Combining Deduction and Algebraic Constraints for Hybrid System Analysis*

André Platzer

University of Oldenburg, Department of Computing Science, Germany
platzer@informatik.uni-oldenburg.de

Abstract. We show how theorem proving and methods for handling real algebraic constraints can be combined for hybrid system verification. In particular, we highlight the interaction of deductive and algebraic reasoning that is used for handling the joint discrete and continuous behaviour of hybrid systems. We illustrate proof tasks that occur when verifying scenarios with cooperative traffic agents. From the experience with these examples, we analyse proof strategies for dealing with the practical challenges for integrated algebraic and deductive verification of hybrid systems, and we propose an iterative background closure strategy.

Keywords: modular prover combination, analytic tableaux, verification of hybrid systems, dynamic logic

1 Introduction

Safety-critical systems occurring in traffic scenarios [9, 27] often are hybrid systems [17, 11], i.e., they combine discrete and continuous behaviour. Discrete behaviour typically originates from a digital controller, which regulates driving and switches to various modes in order to react to changes in the traffic situation. Continuous behaviour is more inherent in the physical process dynamics and results from continuous changes of quantities such as positions over time. Models to describe interacting dynamics use differential equations for continuous evolution and use discrete jumps for discrete state changes [17, 24, 20].

Most verification tools for hybrid systems such as HyTech [3], CheckMate [25], or PHAVer [14], follow the model checking paradigm [7] and work by successive computation of images under hybrid transitions [23].

Because of intricacies of complex continuous dynamics, numerical issues during computations, and general limits of numerical approximation [23], hybrid system model checkers are still much more successful in falsification than in verification. In this work, we are primarily interested in verifying hybrid systems rather than finding bugs. Consequently, we favour a fully symbolic technique, and we follow a deductive approach. Further, unlike model checking, deductive techniques support free parameters [11, 20], which occur in our applications.

We have introduced a family of logics for deductive verification of hybrid systems [20, 22, 21]. We have introduced [20] a dynamic logic $\mathsf{d}\mathcal{L}$ and a calculus

* This research was supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center (SFB/TR 14 AVACS, see www.avacs.org).

for verifying hybrid systems. We have also presented [22] an extension with nominals to investigate compositionality. Moreover, we have introduced [21] a temporal extension for verifying correct behaviour at intermediate states.

While the theoretical background and technical details of our approach can be found in [20, 22, 21], here we discuss the practical aspects of combining deduction and algebraic constraints techniques. In particular, we highlight the principles how both techniques interact for verifying hybrid systems. For this, we analyse the degrees of freedom in implementing our calculus in terms of the nondeterminisms of our proof procedure. We illustrate the impact that various choices of proof strategies have on the overall performance. For hybrid system verification, we observe that the nondeterminisms in the interaction between deductive and real algebraic reasoning have considerable impact on the practical feasibility. In this paper, we analyse and explain the causes and consequences of this effect and propose a proof strategy that avoids these complexity pitfalls.

In this paper, we study the modular combination in the $\mathsf{d}\mathcal{L}$ calculus. Our observations are of more general interest, though, and we conjecture that similar results hold for other tableaux prover combinations of logics with interpreted function symbols that are handled using background decision procedures for computationally expensive theories including real arithmetic, approximations of natural arithmetic, or arrays.

Related Work. There are a selected number of logics dedicated to hybrid systems [24, 11, 28]. They focus on other aspects like topological aspects [11] or parallel composition [24], and they do not provide calculi with a constructive integration of arithmetic reasoning that can be used easily for practical verification. Our calculus, however, can be used for verifying actual operational hybrid system models [20, 21], which is of considerable practical interest [9, 27, 17, 11].

A few other approaches [19, 1] use deduction for verifying hybrid systems and actually integrate arithmetic reasoning in STeP [19] or in PVS [1], respectively. Their working principle is, however, quite different from ours. Given a hybrid automaton [17] and a global system invariant, they compile, in a single step, a verification condition expressing that the invariant is preserved under all transitions of the hybrid automaton. Hence, the hybrid aspects and transition structure vanish completely before the deduction even start. All that remains is a quantified mathematical formula. In contrast, our dynamic logic works by symbolic decomposition and preserves the transition structure during the proof, which simplifies traceability of results considerably. The structure in this symbolic decomposition can be exploited for deriving invariants or parametric constraints [20, 21]. Consequently, in $\mathsf{d}\mathcal{L}$, invariants do not necessarily need to be given beforehand.

Several other approaches combine deductive and arithmetic reasoning, e.g. [6, 2]. Their focus, however, is on general mathematical reasoning in classes of

higher-order logic and is not tailored to verify hybrid systems. Our work, instead, is intended to make practical verification of hybrid systems possible. For a discussion of work related to the logic $\mathsf{d}\mathcal{L}$ itself, we refer to [20].

Structure of this Paper. In Section 2, we summarise the syntax, semantics, and calculus of the differential logic $\mathsf{d}\mathcal{L}$, that we introduced [20]. In Section 3, we report on the kind of applications that we are interested in, and we illustrate typical proof tasks. In Section 4, we analyse the principles how the $\mathsf{d}\mathcal{L}$ calculus combines deductive with algebraic reasoning and illustrate the consequences of various proof strategies in our applications from Section 3. Finally, we draw conclusions and discuss future work in Section 5.

2 Differential Logic

In this section, we briefly recapitulate the differential logic $\mathsf{d}\mathcal{L}$ that we have introduced [20] and point out the characteristic traits of $\mathsf{d}\mathcal{L}$. We only develop the theory as far as necessary and refer to [21, 20, 22] for more background. The logic $\mathsf{d}\mathcal{L}$ is a dynamic logic [16] with programs extended to hybrid programs [20].

The principle of dynamic logic [16] is to combine system operations and correctness statements about system states within a single specification language. By permitting system operations α as actions of a labelled multi-modal logic, 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 states in which condition ϕ holds. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one (terminating) run of α after which ϕ holds. In $\mathsf{d}\mathcal{L}$, hybrid programs play the role of α.

Hybrid programs generalise discrete programs to hybrid change. In addition to the operations of discrete while programs, they have continuous evolution along differential equations as a fundamental operation. For example, the evolution of a train with constant braking can be expressed with a system action for the differential equation $\ddot{z} = -b$ with second time-derivative $\ddot{z}$ of z.

2.1 Syntax of Differential Logic

Terms and Formulas. The formulas of $\mathsf{d}\mathcal{L}$ are built over a finite set V of real-valued variables and a signature Σ containing the usual function and predicate symbols for real arithmetic, such as $0, 1, +, \cdot, =, \leq, <, \geq, >$. Observe that there is no need to distinguish between discrete and continuous variables in $\mathsf{d}\mathcal{L}$.

The set $\mathrm{Trm}(V)$ of *terms* is defined as in classical first-order logic yielding polynomial expressions. The set $\mathrm{Fml}(V)$ of *formulas* of $\mathsf{d}\mathcal{L}$ is defined as in first-order dynamic logic [16]. That is, they are built using propositional connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\forall, \exists$ (first-order part). In addition, if ϕ is a $\mathsf{d}\mathcal{L}$ formula and α a hybrid program, then $[\alpha]\phi, \langle\alpha\rangle\phi$ are formulas (dynamic part).

Hybrid Programs. In $\mathsf{d}\mathcal{L}$ elementary discrete jumps and continuous evolutions interact using regular control structure to form hybrid programs.

Definition 1 (Hybrid programs). *The set* $\mathrm{HP}(V)$ *of* hybrid programs *is inductively defined as the smallest set such that*

- *If* $x \in V$ *and* $\theta \in \mathrm{Trm}(V)$*, then* $(x := \theta) \in \mathrm{HP}(V)$*.*
- *If* $x \in V$*,* $\theta \in \mathrm{Trm}(V)$*, then* $(\dot{x} = \theta) \in \mathrm{HP}(V)$*.*
- *If* $\chi \in \mathrm{Fml}(V)$ *is a quantifier-free first-order formula, then* $(?\chi) \in \mathrm{HP}(V)$*.*
- *If* $\alpha, \beta \in \mathrm{HP}(V)$ *then* $(\alpha; \beta) \in \mathrm{HP}(V)$*.*
- *If* $\alpha, \beta \in \mathrm{HP}(V)$ *then* $(\alpha \cup \beta) \in \mathrm{HP}(V)$*.*
- *If* $\alpha \in \mathrm{HP}(V)$ *then* $(\alpha^*) \in \mathrm{HP}(V)$*.*

The effect of $x := \theta$ is a discrete jump in state space by an instantaneous assignment. That of $\dot{x} = \theta$ is an ongoing continuous evolution controlled by the differential equation $\dot{x} = \theta$. Systems of differential equations, higher-order derivatives, and evolution invariant regions [20] are defined accordingly.

The test action $?\chi$ is used to define conditions. Its semantics is that of a no-op if χ is true in the current state, and that of a failure divergence blocking all further evolution, otherwise. The non-deterministic choice $\alpha \cup \beta$, sequential composition $\alpha; \beta$ and non-deterministic repetition α^* of hybrid programs are as usual. They can be combined with $?\chi$ to form other control structures, see [16].

2.2 Semantics of Differential Logic

The interpretations of $\mathsf{d}\mathcal{L}$ consist of states assigning real values to state variables, which progress along a sequence of states. A potential behaviour of a hybrid system corresponds to a sequence of states that contain the observable values of system variables during its hybrid evolution. The semantics of a hybrid program α is captured by the state transitions that are possible by running α.

A state is a map $\nu : V \to \mathbb{R}$; the set of all states is denoted by $\mathrm{Sta}(V)$. Further, we use $\nu[x \mapsto d]$ to denote the *modification* of a state ν that is identical to ν except for the interpretation of the symbol x, which is $d \in \mathbb{R}$.

For discrete operations, the semantics, $\rho(\alpha)$, of hybrid program α as a state transition relation in $\mathsf{d}\mathcal{L}$ is as customary in dynamic logic (Def. 3). For continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous system flow respecting the differential equation.

Definition 2 (Valuation of terms and formulas). *For terms and formulas, the* valuation $val(\nu, \cdot)$ *with respect to state* ν *is defined as usual for first-order modal logic (e.g. [16]), i.e., using the following definitions for modal operators*

1. $val(\nu, [\alpha]\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for all* ω *with* $(\nu, \omega) \in \rho(\alpha)$
2. $val(\nu, \langle\alpha\rangle\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for some* ω *with* $(\nu, \omega) \in \rho(\alpha)$

Definition 3 (Semantics of hybrid programs). *The* valuation, $\rho(\alpha)$, of a hybrid program α, *is a* transition relation *on states. It specifies which state ω is reachable from a state ν by operations of the hybrid system α and is defined as*

1. $(\nu,\omega) \in \rho(x := \theta) :\Longleftrightarrow \omega = \nu[x \mapsto val(\nu,\theta)]$
2. $(\nu,\omega) \in \rho(\dot{x} = \theta) :\Longleftrightarrow$ *there is a function* $f : [0,r] \to \mathrm{Sta}(V)$ *with* $r \geq 0$ *such that* $f(0) = \nu, f(r) = \omega$, *and* $val(f(\zeta),x)$ *is continuous in* ζ *on* $[0,r]$ *and has a derivative of value* $val(f(\zeta),\theta)$ *at each time* $\zeta \in (0,r)$. *For* $y \neq x$ *and* $\zeta \in [0,r]$, $val(f(\zeta),y) = val(\nu,y)$. *Systems of differential equations are defined accordingly.*
3. $\rho(?\chi) = \{(\nu,\nu) \ : \ val(\nu,\chi) = true\}$
4. $\rho(\alpha;\beta) = \rho(\alpha) \circ \rho(\beta) = \{(\nu,\omega) : (\nu,z) \in \rho(\alpha), (z,\omega) \in \rho(\beta)$ *for some state* $z\}$
5. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
6. $(\nu,\omega) \in \rho(\alpha^*)$ *iff there are* $n \in \mathbb{N}$ *and* $\nu{=}\nu_0, \ldots, \nu_n{=}\omega$ *with* $(\nu_i,\nu_{i+1}) \in \rho(\alpha)$ *for all* $0 \leq i < n$.

2.3 A Calculus for Differential Logic

In this section, we briefly review the $\mathsf{d}\mathcal{L}$ sequent calculus that we introduced [20]. It can be used for verifying hybrid systems in $\mathsf{d}\mathcal{L}$. With the basic idea being to perform a symbolic evaluation, it successively transforms hybrid programs into logical formulas describing their effects.

The $\mathsf{d}\mathcal{L}$ calculus combines deduction and handling of real algebraic constraints modularly. Simply speaking, the purely deductive part of the $\mathsf{d}\mathcal{L}$ calculus handles the discrete part, whereas the continuous part is tackled by real algebraic constraint techniques. On this basis, hybrid system behaviour of interacting discrete-continuous dynamics is handled by a modular calculus combination [20].

The $\mathsf{d}\mathcal{L}$ sequent calculus is summarised in Fig. 1. 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 $\bigwedge_{\phi\in\Gamma} \phi \ \rightarrow \ \bigvee_{\psi\in\Delta} \psi$. Sequents will be treated as an abbreviation. 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 in tableau-style, that is, starting from the desired conclusion at the bottom (goal) to the premisses (sub-goals).

The rule schemata in Fig. 1 can be applied anywhere in the sequent, in particular after adding an arbitrary context Γ, Δ, see [20] for details. Moreover, the symmetric schemata D1–D10 can be applied on either side of the sequent. Finally, in D7 and D8, the schematic modality $\langle\!|\cdot|\!\rangle$ stands for either $[\cdot]$ or $\langle\cdot\rangle$.

For propositional logic, standard rules P1–P9 are listed in Fig. 1. The other rules transform hybrid programs into simpler logical formulas, thereby relating the meaning of programs and formulas. Rules D1–D7 are as in discrete dynamic logic [16, 5]. D8 uses generalised substitutions [5] for handling discrete change. Unlike in uninterpreted first-order logic [13], quantifiers are dealt with using

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

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

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

$$\text{(D1)}\ \frac{\phi \wedge \psi}{\langle ?\phi \rangle \psi} \qquad \text{(D5)}\ \frac{\phi \vee \langle \alpha; \alpha^* \rangle \phi}{\langle \alpha^* \rangle \phi} \qquad \text{(D9)}\ \frac{\exists t{\geq}0\, \langle x := y_x(t) \rangle \phi}{\langle \dot{x} = \theta \rangle \phi}$$

$$\text{(D2)}\ \frac{\phi \to \psi}{[?\phi]\psi} \qquad \text{(D6)}\ \frac{\phi \wedge [\alpha; \alpha^*] \phi}{[\alpha^*]\phi} \qquad \text{(D10)}\ \frac{\forall t{\geq}0\, [x := y_x(t)] \phi}{[\dot{x} = \theta]\phi}$$

$$\text{(D3)}\ \frac{\langle \alpha \rangle \phi \vee \langle \gamma \rangle \phi}{\langle \alpha \cup \gamma \rangle \phi} \qquad \text{(D7)}\ \frac{[\![\alpha]\!][\![\gamma]\!]\phi}{[\![\alpha; \gamma]\!]\phi} \qquad \text{(D11)}\ \frac{\vdash p \qquad \vdash [\alpha^*](p \to [\alpha]p)}{\vdash [\alpha^*]p}$$

$$\text{(D4)}\ \frac{[\alpha]\phi \wedge [\gamma]\phi}{[\alpha \cup \gamma]\phi} \qquad \text{(D8)}\ \frac{\phi_x^\theta}{[\![x := \theta]\!]\phi}$$

$$\text{(F1)}\ \frac{\mathrm{QE}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \forall x\, \phi} \qquad \text{(F3)}\ \frac{\mathrm{QE}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \exists x\, \phi}$$

$$\text{(F2)}\ \frac{\mathrm{QE}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \exists x\, \phi \vdash \Delta} \qquad \text{(F4)}\ \frac{\mathrm{QE}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \forall x\, \phi \vdash \Delta}$$

Rule D8 is only applicable if the substitution of x by θ in ϕ_x^θ introduces no new bindings. In D9–D10, t is a fresh variable, and, for any v, y_v is the solution of the initial value problem $(\dot{x} = \theta, x(0) = v)$. In F1–F4, x does not occur in Γ, Δ. Further, the $\Gamma_i \vdash \Delta_i$ are obtained from the resulting sub-goals of a side deduction. The side deduction is started from the goal $\Gamma \vdash \Delta, \phi$ at the bottom (or $\Gamma, \phi \vdash \Delta$ for F2 and F4). In the resulting sub-goals $\Gamma_i \vdash \Delta_i$, variable x is assumed to occur in first-order formulas only, as quantifier elimination (QE) is then applicable.

Fig. 1: Rule schemata of the $\mathsf{d}\mathcal{L}$ verification calculus.

quantifier elimination [8] over the reals (QE in F1–F4) in a way that is compatible with dynamic modalities. D9–D10 handle continuous evolution given a first-order definable flow y_x for the differential equation $\dot{x} = \theta$ with symbolic initial value x. D11 is an induction schema with inductive invariant p.

At this point, the full details of how F1–F4 use side deductions to lift quantifier elimination to dynamic logic are not important (they can be found in [20]). What is important to note, however, is that quantifier rules and rules for handling modalities need to interact because the actual constraints on quantified symbols depend on the effect of the hybrid programs within modalities [20]. Thus, at some point, after a number of rule applications that handle the dynamic part, rules F1–F4 will be used to discharge (or at least simplify) a proof obligation over real algebraic or semialgebraic constraints by quantifier elimination [8]. The remaining sub-goals will be analysed further again using dynamic rules. The rules F1–F4 constitute the modular interface that combines deduction for handling dynamic reasoning with algebraic constraint techniques for handling continuous reasoning about $\mathbb{R}$. We discuss the consequences and principles of this combination in Section 4 and analyse proof strategies.

3 Analysis of the European Train Control System

In this section, we report on the applications in safety-critical system verification that we verify in the $\mathsf{d}\mathcal{L}$ calculus, see [21, 20]. We illustrate the typical kinds of proof obligations that occur during our deductive analysis of such hybrid systems. Our experience with verifying these applications forms the basis for our analysis of prover combinations and will be used for illustration in Section 4.

Train Control Applications. In the European Train Control System (ETCS) [9, 20], trains are only allowed to move within their current movement authority block (MA). When their MA is not extended before reaching its end, trains always have to stop within the MA because there can be open gates or other trains beyond. Here, we identify a single component which is most responsible for the hybrid characteristics of safe driving. The speed supervision is responsible for locally controlling the movement of a train such that it always remains within its MA. Depending on the current driving situation, the speed supervision determines a safety envelope s around the train, within which driving is safe, and adjusts its acceleration a in accordance with s (called *correction* in [9]).

We assume that an MA has been granted up to track position m and the train is located at position z, heading with initial speed v towards m. In this situation, $\mathsf{d}\mathcal{L}$ can verify safety properties of speed supervision of the form

$$\psi \rightarrow [(\mathit{corr}; \mathit{drive})^*]z \leq m \tag{1}$$

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

$$\mathit{drive} \equiv \tau := 0; (\dot{z} = v, \dot{v} = a, \dot{\tau} = 1; ?v \geq 0 \wedge \tau \leq \varepsilon)$$

$$\psi \equiv v^2 \leq 2b(m - z) \wedge b > 0 \wedge \epsilon > 0 \ . \tag{2}$$

It expresses that a train will *always* remain within its MA m, assuming a constraint ψ for the parameters. In *corr*, the train corrects its acceleration or brakes with force b (as a failsafe recovery manoeuvre [9]) on the basis of the remaining distance $(m-z)$. Then, the train continues moving according to *drive*. There, the position z of the train evolves according to the system $\dot{z} = v, \dot{v} = a$ (i.e., $\ddot{z} = a$). The evolution stops when the speed v drops below zero (or earlier). Simultaneously, clock τ measures the duration of the current *drive* phase before the controllers react to situation changes (we model this to bridge the gap of continuous-time models and discrete-time control design). Clock τ is reset to zero when entering *drive*, constantly evolves along $\dot{\tau} = 1$, and is bound by the invariant region $\tau \leq \varepsilon$. The effect is that a *drive* phase is interrupted for reassessing the driving situation after at most ε seconds, and the *corr*; *drive* loop repeats.

Parameter Constraint Discovery. In addition to proof tasks for safety verification, the $\mathsf{d}\mathcal{L}$ approach is also useful for parameter constraint discovery. That is,

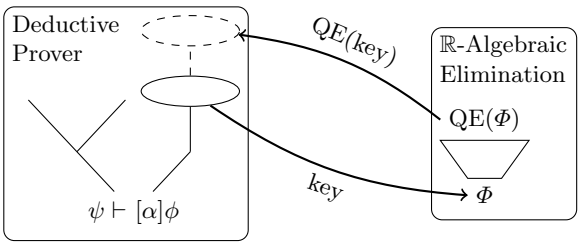


Fig. 2: Prover combination of deduction and algebraic constraint elimination.

instead of starting with a concrete instantiation for ψ in formula (1), the $\mathsf{d}\mathcal{L}$ calculus can be used to identify the required constraints ψ on the free parameters of (1) during the proof. In particular, the constraint ψ in (2) has been discovered by a (semi)automatic discovery process with the $\mathsf{d}\mathcal{L}$ calculus [20].

Finding Inductive Invariants. As a related proof task, the $\mathsf{d}\mathcal{L}$ calculus can be useful for identifying inductive invariants that D11 needs during a proof [20] by analysing partial proofs of individual cases.

4 Combining Deduction and Algebraic Constraints

4.1 Modular Combination of Provers

The principle how the $\mathsf{d}\mathcal{L}$ calculus in Fig. 1 combines deduction technology with methods for handling real algebraic constraints complies with the general background reasoning principles [4, 26, 12]. Unlike in the approaches of Dowek *et al.* [12] and Tinelli [26], the information given to the background prover is not restricted to ground formulas [26] or to atomic formulas as in [12]. From an abstract perspective, the $\mathsf{d}\mathcal{L}$ calculus selects a set Φ of (quantified) formulas from an open branch (Φ is called *key*) and hands it over to the quantifier elimination procedure. The resulting formula obtained by applying QE to Φ is then returned to the main sequent prover as a result, and the main proof continues, see Fig. 2.

In this context, the propositional rules and D-rules (D1–D11) constitute the *foreground rules* in the main prover (left box of Fig. 2) and the arithmetic rules F1–F4 form the set of rules that invoke the *background prover* (right box).

The tableaux procedure [13] for the $\mathsf{d}\mathcal{L}$ calculus is presented in Fig. 3. Observe that the tableaux procedure for our $\mathsf{d}\mathcal{L}$ calculus has a modified set of nondeterministic steps (indicated by $\mathcal{B}$, $\mathcal{M}$, and $\mathcal{F}$, respectively in Fig. 4):

$\mathcal{B}$: *selectBranch*, i.e., which open branch to choose for further rule applications.
$\mathcal{M}$: *selectMode*, i.e., whether to apply foreground $\mathsf{d}\mathcal{L}$ rules (P1–P9 and D1–D11) or background arithmetic rules (F1–F4).
$\mathcal{F}$: *selectFormula*, i.e., which formula(s) to select for rule applications from the current branch in the current mode.

```
while tableaux T has open branches do
  B := selectBranch(T)          (* B–nondeterminism *)
  M := selectMode(B)            (* M–nondeterminism *)
  F := selectFormulas(B,M)      (* F–nondeterminism *)
  if M = foreground then
    B2 := result of applying a D–rule or P–rule to F in B
    replace B by B2 in T
  else
    send key F to background decision procedure QE
    receive result R from QE
    apply a rule F1–F4 to T with QE–result R
  end if
end while
```

Fig. 3: Tableaux procedure for $\mathsf{d}\mathcal{L}$.

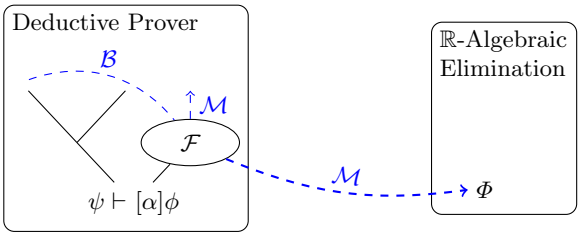


Fig. 4: Nondeterminisms in the tableaux procedure for $\mathsf{d}\mathcal{L}$.

A further, but minor, nondeterminism is whether to expand loops using D6 or to go for an induction by D11. The other $\mathsf{d}\mathcal{L}$ rules do not produce any conflicts once a formula has been selected as they apply to formulas of distinct structures.

At this point, notice that, unlike the classical tableaux procedure [13], we have three rather than four points of nondeterminism, since $\mathsf{d}\mathcal{L}$ does not need closing substitutions. The reason for this is that $\mathsf{d}\mathcal{L}$ has an interpreted domain. Rather than having to try out instantiations that have been determined by unification as in uninterpreted first-order logic [13], we can make use of the structure in the interpreted case of first-order logic over the reals. In particular, arithmetic formulas can be reduced equivalently by QE to simpler formulas in the sense that the quantified symbols no longer occur. As this transformation is an equivalence, there is no loss of information and we do not need to backtrack [13] or simultaneously keep track of multiple local closing instantiations [15].

Despite this, the influence of nondeterminism on the practical prover performance is remarkable. Even though the theory of real arithmetic is decidable by quantifier elimination [8], its complexity is *doubly exponential* in the number of quantifier alternations [10]. While more efficient algorithms exist for linear fragments [18], the practical performance is an issue in nonlinear cases. The computational cost of individual rule applications is quite different from the linear complexity of applying closing substitutions in uninterpreted tableaux.

In principle, exhaustive fair application of background rules by the nondeterminisms $\mathcal{M}$ and $\mathcal{F}$ remains complete for appropriate fragments of $\mathsf{d}\mathcal{L}$. In practice, however, the complexity of real arithmetic quickly makes this naïve approach infeasible. In the remainder of this section, we discuss the consequences of the nondeterminisms and sketch guidelines to overcome the combination problems.

4.2 Nondeterminisms in Branch Selection

In classical uninterpreted tableaux, branch selection has no impact on completeness but can have impact on the proving duration as closing substitutions can sometimes be found much earlier on one branch than on the others. In the interpreted case of $\mathsf{d}\mathcal{L}$, branch selection is even less important. As $\mathsf{d}\mathcal{L}$ has no closing substitutions, there is no direct interference among multiple branches. Branches with (explicitly or implicitly) universally quantified variables have to be closed independently, hence the branch order is not important. For instance, when x is an implicitly universally quantified variable, the branches in the following proof can be handled separately (branches are implicitly combined by conjunction and universal quantifiers distribute over conjunctions):

$$\dfrac{\dfrac{\dfrac{\mathrm{QE}(\forall x\ \dots)}{\Gamma, b>0 \vdash bx^2 \geq 0}\,\mathrm{F1} \qquad \dfrac{\mathrm{QE}(\forall x\ \dots)}{\Gamma, b>0 \vdash bx^4+x^2 \geq 0}\,\mathrm{F1}}{\Gamma, b>0 \vdash (bx^2 \geq 0 \land bx^4+x^2 \geq 0)}\,\mathrm{P5}$$

For existentially quantified variables, the situation is a bit more subtle as multiple branches interfere indirectly in the sense that a simultaneous solution needs to be found for all branches at once. In $\exists v\,(v>0 \land v<0)$, for instance, the two branches resulting from the cases $v>0$ and $v<0$ cannot be handled separately as the existential quantifier claims the existence of a simultaneous solution for $v>0$ and $v<0$, not two different solutions. Thus, when v is an implicitly existentially quantified variable, the branches in the following proof need to synchronise before quantifier elimination is applied:

$$\dfrac{\dfrac{\dfrac{\mathrm{QE}(\exists v\ \dots)}{b>2 \vdash b(v-1)>0}\,\mathrm{D8} \qquad \dfrac{}{b>2 \vdash (v+1)^2+b\epsilon(v+1)>0}\,\mathrm{D8}}{b>2 \vdash [v:=v-1]bv>0 \qquad b>2 \vdash [v:=v+1]v^2+b\epsilon v>0}}{b>2 \vdash ([v:=v-1]bv>0 \land [v:=v+1]v^2+b\epsilon v>0)}$$

The order in which the intermediate steps at two branches are handled has no impact on the proof. Branches like these *synchronise* on an existential variable v in the sense that all occurrences of v need to be first-order for quantifier elimination to work. Consequently, the only fairness assumption for $\mathcal{B}$ is that whenever a formula of a branch is selected that is waiting for synchronisation with another branch to become first-order, then it propagates its rule application to the other

branch. In the above case the left branch synchronises with the right branch on v. Hence, rule F3 can only be applied to $b(v-1) > 0$ on the left branch after D8 has been applied on the right branch to yield first-order occurrences of v.

4.3 Nondeterminisms in Formula Selection

In background proving mode, it turns out that nondeterminism $\mathcal{F}$ is important for the practical performance. When a branch closes or, at least, can be simplified significantly by a quantifier elimination call, then the running time of a single decision procedure call seems to depend strongly on the number of irrelevant formulas that are selected in addition to the relevant ones by $\mathcal{F}$.

Clearly, when Φ is a set of formulas that yields a tautology such that applying F1–F4 closes a branch, then selecting any superset $\Psi \supseteq \Phi$ of Φ from a branch yields the same answer in the end (a sequent forms a disjunction of its formulas hence it can be closed to true when any subset closes). However, the running time until this result will be found in the larger Ψ is strongly disturbed by the presence of complicated additional but irrelevant formulas. From our experience with Mathematica, decision procedures for full real arithmetic seem to be distracted considerably by such irrelevant additional information.

Yet, such additional information accumulates in tableaux procedures quite naturally, because the purpose of a proof branch in $\mathsf{d}\mathcal{L}$ is to keep track of all that is known about a particular (symbolic) case of the system behaviour. Generally, not all of this knowledge finally turns out to be relevant for that case but only plays a role in other branches. Nevertheless, throwing away part of this knowledge light-heartedly would, of course, endanger completeness.

For instance, the safety statement (1) in Section 3 depends on a constraint on the safety envelope s that regulates braking versus acceleration by the condition $m - z \geq s$ in *corr*. A maximal acceleration of a is permitted in case $m - z \geq s$, when adaptively choosing s depending on the current speed v, maximum braking force b, and maximum controller response time ϵ in accordance with the following constraint (which can be discovered by the $\mathsf{d}\mathcal{L}$ calculus [20]):

$$s \;\geq\; \frac{v^2}{2b} + \left(\frac{a}{b}+1\right)\left(\frac{a}{2}\varepsilon^2 + \varepsilon v\right) \quad . \tag{3}$$

This constraint is necessary for some but not for all cases of the safety analysis, though. In the case where the braking behaviour of ETCS is analysed, for instance, the constraint on s is irrelevant, because braking is the safest operation that a train can do to avoid crashing into preceding trains. The unnecessary presence of several quite complicated constraints like, for instance, (3), however, can distract quantifier elimination procedures considerably.

A possible solution for this is to iteratively consider more formulas of the sequent and attempt decision procedure calls. There, only those additional formulas need to be considered that share variables with any of the other selected

formulas. Further, timeouts can be used to discontinue lengthy decision procedure calls and continue along other choices of the nondeterminisms in Fig. 3. For complicated cases with a prohibitive complexity, this heuristic process, which we followed manually, worked well on our examples.

4.4 Nondeterminisms in Mode Selection

In its own right, nondeterminism $\mathcal{M}$ has less impact on the prover performance than $\mathcal{F}$. Every part of a branch could be responsible for closing it. In particular, the foreground closing rule P9 of the main prover can only close branches for comparatively trivial reasons like $b > 0, \epsilon > 0 \vdash \epsilon > 0$. Hence, mode selection has to give a chance to the background procedure every once in a while, following some fair selection strategy. From the observation that some decision procedure calls can run for hours without terminating, we can see, however, that $\mathcal{M}$ needs to be devised with considerable care.

As the reason for closing a branch can be hidden in any part of the sequent, some expensive decision procedure calls can be superfluous if the branch can be closed by continuing $\mathsf{d}\mathcal{L}$ reasoning on the other parts. For instance, if F is some complicated algebraic constraint, decision procedure calls triggered by nondeterminism $\mathcal{M}$ can lead to nontermination within any feasible time for

$$\ldots, \epsilon > 0, m - z \geq s \vdash F, [\mathit{drive}]\epsilon > 0, \ldots$$

Instead, if $\mathcal{M}$ chooses foreground rules, then an analysis of $[\mathit{drive}]\epsilon > 0$ by $\mathsf{d}\mathcal{L}$ rules will quickly discover that the maximum reaction-time ϵ remains constant while driving. Then, this part of the induction step closes without the need to solve constraint F at all. For this reason, proof strategies that eagerly check for closing branches by background procedure calls are not successful in practice.

Unfortunately, converse strategies that strongly favour foreground $\mathsf{d}\mathcal{L}$ rule applications in $\mathcal{M}$, are not appropriate either. There, splitting rules like P5 and P7 can eagerly split the problems onto multiple branches without necessarily making them any easier to solve. If this happens, then slightly different but similar arithmetic problems of about the same complexity need to be solved on multiple branches rather than just one resulting in runtime blow-up.

The reason why this can happen is that there is a syntactic redundancy in the sequent encoding of formulas. For instance, the sets of sequents before and after the following rule application are equivalent:

$$\text{P5}\frac{\psi \vdash v^2 \leq 2b(m-z) \qquad \psi \vdash \epsilon > 0 \qquad \psi \vdash (z \geq 0 \rightarrow v \leq 0)}{\psi \vdash v^2 \leq 2b(m-z) \wedge \epsilon > 0 \wedge (z \geq 0 \rightarrow v \leq 0)}$$

Yet, closing the three sequents above the bar by quantifier elimination is not necessarily easier than the single sequent below (neither conversely). Even worse,

if the sequents close by applying rules to ψ, then similar reasoning has to be repeated for three branches. This threefold reasoning may not be detected as identical when ψ is again split differently on the three resulting branches.

Further, the representational equivalence in sequents is purely syntactic, i.e., up to permutation, the representations share the same disjunctive normal form. In the uninterpreted case, this syntactic redundancy is exploited by the rules P1–P9 in order to transform sequents towards a canonical form with atomic formulas, where partial closing situations are more readily identifiable. In the presence of a background decision procedure, however, reduction to sequents with atomic formulas is no longer necessary as it will be undone when handing the formulas over to the background decision procedure.

Even worse, algebraic constraint handling techniques as in Mathematica can come up with a result that is only a restated version of the input when a selected (open) formula cannot be simplified or closed. For instance, the sequent $z < m \vdash v^2 \leq 2b(m-z)$ "reduces" to $\vdash b \geq v^2/(2m-2z) \lor m \leq z$ without any progress. Such reformulation can easily lead to infinite proof loops when the outcome is split by P8 and again handled by the background procedures.

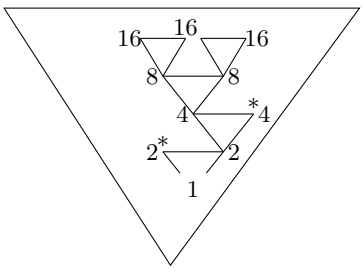
4.5 Iterative Background Closure Strategy

As a strategy to solve the previously addressed issues, we propose the priorities for rule applications in Fig. 5a (with rules at the top taking precedence over rules at the bottom). In this strategy, algebraic constraints are left intact as opposed to being split among multiple branches, because arithmetic rules have a higher priority than propositional rules on first-order constraints. Further, the result of the background procedure is only accepted when the number of variables has decreased to avoid proof loops. Arithmetic background rules have priority 1 *or* 6.

The effect of using priority 1 is that branches are checked eagerly for closing conditions or variable reductions. If reasoning about algebraic constraints does not yield any progress (no variables can be eliminated), then $\mathsf{d}\mathcal{L}$ rules further analyse the system. For this choice, it is important to work with timeouts to prevent lengthy decision procedure calls from blocking $\mathsf{d}\mathcal{L}$ proof progress.

1. *arithmetic rules* F1–F4 if variable eliminated
2. propositional rules P1–P4, P8–P9 on modalities
3. dynamic rules D1–D4, D7–D8
4. dynamic evolution rules D9–D10
5. splitting rules P5–P7 on modalities
6. *arithmetic rules* F1–F4 if variable eliminated
7. propositional rules P1–P9 on first-order formulas

5a: Proof strategy priorities.



5b: Iterative background closure.

This problem is reduced significantly when priority 6 is used for arithmetic rules instead. The effect of priority 6 is that formulas containing modalities are analysed as much as possible before arithmetic reasoning is applied to algebraic constraint formulas. Then, however, the prover can again take too much time analysing the effects of programs on branches which would already close due to simple arithmetic facts like in $\epsilon > 0, \epsilon < 0 \vdash [\alpha]\phi$.

A simple compromise is to use a combination of background rules with priority 1 for quick linear arithmetic [18] and to fall back to expensive quantifier elimination calls for nonlinear arithmetic with priority 6.

As a more sophisticated control strategy on top of the static priorities in Fig. 5a, we propose *iterative background closure* (IBC). There, the idea is to periodically apply arithmetic rules with a timeout T that increases by a factor of 2 after background procedure runs have timed out, see Fig. 5b. Thus, background rules interleave with other rule applications (triangles in Fig. 5b), and the timeout for the sub-goals increases as indicated until the background procedure successfully eliminated variables on a branch (marked by $*$). The effect is that the prover avoids splitting in the average case but is still able to split cases when combined handling turns out to be prohibitively expensive.

5 Conclusions and Future Work

From the experience of using our $\mathsf{d}\mathcal{L}$ calculus [20] for verifying parametric hybrid systems in traffic applications, we have investigated combinations of deductive and algebraic reasoning from a practical perspective. We have analysed the principles of this prover combination, identified the nondeterminisms that remain in the $\mathsf{d}\mathcal{L}$ tableaux procedure, and analysed their impact. We have proposed proof strategies that navigate among these nondeterminisms, including an iterative background closure strategy. Similar to the huge importance of subsumption in resolution, background-style tableaux proving requires quick techniques to rule out branches closing for simple arithmetic reasons. In our preliminary experiments with verifying cooperating traffic agents, our proof strategies significantly reduced the number of interactions and the overall running time significantly.

Future work includes validation of the IBC strategy by experiments in other case studies using a full implementation in our verification tool. Further, we will develop techniques that guide the selection of algebraic constraints by term weight and variable occurrence to discharge simple cases quickly.

Acknowledgements. I thank the anonymous referees for their helpful comments.

References

1. Ábrahám-Mumm, E., Steffen, M., Hannemann, U.: Verification of hybrid systems: Formalization and proof rules in PVS. In: ICECCS, IEEE Computer Society (2001) 48–57

2. Adams, A., Dunstan, M., Gottliebsen, H., Kelsey, T., Martin, U., Owre, S.: Computer algebra meets automated theorem proving: Integrating Maple and PVS. In Boulton, R.J., Jackson, P.B., eds.: TPHOLs. Volume 2152 of LNCS., Springer (2001) 27–42
3. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Trans. Software Eng. **22**(3) (1996) 181–201
4. Beckert, B.: Equality and other theories. In D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J., eds.: Handbook of Tableau Methods. Kluwer, Dordrecht (1999)
5. Beckert, B., Platzer, A.: Dynamic logic with non-rigid functions. In Furbach, U., Shankar, N., eds.: IJCAR. Volume 4130 of LNCS., Springer (2006) 266–280
6. Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuta, E., Vasaru, D.: A survey of the Theorema project. In: ISSAC. (1997) 384–391
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, USA (1999)
8. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**(3) (1991) 299–328
9. Damm, W., Hungar, H., Olderog, E.R.: On the verification of cooperating traffic agents. In de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P., eds.: FMCO. Volume 3188 of LNCS., Springer (2003) 77–110
10. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1/2) (1988) 29–35
11. Davoren, J.M., Nerode, A.: Logics for hybrid systems. Proc. IEEE **88**(7) (Jul 2000) 985–1010
12. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. J. Autom. Reasoning **31**(1) (2003) 33–72
13. Fitting, M.: First-Order Logic and Automated Theorem Proving. Second edn. Springer (1996)
14. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In Morari, M., Thiele, L., eds.: HSCC. Volume 3414 of LNCS., Springer (2005) 258–273
15. Giese, M.: Incremental closure of free variable tableaux. In Goré, R., Leitsch, A., Nipkow, T., eds.: IJCAR. Volume 2083 of LNCS., Springer (2001) 545–560
16. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press (2000)
17. Henzinger, T.A.: The theory of hybrid automata. In: LICS, IEEE Computer (1996) 278–292
18. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. Comput. J. **36**(5) (1993) 450–462
19. Manna, Z., Sipma, H.: Deductive verification of hybrid systems using STeP. In Henzinger, T.A., Sastry, S., eds.: HSCC. Volume 1386 of LNCS., Springer (1998) 305–318
20. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In Olivetti, N., ed.: TABLEAUX. Volume 4548 of LNCS., Springer (2007) 216–232
21. Platzer, A.: A temporal dynamic logic for verifying hybrid system invariants. In Artemov, S., Nerode, A., eds.: LFCS. Volume 4514 of LNCS., Springer (2007) 457–471
22. Platzer, A.: Towards a hybrid dynamic logic for hybrid dynamic systems. In Blackburn, P., Bolander, T., Braüner, T., de Paiva, V., Villadsen, J., eds.: Proc., LICS International Workshop on Hybrid Logic, HyLo 2006, Seattle, USA. Volume 174 of ENTCS. (Jun 2007) 63–77
23. Platzer, A., Clarke, E.M.: The image computation problem in hybrid systems model checking. In Bemporad, A., Bicchi, A., Buttazzo, G., eds.: HSCC. Volume 4416 of LNCS., Springer (2007) 473–486
24. Rönkkö, M., Ravn, A.P., Sere, K.: Hybrid action systems. Theor. Comput. Sci. **290**(1) (2003) 937–973
25. Silva, B.I., Richeson, K., Krogh, B.H., Chutinan, A.: Modeling and verification of hybrid dynamical system using CheckMate. In: ADPM 2000. (2000)
26. Tinelli, C.: Cooperation of background reasoners in theory reasoning by residue sharing. J. Autom. Reasoning **30**(1) (2003) 1–31
27. Tomlin, C., Pappas, G.J., Sastry, S.: Conflict resolution for air traffic management: a study in multi-agent hybrid systems. IEEE Transactions on Automatic Control **43**(4) (April 1998) 509–521
28. Zhou, C., Ravn, A.P., Hansen, M.R.: An extended duration calculus for hybrid real-time systems. In Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H., eds.: Hybrid Systems. Volume 736 of LNCS., Springer (1992) 36–59