was supported by a fellowship of the German Academic Exchange Service (DAAD). It was also sponsored by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see www.avacs.org).

S. Artemov and A. Nerode (Eds.): LFCS 2007, LNCS 4514, pp. 457–471, 2007.

Dynamic logic (DL) [13] is a successful approach for deductively verifying (infinite-state) systems [2, 3, 13, 16]. Like model checking, DL can analyse the behaviour of actual system models, which are specified operationally. Yet, operational models are *internalised* within DL-formulas, and DL is closed under logical operators. Thus, DL can refer to multiple systems and analyse their relationship. This can be important for verifying larger systems compositionally or for investigating refinement relations, see [22]. Further, Davoren and Nerode [9] argue that, unlike model checking, deductive methods support formulas with free parameters. However, DL only considers the behaviour at final states, which is insufficient for verifying safety invariants that have to hold all the time.

We close this gap of expressivity by combining first-order dynamic logic [13] with temporal logic [10,11,24]. Moreover, we generalise both operational system models and semantics to hybrid systems [14]. In this paper, we introduce a temporal dynamic logic dTL, which provides modalities for quantifying over traces of hybrid systems. We equip it with temporal operators to state what is true all along a trace or at some point during a trace. As in our non-temporal dynamic logic $\mathsf{d}\mathcal{L}$ [19, 20, 22], we use hybrid programs as an operational model for hybrid systems. They admit a uniform treatment of interacting discrete and continuous evolution in logic.

As a semantical foundation for combined temporal dynamic formulas, we introduce a hybrid trace semantics for dTL. We prove that dTL is a conservative extension of $\mathsf{d}\mathcal{L}$: for non-temporal specifications, trace semantics is equivalent to the non-temporal final state semantics of [19, 22].

As a means for verification, we introduce a sequent calculus for dTL that successively reduces temporal statements about traces of hybrid programs to non-temporal formulas. In this way, we make the intuition formally precise that safety invariants can be checked by augmenting proofs with appropriate assertions about intermediate states. Like in [22], our calculus supports compositional reasoning. It structurally decomposes correctness statements about hybrid programs into corresponding statements about its parts by symbolic transformation.

Our approach combines the advantages of DL in reasoning about the behaviour of (multiple and parametric) operational system models with those of TL to verify temporal statements about traces. On the downside, we show that our logic is incomplete. Yet, reachability in hybrid systems is already undecidable [14]. We argue that, despite this theoretical obstacle, dTL can verify practical systems and demonstrate this by studying safety invariants in train control [7, 12].

The first contribution of this paper is the logic dTL, which provides a coherent foundation for reasoning about the temporal behaviour of operational models of hybrid systems with symbolic parameters. The main contribution is our calculus for deductively verifying temporal statements about hybrid systems.

Hybrid Systems. The behaviour of safety-critical systems typically depends on both the state of a discrete controller and continuous physical quantities. Hybrid systems are mathematical models for dynamic systems with interacting discrete

and continuous behaviour [9,14]. Their behaviour combines continuous evolution (called *flow*) characterised by differential equations and discrete jumps.

Dynamic Logic. The principle of dynamic logic is to combine system operations and correctness statements about system states within a single specification language (see [13] for a general introduction in the discrete case). By permitting system operations α as actions of modalities, dynamic logic provides formulas of the form $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, where $[\alpha]\phi$ expresses that all terminating runs of system α lead to final states in which condition ϕ holds. Likewise, $\langle\alpha\rangle\phi$ expresses that it is possible for α to execute and result in a final state satisfying ϕ. In dTL, hybrid programs [19, 20, 22] play the role of α. In this paper, we modify the semantics of $[\alpha]$ to refer to all *traces* of α rather than only all final states reachable with α (similarly for $\langle\alpha\rangle$). For instance, the formula $[\alpha]\Box\phi$ expresses that ϕ is true at each state during all traces of the hybrid system α. With this, dTL can also be used to verify temporal statements about the behaviour of α at intermediate states during system runs.

Related Work. Based on [25], Beckert and Schlager [4] added separate trace modalities to dynamic logic and presented a relatively complete calculus. Their approach only handles discrete state spaces. In contrast, dTL works for hybrid programs with continuous state spaces. There, a particular challenge is that invariants may change their truth-value during a single continuous evolution.

Mysore et al. [18] analysed model checking of TCTL [1] properties for semi-algebraic hybrid systems and proved undecidability. Our logic internalises operational models and supports multiple parametric systems.

Zhou et al. [26] presented a duration calculus extended by mathematical expressions with derivatives of state variables. Their calculus is unwieldy as it uses a multitude of rules and requires external mathematical reasoning about derivatives and continuity.

Davoren and Nerode [9] extended the propositional modal μ-calculus with a semantics in hybrid systems and examine topological aspects. In [8], Davoren et al. gave a semantics in general flow systems for a generalisation of CTL* [11]. In both cases, the authors of [9] and [8] provided Hilbert-style calculi to prove formulas that are valid for all systems simultaneously using abstract actions.

The strength of our logic primarily is that it is a first-order dynamic logic: it handles actual hybrid programs like $x := x + 1; \dot{x} = 2y$ rather than only abstract actions of unknown effect. Our calculus directly supports verification of hybrid programs with first-order definable flows; first-order approximations of more general flows can be used according to [23]. First-order DL is more expressive and calculi are deductively stronger than other approaches [4,17].

Structure of this Paper. After introducing syntax and semantics of the temporal dynamic logic dTL in Sect. 2, we introduce a sequent calculus for verifying temporal dTL specifications of hybrid systems in Sect. 4 and prove soundness. In Sect. 5, we prove safety invariants of the train control system presented in Sect. 3.

Alternating path and trace quantifiers for liveness verification are discussed in Sect. 6. Finally, we draw conclusions and discuss future work in Sect. 7.

2 Temporal Dynamic Logic for Hybrid Systems

2.1 Overview: The Basic Concepts of dTL

The temporal dynamic logic dTL extends dynamic logic [13] with three concepts for verifying temporal specifications of hybrid systems:

Hybrid programs. The behaviour of hybrid systems can be described by hybrid programs [19, 20, 22], which generalise real-time programs [15] to hybrid change. The distinguishing feature of hybrid programs in this context is that they provide uniform discrete jumps and continuous evolutions along differential equations. While hybrid automata [14] can be embedded, program structures are more amenable to compositional symbolic processing by calculus rules [19].

Modal operators. Modalities of dynamic logic express statements about all possible behaviour ($[\alpha]\pi$) of a system α, or about the existence of a trace ($\langle\alpha\rangle\pi$), satisfying condition π. As in [19, 20, 22], the system α is described as a hybrid program. Yet, unlike in standard dynamic logic [13], π is a *trace formula* in dTL, and π is allowed to refer to all states that occur *during* a trace using temporal operators.

Temporal operators. For dTL, the temporal trace formula $\Box\phi$ expresses that the formula ϕ holds all along a trace selected by $[\alpha]$ or $\langle\alpha\rangle$. For instance, the state formula $\langle\alpha\rangle\Box\phi$ says that the state formula ϕ holds at every state along at least one trace of α. Dually, the trace formula $\Diamond\phi$ expresses that ϕ holds at some point during such a trace. It can occur in a state formula $\langle\alpha\rangle\Diamond\phi$ to express that there is such a state in some trace of α, or as $[\alpha]\Diamond\phi$ to say that, along each trace, there is a state satisfying ϕ. In this paper, the primary focus of attention is on homogeneous combinations of path and trace quantifiers like $[\alpha]\Box\phi$ or $\langle\alpha\rangle\Diamond\phi$.

2.2 Syntax of dTL

State and Trace Formulas. The formulas of dTL are built over a non-empty set V of real-valued variables and a fixed signature Σ of function and predicate symbols. For simplicity, Σ is assumed to contain exclusively the usual function and predicate symbols for real arithmetic, such as $0, 1, +, \cdot, =, \leq, <, \geq, >$.

The set $\mathrm{Trm}(V)$ of *terms* is defined as in classical first-order logic. The formulas of dTL are defined similar to first-order dynamic logic [13]. However, the modalities $[\alpha]$ and $\langle\alpha\rangle$ accept trace formulas that refer to the temporal behaviour of *all* states along a trace. Inspired by CTL and CTL* [10, 11], we distinguish between state formulas, that are true or false in states, and trace formulas, that are true or false for system traces. The sets $\mathrm{Fml}(V)$ of state formulas, $\mathrm{Fml}_T(V)$ of trace formulas, and $\mathrm{HP}(V)$ of hybrid programs with variables in V are simultaneously inductively defined in Definition 1 and 2, respectively.

Definition 1 (Formulas). *The set* $\mathrm{Fml}(V)$ *of* (state) formulas *is simultaneously inductively defined as the smallest set such that:*

1. *If* $p \in \Sigma$ *is a predicate,* $\theta_1, \ldots, \theta_n \in \mathrm{Trm}(V)$*, then* $p(\theta_1, \ldots, \theta_n) \in \mathrm{Fml}(V)$.
2. *If* $\phi, \psi \in \mathrm{Fml}(V)$*, then* $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi) \in \mathrm{Fml}(V)$.
3. *If* $\phi \in \mathrm{Fml}(V)$ *and* $x \in V$*, then* $\forall x\, \phi, \exists x\, \phi \in \mathrm{Fml}(V)$.
4. *If* $\pi \in \mathrm{Fml}_T(V)$ *and* $\alpha \in \mathrm{HP}(V)$*, then* $[\alpha]\pi, \langle\alpha\rangle\pi \in \mathrm{Fml}(V)$.

The set $\mathrm{Fml}_T(V)$ *of* trace formulas *is the smallest set with:*

1. *If* $\phi \in \mathrm{Fml}(V)$*, then* $\phi \in \mathrm{Fml}_T(V)$.
2. *If* $\phi \in \mathrm{Fml}(V)$*, then* $\Box\phi, \Diamond\phi \in \mathrm{Fml}_T(V)$.

Formulas without $\Box$ and $\Diamond$, i.e., without case 2 of the trace formulas, are called *non-temporal* $\mathsf{d}\mathcal{L}$ *formulas* [19, 22]. Unlike in CTL, state formulas are true on a trace (case 1) if they hold for the *last* state of a trace, not for the first. Thus, $[\alpha]\phi$ expresses that ϕ is true at the end of each trace of α. In contrast, $[\alpha]\Box\phi$ expresses that ϕ is true all along all states of every trace of α. This combination gives a smooth embedding of non-temporal $\mathsf{d}\mathcal{L}$ into dTL and makes it possible to define a compositional calculus. Like CTL, dTL allows nesting with a branching time semantics [10], e.g., $[\alpha]\Box(x \geq 2 \rightarrow \langle\gamma\rangle\Diamond x \leq 0)$.

Hybrid Programs. The hybrid programs [19, 20, 22] occurring in dynamic modalities of dTL are built from elementary discrete jumps and continuous evolutions using a regular control structure [13].

Definition 2 (Hybrid programs). *The set* $\mathrm{HP}(V)$ *of* hybrid programs *is inductively defined as the smallest set such that:*

1. *If* $x \in V$ *and* $\theta \in \mathrm{Trm}(V)$*, then* $(x := \theta) \in \mathrm{HP}(V)$.
2. *If* $x \in V$ *and* $\theta \in \mathrm{Trm}(V)$*, then* $(\dot{x} = \theta) \in \mathrm{HP}(V)$.
3. *If* $\chi \in \mathrm{Fml}(V)$ *is quantifier-free and first-order, then* $(?\chi) \in \mathrm{HP}(V)$.
4. *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha \cup \gamma) \in \mathrm{HP}(V)$.
5. *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha; \gamma) \in \mathrm{HP}(V)$.
6. *If* $\alpha \in \mathrm{HP}(V)$ *then* $(\alpha^*) \in \mathrm{HP}(V)$.

The effect of $x := \theta$ is an instantaneous discrete jump in state space or a mode switch. That of $\dot{x} = \theta$ is an ongoing continuous evolution regulated by the differential equation with time-derivative $\dot{x}$ of x and term θ (accordingly for systems of differential equations).

Test actions $?\chi$ are used to define conditions. Their semantics is that of a no-op if χ is true in the current state, and that of a dead end operator aborting any further evolution, otherwise. The sequential composition $\alpha; \gamma$, nondeterministic choice $\alpha \cup \gamma$, and non-deterministic repetition α^* of system actions are as usual [13]. They can be combined with $?\chi$ to form other control structures [13].

In dTL, there is no need to distinguish between discrete and continuous variables or between system parameters and state variables, as they share the same

uniform semantics. For pragmatic reasons, an informal distinction can nevertheless improve readability. For instance, $\exists x\,[\dot{x}=-x]x\leq 5$ expresses that there is a choice of the initial value for x (which could be a parameter) such that after all evolutions along $\dot{x}=-x$, the outcome of the state variable x will be at most 5.

2.3 Trace Semantics of dTL

In standard dynamic logic [13] and $\mathsf{d}\mathcal{L}$ [19,22], modalities only refer to the final states of system runs and the semantics is a reachability relation on states: State ω is reachable from state ν using α if there is a run of α which terminates in ω when started in ν. For dTL, however, formulas can refer to intermediate states of runs as well. Thus, the semantics of a hybrid system α is the set of its possible *traces*, i.e., successions of states that occur during the evolution of α.

States contain values of system variables during a hybrid evolution. A *state* is a map $\nu : V \to \mathbb{R}$; the set of all states is denoted by $\mathrm{Sta}(V)$. In addition, we distinguish a state Λ to denote the failure of a system run when it is *aborted* due to a test $?\chi$ that yields *false*. In particular, Λ can only occur at the end of an aborted system run and marks that there is no further extension.

Hybrid systems evolve along piecewise continuous traces in multi-dimensional space as time passes. Continuous phases are governed by differential equations, whereas discontinuities are caused by discrete jumps in state space. Unlike in discrete cases [4,25], traces are not just sequences of states, since hybrid systems pass through uncountably many states even in bounded time. Beyond that, continuous changes are more involved than in pure real-time [1, 15], because all variables can evolve along different differential equations. Generalising the real-time traces of [15], the following definition captures hybrid behaviour by splitting the uncountable succession of states into periods σ_i that are regulated by the same control law. For discrete jumps, some periods are point flows of duration 0.

Definition 3 (Hybrid Trace). *A* trace *is a (non-empty) finite or infinite sequence $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$ of functions $\sigma_i : [0, r_i] \to \mathrm{Sta}(V)$ with respective durations $r_i \in \mathbb{R}$ (for $i \in \mathbb{N}$). A* position *of σ is a pair (i, ζ) with $i \in \mathbb{N}$ and ζ in the interval $[0, r_i]$; the state of σ at (i, ζ) is $\sigma_i(\zeta)$. Positions of σ are ordered lexicographically by $(i, \zeta) \prec (j, \xi)$ iff either $i < j$, or $i = j$ and $\zeta < \xi$. Further, for a state $\nu \in \mathrm{Sta}(V)$, $\hat{\nu} : 0 \mapsto \nu$ is the* point flow *at ν with duration 0. A trace* terminates *if it is a finite sequence $(\sigma_0, \sigma_1, \dots, \sigma_n)$ and $\sigma_n(r_n) \neq \Lambda$. In that case, the last state* $\mathrm{last}\,\sigma$ *is denoted as $\sigma_n(r_n)$. The first state* $\mathrm{first}\,\sigma$ *is $\sigma_0(0)$.*

Unlike in [1,15], the definition of traces also admits finite traces of bounded duration, which is necessary for compositionality of traces in $\alpha;\gamma$. The semantics of hybrid programs α as the set $\tau(\alpha)$ of its possible traces depends on valuations $val(\nu, \cdot)$ of formulas and terms at intermediate states ν. The valuation of terms [13], and interpretations of function and predicate symbols are as usual for real arithmetic. The valuation of formulas will be defined in Definition 5. We use $\nu[x \mapsto d]$ to denote the *modification* that agrees with state ν on all variables except for the symbol x, which is changed to $d \in \mathbb{R}$.

Definition 4 (Trace semantics of hybrid programs). *The* trace semantics, $\tau(\alpha)$, of a hybrid program α, *is the set of all its possible hybrid traces and is defined as follows:*

1. $\tau(x := \theta) = \{(\hat{\nu}, \hat{\omega}) \ : \ \omega = \nu[x \mapsto val(\nu, \theta)]$ *for* $\nu \in \mathrm{Sta}(V)\}$
2. $\tau(\dot{x} = \theta) = \{(f) \ : \ 0 \leq r \in \mathbb{R}$ *and* $f : [0, r] \to \mathrm{Sta}(V)$ *is such that the function* $val(f(\zeta), x)$ *is continuous in* ζ *on* $[0, r]$ *and has a derivative of value* $val(f(\zeta), \theta)$ *at each* $\zeta \in (0, r)$. *Variables without a differential equation do not change*$\}$
3. $\tau(?\chi) = \{(\hat{\nu}) \ : \ val(\nu, \chi) = true\} \cup \{(\hat{\nu}, \hat{\Lambda}) \ : \ val(\nu, \chi) = false\}$
4. $\tau(\alpha \cup \gamma) = \tau(\alpha) \cup \tau(\gamma)$
5. $\tau(\alpha; \gamma) = \{\sigma \circ \varsigma \ : \ \sigma \in \tau(\alpha)\,, \varsigma \in \tau(\gamma)$ *when* $\sigma \circ \varsigma$ *is defined*$\}$; *the composition of* $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$ *and* $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \dots)$ *is*

$$\sigma \circ \varsigma = \begin{cases} (\sigma_0, \dots, \sigma_n, \varsigma_0, \varsigma_1, \dots) & \text{if } \sigma \text{ terminates at } \sigma_n \text{ and } \operatorname{last}\sigma = \operatorname{first}\varsigma \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$

6. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$, *where* $\alpha^{n+1} = (\alpha^n; \alpha)$ *for* $n \geq 1$, *and* $\alpha^0 = (?true)$.

Time passes differently during discrete and continuous change. During continuous evolution, the discrete step index i of positions (i, ζ) remains constant, whereas the continuous duration ζ remains 0 during discrete point flows. This permits multiple discrete state changes to happen at the same (super-dense) continuous time, unlike in [1].

Definition 5 (Valuation of formulas). *The valuation of state and trace formulas is defined respectively. For state formulas, the* valuation $val(\nu, \cdot)$ *with respect to state* ν *is defined as follows:*

1. $val(\nu, p(\theta_1, \dots, \theta_n)) = p^\ell\big(val(\nu, \theta_1), \dots, val(\nu, \theta_n)\big)$, *where* p^ℓ *is the relation associated to* p.
2. $val(\nu, \phi \wedge \psi)$ *is defined as usual, the same holds for* $\neg, \vee, \to$.
3. $val(\nu, \forall x\, \phi) = true :\Longleftrightarrow val(\nu[x \mapsto d], \phi) = true$ *for all* $d \in \mathbb{R}$
4. $val(\nu, \exists x\, \phi) = true :\Longleftrightarrow val(\nu[x \mapsto d], \phi) = true$ *for some* $d \in \mathbb{R}$
5. $val(\nu, [\alpha]\pi) = true :\Longleftrightarrow$ *for each trace* $\sigma \in \tau(\alpha)$ *that starts in* $\operatorname{first}\sigma = \nu$, *if* $val(\sigma, \pi)$ *is defined, then* $val(\sigma, \pi) = true$.
6. $val(\nu, \langle\alpha\rangle\pi) = true :\Longleftrightarrow$ *there is a trace* $\sigma \in \tau(\alpha)$ *starting in* $\operatorname{first}\sigma = \nu$, *such that* $val(\sigma, \pi) = true$.

For trace formulas, the valuation $val(\sigma, \cdot)$ *with respect to trace* σ *is:*

1. *If* ϕ *is a state formula, then* $val(\sigma, \phi) = val(\operatorname{last}\sigma, \phi)$ *if* σ *terminates, whereas* $val(\sigma, \phi)$ *is not defined if* σ *does not terminate.*
2. $val(\sigma, \Box\phi) = true :\Longleftrightarrow val(\sigma_i(\zeta), \phi) = true$ *for all positions* (i, ζ) *of* σ *with* $\sigma_i(\zeta) \neq \Lambda$.
3. $val(\sigma, \Diamond\phi) = true :\Longleftrightarrow val(\sigma_i(\zeta), \phi) = true$ *for some position* (i, ζ) *of* σ *with* $\sigma_i(\zeta) \neq \Lambda$.

As usual, a (state) formula is *valid* if it is true in all states.

2.4 Conservative Temporal Extension

The following result shows that the extension of dTL by temporal operators does not change the meaning of non-temporal $\mathsf{d}\mathcal{L}$ formulas. The trace semantics given in Definition 5 is equivalent to the final state reachability relation semantics [19, 22] for the sublogic $\mathsf{d}\mathcal{L}$ of dTL. A proof for this can be found in [21].

Proposition 1. *The logic* dTL *is a* conservative extension *of non-temporal* $\mathsf{d}\mathcal{L}$*, i.e., the set of valid* $\mathsf{d}\mathcal{L}$*-formulas is the same with respect to transition reachability semantics of* $\mathsf{d}\mathcal{L}$ *[19, 22] as with respect to the trace semantics of* dTL *(Definition 5).*

3 Safety Invariants in Train Control

In the European Train Control System (ETCS) [12], trains are coordinated by decentralised Radio Block Centres (RBC), which grant or deny movement authorities (MA) to the individual trains by wireless communication. In emergencies, trains always have to stop within the MA issued by the RBC, see Fig. 1. Following the reasoning pattern for traffic agents in [7], each train negotiates with the RBC to extend its MA when approaching the end, say m, of its current MA. Since wireless communication takes time, this negotiation is initiated in due time before reaching m. During negotiation, trains are assumed to keep their desired speed as in [7]. Before entering negotiation at some point ST, the train still has sufficient distance to MA (it is in *far* mode) and can regulate its speed freely within the track limits.

Depending on weather conditions, slope of track etc., the local train motion control determines a safety envelope s around the train, within which it considers driving safe, and adjusts its acceleration a in accordance with s (called *correction* [7]). In particular, depending on the maximum RBC response time, this determines the latest point, SB, on the track where a response from the RBC must have arrived to guarantee safe driving.

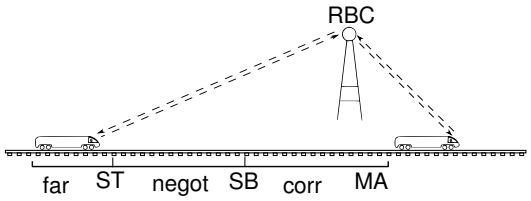

Fig. 1. ETCS train coordination by movement authorities

As a model for train movements, we use the ideal-world model adapted from [7]. It does not model friction, slopes, or mass of train but is perfectly suitable for analysing the cooperation level of train control [7]. The local safety

properties that where used when verifying the cooperation protocol can then be shown for more detailed models of individual components.

For a safe operation of multiple traffic agents, it is crucial that the MA is respected at *every* point in time during this protocol, not only at its end. Hence, we need to consider temporal safety invariants. For instance, when the train has entered the negotiation phase at its current position z, dTL can analyse the following safety invariant of a part of the train controller:

$$\psi \;\rightarrow\; [\mathit{negot}; \mathit{corr}; \dot{z} = v, \dot{v} = a]\Box(\ell \leq L \rightarrow z < m) \tag{1}$$

$$\text{where } \mathit{negot} \;\equiv\; \dot{z} = v, \dot{\ell} = 1$$

$$\mathit{corr} \;\equiv\; (?m - z < s; a := -b) \cup (?m - z \geq s; a := \dots) \;.$$

It expresses that—under a sanity condition ψ for parameters—a train will *always* remain within its MA m, as long as the accumulated RBC negotiation latency ℓ is at most L. We refer to [12] for details on what contributes to ℓ. Like in [7], we model the train to first negotiate while keeping a constant speed ($\dot{z} = v$) in *negot*. Thereafter, in *corr*, the train corrects its acceleration or brakes with force b (as a failsafe recovery manoeuvre) on the basis of the remaining distance $(m - z)$. Finally, the train continues moving according to the system ($\dot{z} = v, \dot{v} = a$) or, equivalently, $\ddot{z} = a$. Instead of manually choosing specific values for the free parameters of (1) as in [7,12], we will use the techniques developed in this paper to automatically synthesise constraints on the relationship of parameters that are required for a safe operation of cooperative train control.

4 A Verification Calculus for Safety Invariants

In this section, we introduce a sequent calculus for verifying temporal specifications of hybrid systems in dTL. With the basic idea being to perform a symbolic decomposition, hybrid programs are successively transformed into simpler logical formulas describing their effects. There, statements about the temporal behaviour of a hybrid program are successively reduced to corresponding non-temporal statements about the intermediate states.

For propositional logic, standard rules P1–P9 are listed in Fig. 2. The rule P10 is a shortcut to handle quantifiers of first-order real arithmetic, which is decidable. We use P10 as a modular interface to arithmetic and refer to [19] for a goal-oriented integration of arithmetic, which combines with dTL. Rules D1–D8 work similar to those in [3,13]. For handling discrete change, D8 inductively uses substitutions. D9–D10 handle continuous evolutions given a first-order definable flow y_x. In particular, in conjunction with P10, they fully encapsulate handling of differential equations within hybrid systems.

Rules T1–T10 successively transform temporal specifications of hybrid programs into non-temporal $\mathsf{d}\mathcal{L}$ formulas. The idea underlying this transformation is to decompose hybrid programs and recursively augment intermediate state transitions with appropriate specifications. D1–D2 are identical for dTL and $\mathsf{d}\mathcal{L}$ specifications, hence they apply for all trace formulas π and not just for state formulas. Rules for handling $[\alpha]\Diamond\phi$ and $\langle\alpha\rangle\Box\phi$ are discussed in Sect. 6.

4.1 Rules of the Calculus

A *sequent* is of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sets of formulas. Its semantics is that of the formula $\bigwedge_{\phi\in\Gamma}\phi \rightarrow \bigvee_{\psi\in\Delta}\psi$ and will be treated as an abbreviation. In the following, an *update* $\mathcal{U}$ is a list of discrete assignments of the form $x := \theta$ (see [3] for advanced update techniques, which can be combined with our calculus).

Definition 6 (Provability, derivability). *A formula ψ is* provable *from a set Φ of formulas, denoted by $\Phi \vdash_{\mathrm{dTL}} \psi$ iff there is a finite set $\Phi_0 \subseteq \Phi$ for which the sequent $\Phi_0 \vdash \psi$ is derivable. In turn, a sequent of the form $\Gamma, \langle\mathcal{U}\rangle\Phi \vdash \langle\mathcal{U}\rangle\Psi, \Delta$ (for some update $\mathcal{U}$, including the empty update, and finite sets Γ, Δ of context formulas) is* derivable *iff there is an instance*

$$\frac{\Phi_1 \vdash \Psi_1 \qquad \ldots \qquad \Phi_n \vdash \Psi_n}{\Phi \vdash \Psi}$$

of a rule schema of the dTL *calculus in Fig. 2 such that*

$$\Gamma, \langle\mathcal{U}\rangle\Phi_i \vdash \langle\mathcal{U}\rangle\Psi_i, \Delta$$

is derivable for each $1 \leq i \leq n$. Moreover, the symmetric schemata Di and Ti can be applied on either side of the sequent (in context Γ, Δ and update $\langle\mathcal{U}\rangle$). The schematic modality $\langle\!|\cdot|\!\rangle$ can be instantiated with both $[\cdot]$ and $\langle\cdot\rangle$ in all rule schemata. The same modality instance has to be chosen within a single schema instantiation, though.

As usual in sequent calculus—although the direction of entailment is from premisses (above rule bar) to conclusion (below)—the order of reasoning is *goal-directed*: Rules are applied starting from the desired conclusion at the bottom (goal) to the premisses (sub-goals).

Rule T1 decomposes invariants of $\alpha;\gamma$ into an invariant of α and an invariant of γ that holds when γ is started in *any* final state of α. T3 expresses that invariants of assignments need to hold before and after the discrete change (similarly for T2, except that tests do not lead to a state change). T4 can directly reduce invariants of continuous evolutions to non-temporal formulas as restrictions of solutions of differential equations are themselves solutions of different duration. T5 relies on T1 and is simpler than D7, because the other rules will inductively produce a premiss that ϕ holds in the current state. The dual rules T6–T10 work similarly. The usual induction schemes [13, 17] can be added to the dTL calculus. Inductive invariant properties can be handled by augmenting induction rules with an additional branch that takes care of the temporal properties.

4.2 Soundness and Incompleteness

The following result shows that verification with the dTL calculus always produces correct results about safety of hybrid systems, i.e., the dTL calculus is sound.

(P1) $\dfrac{\vdash \phi}{\neg\phi \vdash}$ (P4) $\dfrac{\phi, \psi \vdash}{\phi \wedge \psi \vdash}$ (P7) $\dfrac{\phi \vdash \qquad \psi \vdash}{\phi \vee \psi \vdash}$

(P2) $\dfrac{\phi \vdash}{\vdash \neg\phi}$ (P5) $\dfrac{\vdash \phi \qquad \vdash \psi}{\vdash \phi \wedge \psi}$ (P8) $\dfrac{\vdash \phi, \psi}{\vdash \phi \vee \psi}$

(P3) $\dfrac{\phi \vdash \psi}{\vdash \phi \to \psi}$ (P6) $\dfrac{\vdash \phi \qquad \psi \vdash}{\phi \to \psi \vdash}$ (P9) $\dfrac{}{\phi \vdash \phi}$

(P10) $\dfrac{F_0 \vdash G_0}{F \vdash G}$

(D1) $\dfrac{\langle\alpha\rangle\pi \vee \langle\gamma\rangle\pi}{\langle\alpha \cup \gamma\rangle\pi}$ (D6) $\dfrac{\phi \vee \langle\alpha;\alpha^*\rangle\phi}{\langle\alpha^*\rangle\phi}$

(D2) $\dfrac{[\alpha]\pi \wedge [\gamma]\pi}{[\alpha \cup \gamma]\pi}$ (D7) $\dfrac{\phi \wedge [\alpha;\alpha^*]\phi}{[\alpha^*]\phi}$

(D3) $\dfrac{\langle\!\langle\alpha\rangle\!\rangle\langle\!\langle\gamma\rangle\!\rangle\phi}{\langle\!\langle\alpha;\gamma\rangle\!\rangle\phi}$ (D8) $\dfrac{F_x^\theta}{\langle\!\langle x := \theta\rangle\!\rangle F}$

(D4) $\dfrac{\chi \wedge \phi}{\langle?\chi\rangle\phi}$ (D9) $\dfrac{\exists t{\geq}0\, \langle x := y_x(t)\rangle\phi}{\langle\dot{x} = \theta\rangle\phi}$

(D5) $\dfrac{\chi \to \phi}{[?\chi]\phi}$ (D10) $\dfrac{\forall t{\geq}0\, [x := y_x(t)]\phi}{[\dot{x} = \theta]\phi}$

(T1) $\dfrac{[\alpha]\Box\phi \wedge [\alpha][\gamma]\Box\phi}{[\alpha;\gamma]\Box\phi}$ (T6) $\dfrac{\langle\alpha\rangle\Diamond\phi \vee \langle\alpha\rangle\langle\gamma\rangle\Diamond\phi}{\langle\alpha;\gamma\rangle\Diamond\phi}$

(T2) $\dfrac{\phi}{[?\chi]\Box\phi}$ (T7) $\dfrac{\phi}{\langle?\chi\rangle\Diamond\phi}$

(T3) $\dfrac{\phi \wedge [x := \theta]\phi}{[x := \theta]\Box\phi}$ (T8) $\dfrac{\phi \vee \langle x := \theta\rangle\phi}{\langle x := \theta\rangle\Diamond\phi}$

(T4) $\dfrac{[\dot{x} = \theta]\phi}{[\dot{x} = \theta]\Box\phi}$ (T9) $\dfrac{\langle\dot{x} = \theta\rangle\phi}{\langle\dot{x} = \theta\rangle\Diamond\phi}$

(T5) $\dfrac{[\alpha;\alpha^*]\Box\phi}{[\alpha^*]\Box\phi}$ (T10) $\dfrac{\langle\alpha;\alpha^*\rangle\Diamond\phi}{\langle\alpha^*\rangle\Diamond\phi}$

In these rules, ϕ and ψ are (state) formulas, whereas π is a trace formula. Unlike ϕ and ψ, the trace formula π may thus begin with $\Box$ or $\Diamond$. In D8, F is a first-order formula and the substitution of F_x^θ, which replaces x by θ in F, does not introduce new bindings. In D9–D10, t is a fresh variable and y_v the solution of the initial value problem $(\dot{x} = \theta, x(0) = v)$. In P10, $\mathrm{Cl}_\forall\,(F_0 \to G_0) \to \mathrm{Cl}_\forall\,(F \to G)$ is an instance of a first-order tautology of real arithmetic and $\mathrm{Cl}_\forall$ the universal closure.

Fig. 2. Rule schemata of the temporal dynamic dTL verification calculus.

Theorem 1 (Soundness). *The* dTL *calculus is* sound*, i.e., derivable (state) formulas are valid. (See [21] for a proof.)*

Theorem 2 (Incompleteness). *Fragments of* dTL *are inherently incomplete, i.e. cannot have a complete calculus. (See [21] for a proof.)*

5 Verification of Train Control Safety Invariants

Continuing the ETCS study from Sect. 3, we consider a slightly simplified version of equation (1) that gives a more concise proof. By a safe abstraction (provable in dTL), we simplify *corr* to permit braking even when $m - z \geq s$, since braking remains safe with respect to $z < m$. We use the following abbreviations in addition to (1):

$$\begin{aligned} \psi &\equiv z < m \land v > 0 \land \ell = 0 \land L \geq 0 \\ \phi &\equiv \ell \leq L \rightarrow z < m \\ \mathit{corr} &\equiv a := -b \cup (?m - z \geq s; a := \dots) \ . \end{aligned}$$

Within the following proof, $\langle\!|\,|\!\rangle$ brackets are used instead of modalities to visually identify the update prefix (Definition 6). To give shorter formulas, we generalise update application D8 to work within quantifiers according to [3]. The dTL proof of the safety invariant in (1) splits into two cases as follows:

$$\dfrac{\dfrac{\dfrac{\dots}{\psi \vdash [\mathit{negot}]\Box\phi} \qquad \dfrac{\dots}{\psi \vdash [\mathit{negot}][\mathit{corr}; \dot z = v, \dot v = a]\Box\phi}}{\psi \vdash [\mathit{negot}; \mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{T1}}{\vdash \psi \rightarrow [\mathit{negot}; \mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{P3}$$

There, the left branch proves that ϕ holds while negotiating and is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\psi \vdash Lv + z < m}{\psi \vdash \forall l{\geq}0\,(l \leq L \rightarrow lv + z < m)}\,\text{P10}}{\psi \vdash \forall l{\geq}0\,\langle\!|z := lv + z, \ell := l|\!\rangle\phi}\,\text{D8}}{\psi \vdash [\mathit{negot}]\phi}\,\text{D10}}{\psi \vdash [\mathit{negot}]\Box\phi}\,\text{T4}$$

The right branch shows that ϕ continues to hold after negotiation has completed when continuing with an adjusted acceleration a:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\psi, \ell{\geq}0 \vdash v^2 < 2b(m - Lv - z) \land Lv + z < m}{\psi, \ell{\geq}0 \vdash \langle\!|z := \ell v{+}z, a := \text{-}b|\!\rangle\forall t{\geq}0\,(\ell{\leq}L \rightarrow \frac{a}{2}t^2{+}vt{+}z{<}m)}\,\text{P10}}{\psi, \ell{\geq}0 \vdash \langle\!|z := \ell v{+}z, a := \text{-}b|\!\rangle\forall t{\geq}0\,\langle\!|z := \frac{a}{2}t^2{+}vt{+}z|\!\rangle\phi)}\,\text{D8}}{\psi, \ell{\geq}0 \vdash \langle\!|z := \ell v{+}z, a := \text{-}b|\!\rangle[\dot z = v, \dot v = a]\Box\phi}\,\text{T4,D10}}{\psi, \ell{\geq}0 \vdash \langle\!|z := \ell v{+}z|\!\rangle[\mathit{corr}][\dot z = v, \dot v = a]\Box\phi}\,\text{D2}}{\psi, \ell{\geq}0 \vdash \langle\!|z := \ell v{+}z|\!\rangle[\mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{T1}}{\psi \vdash \ell{\geq}0 \rightarrow \langle\!|z := \ell v{+}z|\!\rangle[\mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{P3}}{\psi \vdash \forall \ell{\geq}0\,\langle\!|z := \ell v{+}z|\!\rangle[\mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{P10}}{\psi \vdash [\mathit{negot}][\mathit{corr}; \dot z = v, \dot v = a]\Box\phi}\,\text{D10}$$

The application of T1 in this latter case spawns a third case (marked with $\triangleright$) to show that ϕ holds during *corr*. However, the reasoning in this third case is subsumed by the cases above, since the changes on a in *corr* do not interfere with condition ϕ. Generally, this optimisation of T1 is applicable whenever the modified vocabulary is disjoint from ϕ. Here, D10 and P10 are implemented in Mathematica to handle evolutions [19].

The leaves of the proof branches above can even be used to automatically *synthesise parameter constraints* that are necessary to avoid MA violation. The parametric safety constraint obtained by combining the open conditions conjunctively is $Lv + z < m \wedge v^2 < 2b(m - Lv - z)$. It simplifies to $v^2 < 2b(m - Lv - z)$ as $b > 0$. This yields bounds for the speed limit and negotiation latency in order to guarantee safe driving and closing of the proof. Similarly, D2 leads to a branch for the case $[?m - z \geq s; a := \ldots]$, from which corresponding conditions about the safety envelope s can be derived depending on the particular speed controller. Yet, this is beyond the scope of this paper.

6 Liveness by Quantifier Alternation

Liveness specifications of the form $[\alpha]\Diamond\phi$ or $\langle\alpha\rangle\Box\phi$ are sophisticated (Σ^1_1-hard because they can express infinite occurrence in Turing machines). Beckert and Schlager [4] say they failed to find sound rules for a discrete case that corresponds to $[\alpha;\gamma]\Diamond\phi$.

For *finitary liveness semantics*, we accomplish this as follows. In this section, we modify the meaning of $[\alpha]\Diamond\phi$ to refer to all *terminating* traces of α. Then, the straightforward generalisation T11 in Fig. 3 is sound, even in the hybrid case(see [21] for proofs). But T11 still leads to an incomplete axiomatisation as it does not cover the case where, in some traces, ϕ becomes true at some point during α, and in other traces, ϕ only becomes true during γ. To overcome this limitation, we use a program transformation approach. We instrument the hybrid program to monitor the occurrence of ϕ during all changes: In T12, $\check{\alpha}$ results from replacing all occurrences of $x := \theta$ by $x := \theta; ?\phi \to t = 1$ and $\dot{x} = \theta$ by $\dot{x} = \theta \,\&\, (\phi \to t = 1)$. The latter denotes continuous evolution restricted to the region of the state space that satisfies $\phi \to t = 1$ (see [19] for details). The effect is that t detects whether ϕ has occurred during any change in α. In particular, t is guaranteed to be 1 after all runs, if ϕ occurs at least once along all traces of α. This trick directly works for quantifier-free first-order conditions ϕ. Using the combination presented in [22], nominals can be used as state labels to address the same issue for general ϕ.

$$\text{(T11)}\ \frac{\vdash [\alpha]\Diamond\phi, [\alpha][\gamma]\Diamond\phi}{\vdash [\alpha;\gamma]\Diamond\phi} \qquad \text{(T12)}\ \frac{\phi \vee \forall t\, [\check{\alpha}]t = 1}{[\alpha]\Diamond\phi}$$

Fig. 3. Transformation rules for alternating temporal path and trace quantifiers.

7 Conclusions and Future Work

For reasoning about hybrid systems, we have introduced a temporal dynamic logic, dTL, with modal path quantifiers over traces and temporal quantifiers along the traces. It combines the capabilities of dynamic logic [13] to reason about possible system behaviour with the power of temporal logic [10, 11, 24] in reasoning about the behaviour along traces. Furthermore, we have presented a calculus for verifying temporal safety specifications of hybrid programs in dTL.

Our sequent calculus for dTL is a modular combination of temporal and non-temporal reasoning. Temporal formulas are handled using rules that augment intermediate state transitions with corresponding sub-specifications. Purely non-temporal rules handle the effects of discrete and continuous evolution.

As an example, we demonstrate that our logic is suitable for reasoning about safety invariants in the European Train Control System [12]. Further, we have successfully applied our calculus to automatically synthesise (non-linear) parametric safety constraints for this system.

We are currently extending our preliminary verification tool for parametric hybrid systems to cover the full dTL calculus. Future work includes extending dTL with CTL*-like [11] formulas of the form $[\alpha](\psi \wedge \Box\phi)$ to avoid splitting of the proof into two very similar sub-proofs for temporal parts $[\alpha]\Box\phi$ and non-temporal parts $[\alpha]\psi$ arising in T1. Our combination of temporal logic with dynamic logic is more suitable for this purpose than the approach in [4], since dTL has uniform modalities and uniform semantics for temporal and non-temporal specifications. This extension will also simplify the treatment of alternating liveness quantifiers conceptually.

References

1. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.
2. B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *LNCS*. Springer-Verlag, 2007.
3. B. Beckert and A. Platzer. Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In U. Furbach and N. Shankar, editors, *IJCAR*, volume 4130 of *LNCS*, pages 266–280. Springer, 2006.
4. B. Beckert and S. Schlager. A sequent calculus for first-order dynamic logic with trace modalities. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 626–641. Springer, 2001.
5. A. Bemporad, A. Bicchi, and G. Buttazzo, editors. *Hybrid Systems: Computation and Control, 10th International Conference, HSCC 2007, Pisa, Italy, Proceedings*, volume 4416 of *LNCS*. Springer, 2007.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
7. W. Damm, H. Hungar, and E.-R. Olderog. On the verification of cooperating traffic agents. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *FMCO*, volume 3188 of *LNCS*, pages 77–110. Springer, 2003.

8. J. M. Davoren, V. Coulthard, N. Markey, and T. Moor. Non-deterministic temporal logics for general flow systems. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*, pages 280–295. Springer, 2004.
9. J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, July 2000.
10. E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
11. E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.
12. J. Faber and R. Meyer. Model checking data-dependent real-time properties of the European Train Control System. In *FMCAD*, pages 76–77. IEEE Computer Society Press, Nov 2006.
13. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic logic*. MIT Press, 2000.
14. T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.
15. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.
16. D. Hutter, B. Langenstein, C. Sengler, J. H. Siekmann, W. Stephan, and A. Wolpers. Deduction in the verification support environment (VSE). In M.-C. Gaudel and J. Woodcock, editors, *FME*, volume 1051 of *LNCS*, pages 268–286. Springer, 1996.
17. D. Leivant. Partial correctness assertions provable in dynamic logics. In I. Walukiewicz, editor, *FoSSaCS*, volume 2987 of *LNCS*, pages 304–317. Springer, 2004.
18. V. Mysore, C. Piazza, and B. Mishra. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In D. Peled and Y.-K. Tsay, editors, *ATVA*, volume 3707 of *LNCS*, pages 217–233. Springer, 2005.
19. A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. 2007.
20. A. Platzer. Differential logic for reasoning about hybrid systems. In Bemporad et al. [5], pages 746–749.
21. A. Platzer. A temporal dynamic logic for verifying hybrid system invariants. Reports of SFB/TR 14 AVACS 12, February 2007. ISSN: 1860-9821, available at http://www.avacs.org.
22. A. Platzer. Towards a hybrid dynamic logic for hybrid dynamic systems. In P. Blackburn, T. Bolander, T. Braüner, V. de Paiva, and J. Villadsen, editors, *Proc., LICS International Workshop on Hybrid Logic, 2006, Seattle, USA*, ENTCS, 2007.
23. A. Platzer and E. M. Clarke. The image computation problem in hybrid systems model checking. In Bemporad et al. [5], pages 473–486.
24. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.
25. V. R. Pratt. Process logic. In *POPL*, pages 93–100, 1979.
26. C. Zhou, A. P. Ravn, and M. R. Hansen. An extended duration calculus for hybrid real-time systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 36–59. Springer, 1992.