
AVACS – Automatic Verification and Analysis of Complex Systems

# REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

## Exact State Set Representations in the Verification of Linear Hybrid Systems with Large Discrete State Space

by

Werner Damm Stefan Disch Hardi Hungar
Swen Jacobs Jun Pang Florian Pigorsch
Christoph Scholl Uwe Waldmann Boris Wirtz

AVACS Technical Report No. 21
June 2007
ISSN: 1860-9821

**Publisher**: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
**Editors**: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
**ATR**s (AVACS Technical Reports) are freely downloadable from `www.avacs.org`


**Author(s) contact**: Werner Damm (`wd@avacs.org`).

# Exact State Set Representations in the Verification of Linear Hybrid Systems with Large Discrete State Space

Werner Damm[2,3], Stefan Disch[1], Hardi Hungar[3], Swen Jacobs[4], Jun Pang[2], Florian Pigorsch[1], Christoph Scholl[1], Uwe Waldmann[4], and Boris Wirtz[2]

[1] Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 51, 79110 Freiburg, Germany
[2] Carl von Ossietzky Universität Oldenburg
Ammerländer Heerstraße 114-118, 26111 Oldenburg, Germany
[3] OFFIS e.V., Escherweg 2, 26121 Oldenburg, Germany
[4] Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany

**Abstract.** We propose algorithms significantly extending the limits for maintaining exact representations in the verification of linear hybrid systems with large discrete state spaces. We use AND-Inverter Graphs (AIGs) extended with linear constraints (LinAIGs) as symbolic representation of the hybrid state space, and show how methods for maintaining compactness of AIGs can be lifted to support model-checking of linear hybrid systems with large discrete state spaces. This builds on a novel approach for eliminating sets of redundant constraints in such rich hybrid state representations by a suitable exploitation of the capabilities of SMT solvers, which is of independent value beyond the application context studied in this paper. We used a benchmark derived from an Airbus flap control system (containing $2^{20}$ discrete states) to demonstrate the relevance of the approach.

## 1 Introduction

We target the verification of safety properties for embedded control applications in the transportation domain. Typical for such applications is a ratio of between 1:5 to 1:10 between the core control algorithms and diagnostic and fault-tolerance measures integrated into the controller, leading to a blow up of the discrete state space against pure control applications often reaching some $10^6$ discrete states. As an example, we analyze a model derived from an Airbus flap controller [13], which on top of its control-loop for flap extraction and retraction is performing envelope protection to prevent loads on flaps possibly causing physical ruptures, and offers extensive monitoring of the health of its sub-systems to e. g. react on loss of hydraulic pressure, rupture of the transmission shaft, or hardware failures. To prove safety of such controllers, we must combine methods for analyzing the pure control part (typically using linear dynamics for design models serving as reference for subsequent implementation steps) with state-space exploration methods dealing with large discrete state spaces. Such applications are out of

reach for existing hybrid verification tools such as CheckMate [26], PHAVer [11], HyTech [15], `d/dt` [5]: while their strength rests in being able to address complex dynamics, they do not scale in the discrete dimension, since modes – the only discrete states considered – are represented explicitly when performing reachability analysis. On the other hand, hardware verification tools such as SMV [20] and VIS [27] scale to extremely large discrete systems, but clearly fail to be applicable to systems with continuous dynamics. To achieve a compact representation of such hybrid state-spaces, we enrich AND-Inverter Graphs (AIGs) with linear constraints. Previous work [23] demonstrated advantages of AIGs over BDDs for representing large discrete state-spaces compactly, due to their higher robustness in handling broad classes of Boolean functions in exhaustive state-space exploration. We lift methods such as test vector generation and SAT checking to detect equivalent (and thus redundant) nodes to the LinAIG level, providing a suite of heuristics including precise checks for equivalent LinAIG nodes using the SMT solver HySAT [10]. Moreover, we provide efficient methods for detecting and eliminating redundant linear constraints from our LinAIG representations which are basically arbitrary boolean combinations of boolean variables and linear constraints. This extends results for eliminating redundant linear constraints from convex polyhedra used by Wang [28] and Frehse [11]. Our approach can be applied to perform backward reachability both for discrete time models (such as reference models for embedded controller implementation) and linear hybrid automata enriched with large discrete state spaces. In the latter case, we exploit the fact that the number of modes of a single controller is typically small (in the order of tens of modes) – this allows us to co-factor the LinAIG representation along modes. For each mode, we use the Loos-Weispfenning quantifier elimination technique for backward evaluation of the symbolic state-space representation along continuous flows. We counteract a worst-case quadratic blow up of linear constraints by tightly integrating redundancy elimination into the quantifier elimination process. Jointly, the presented techniques allow to achieve preciseness while maintaining sufficiently compact representations for the targeted application class.

This paper significantly extends our previous work [7] in adding quantifier elimination and redundancy elimination. The introduction of quantifier elimination was originally motivated by the wish to reduce the diameter of discrete time models. In allowing to fold the effect of large sequences of discretized flows into a single substitution, we accelerate hybrid system verification. This is different from the acceleration by folding hybrid control loops as in [6] which is performed in the world of few discrete states.

The presented methods are orthogonal to and may in the future be combined with abstraction techniques (such as bounding the degree of precision or loosening constraints as in [11]), incorporating robustness [12, 8] or slackness [1, 2] in models allowing precise abstractions by finite grids under robustness respectively slackness assumptions, counter-example guided abstraction refinement as in [24, 17, 25] and techniques such as hybridization [4] for approximate linearization of richer dynamics.

The paper is organized as follows: Sections 2 and 3 give the formal mathematical model and present the backward-reachability algorithm. Sections 4 and 5 are

dedicated to flow extrapolation and redundancy elimination. Evaluation results on the flap controller case study are presented in Section 6.

## 2 System Model

### 2.1 An Informal Description

This section elaborates on the characteristics of the systems to be analyzed, and motivates particular choices incorporated in the formal definition given in the following section. Our definition of hybrid systems can be seen as an extension of linear hybrid automata (LHA) [14] with a set of discrete variables. The state space is spanned by three classes of variables:

- *Continuous variables* represent sensor values, actuator values, plant states, and other real-valued variables used for the modeling of control-laws and plant dynamics.
- *Mode variables* represent a finite (small) set of *modes*, corresponding to the discrete states of an LHA; each mode is uniquely associated with a constant slope for each of the continuous variables, determining how the continuous valuation evolves over time as long as the system is in the given mode.
- *Discrete variables* code states from state-machines, switches, counters, sanity bits of sensor values, etc., and appear in modeling tools typically as bits, range types, or integer sub-ranges. In this paper we will assume some Boolean encoding of these variables. There are additional discrete input variables to our system.

Our models are closed-loop models, combining controller and its controlled plant, hence sensors and actuators are internal continuous variables. Interactions of the environment are only possible through discrete input variables, allowing e. g. to select set-points, and to react to protocol messages. Non-deterministic choices are also modeled using discrete input variables. We remark that employing the construction from [3] permits us to extend our procedure to cope with slope sets bounded by constants which allow for non-determinism in plant dynamics, though we will not provide technical details of this extension in this paper.

The system evolves in alternating between continuous flows, in which time passes and only continuous variables are changed according to their slopes associated with the currently active mode of the system, and sequences of discrete transitions, which happen in zero time. Such discrete transitions update both discrete and continuous variables, and finally select the next active mode. All (discrete) transitions are urgent, eliminating the need to associate state invariants with modes, as in other models of hybrid systems. Discrete inputs enter only in assignments to other discrete variables, i. e. they are disregarded during continuous evolutions. To allow e. g. for periodic sampling of discrete inputs, one can explicitly encode a (continuous) clock within one mode, and test for expiration of the clock-cycle within a transition guard.

### 2.2 Formal Model

We assume disjoint sets of variables $C$, $D$ and $I$. The elements of $C$ are continuous variables, which are interpreted over the reals $\mathbb{R}$. The elements of $D$ and $I$ are discrete variables, where $I$ will be used for inputs. For simplicity, we assume that they are of type boolean and range over the domain $\mathbb{B} = \{0, 1\}$. In the same way we assume that modes are encoded by a set $\mathbf{M} \subseteq \{0, 1\}^l$ of boolean vectors of some fixed length $l$, leading to a set $M$ of $l$ (boolean) mode variables. We denote a valuation of (a subset of) these variables by $(\mathbf{d}, \mathbf{i}, \mathbf{c}, \mathbf{m})$.

A set of valuations (or states) can be represented symbolically using a suitable (quantifier-free) logic formula over $D \cup I \cup C \cup M$. We denote by $\mathcal{B}(D \cup I)$ the set of boolean expressions over $D \cup I$ and by $\mathcal{B}(M)$ the set of boolean expressions over $M$. Here we restrict terms over $C$ to the class of linear terms of the form $\sum \alpha_i c_i + \alpha_0$ with rational constants $\alpha_i$ and $c_i \in C$. Predicates are given by the set $\mathcal{L}(C)$ of linear constraints, they have the form $t \sim 0$, where $\sim \in \{=, <, \leq\}$ and $t$ is a linear term. Finally, $\mathcal{P}(D, C)$ is the set of all boolean combinations of variables from $D$ and linear constraints over $C$.

In the following we use $\xi$ for formulas in $\mathcal{P}(D, C)$, $\theta$ for boolean expressions from $\mathcal{B}(M)$, $g$ for boolean expressions from $\mathcal{B}(D \cup I)$, $t$ for linear terms over $C$, and $\ell$ for linear constraints over $C$.

**Definition 1 (Syntax of CTHSs).** *A* continuous-time hybrid system *CTHS contains six components:*

- $D = \{d_1, \ldots, d_n\}$ *is a finite set of discrete variables,* $I = \{d_{n+1}, \ldots, d_p\}$, $(p \geq n)$ *is a finite set of discrete inputs.*
- $C = \{c_1, \ldots, c_f\}$ *is a finite set of continuous variables.*
- $M = \{m_1, \ldots, m_l\}$ *is a finite set of mode variables,* $\mathbf{M} = \{\mathbf{m}_1, \ldots, \mathbf{m}_k\} \subseteq \{0, 1\}^l$ *is a finite set of modes, each value* $\mathbf{m}_i$ *is associated with a vector* $\mathbf{v}_i \in \mathbb{R}^f$ *of slopes for the variables in* $C$.
- *GC is a global constraint in the form* $g_{gc}(D) \wedge \bigwedge_i \ell_i$ .
- *Init is a set of initial states, given in the form of* $\xi_0 \wedge \theta_0$, *where* $\xi_0 \in \mathcal{P}(D, C)$ *and* $\theta_0 \in \mathcal{B}(M)$.
- *DTrs is the set of discrete transitions; each discrete transition is given as a guarded assignment* $ga_i$ $(i = 1, \ldots, u$ *and* $u \geq 1)$ *in the form*

$$\begin{aligned} \xi_i \wedge \theta_i \rightarrow\ & (d_1, \ldots, d_n) := (g_{i,1}, \ldots, g_{i,n}); \\ & (c_1, \ldots, c_f) := (t_{i,1}, \ldots, t_{i,f}); \\ & (m_1, \ldots, m_l) := \mathbf{m}_{j_i}. \end{aligned}$$

The typical usage of $GC$ is to specify lower and upper bounds for continuous variables in runs to be considered. Note that discrete inputs may appear on the right-hand side of assignments, but not in conditions.

We add the following derived notions and restrictions to the CTHSs we consider:

**Definition 2 (Restrictions on CTHSs).**

- *The guards of the discrete transitions must be mutually exclusive, i. e.* $(\xi_i \wedge \theta_i) \Rightarrow \neg(\xi_j \wedge \theta_j)$ *for* $i \neq j$.

– *For each mode* $\mathbf{m}_i$ *its* boundary condition $\beta_i$ *is given by the cofactor of the disjunction of all discrete transition guards wrt.* $\mathbf{m}_i$.[5] *The boundary conditions have to form closed subsets of* $\mathbb{R}^f$ *for each valuation of variables in* $D$.

**Definition 3 (Semantics of CTHSs).**

– *A* state *of a CTHS is a valuation* $s = (\mathbf{d}, \mathbf{c}, \mathbf{m})$ *of* $D$, $C$ *and* $M$.
– *A discrete transition* $ga_i$ *relates two states* $s \to_i s'$ *iff the guard* $\xi_i \wedge \theta_i$ *is true in* $s$ *and the values in* $s'$ *result from executing the assignments for some valuation* $\mathbf{i}$ *of the input variables.*
– *A state* $s = (\mathbf{d}, \mathbf{c}, \mathbf{m}_i)$ *evolves in time* $\lambda \in \mathbb{R}_{>0}$ *into* $s' = (\mathbf{d}, \mathbf{c} + \lambda \mathbf{v}_i, \mathbf{m}_i)$, *written as* $s \rightsquigarrow^{\lambda} s'$. $s'$ *is a* $\lambda$*-time successor of* $s$ ($s \to^{\lambda} s'$), *if* $s \rightsquigarrow^{\lambda} s'$ *and for all* $s''$ *with* $s = s''$ *or* $s \rightsquigarrow^{\lambda''} s''$*for some* $\lambda'' < \lambda$, *we have* $s'' \models GC$ *and* $s'' \not\models \beta_i$ *(i. e. neither we violate the global constraints nor hit a discrete transition guard along the way).*
– $\to =_{\mathrm{df}} \left(\bigcup_{i=1}^{u} \to_i\right) \cup \left(\bigcup_{\lambda>0} \to^{\lambda}\right)$ *is the transition relation of the CTHS. A* trajectory *is a finite or infinite sequence of states* $(s^j)_{j\geq 0}$ *with* $s^0 \in Init$, *all* $s^j \models GC$, *and* $s^{j-1} \to s^j$ *for each* $j > 0$. *A state is* reachable *if there is a trajectory ending in that state.*

Note that the definition of a time successor makes the discrete transitions *urgent*: they fire once they become enabled. This explains why we do not need invariants of modes while on the other hand we have to require closed sets for boundary conditions.

## 3 Approach

In this section, we describe the main structure of our algorithm. We recall the ingredients which it shares with its predecessor from [7] and point to the new constituents which are detailed in the ensuing sections.

*Overview.* Our algorithm checks whether all reachable states are within a given set of (safe) states $S_0$. To establish this, a backwards fixpoint computation is performed. Starting with the set $S_0$ enriched by all states violating the global constraints, repeatedly the (safe) pre-image is computed until a fixpoint is reached or some initial state is removed from the fixpoint approximant. In the latter case, a state outside of $S_0$ is reachable (while observing the global constraints). So we employ repeatedly

$$\mathit{Safepre}(S) \;=_{\mathrm{df}}\; \{\, s \in S \mid \forall s'.\, s \to s' \Rightarrow s' \in S \,\}\ ,$$

which corresponds to the temporal operator $\mathbf{AX}$. We have chosen the backwards direction, because for discrete transitions the pre-image is expressed essentially by a substitution (see Hoare's program logic [16]).

[5] The cofactor is the partial evaluation of the disjunction wrt. $(m_1, \ldots, m_l) = \mathbf{m}_i$. It does not depend on $M$ anymore.

*Step computation.* We split the computation of *Safepre* into a discrete ($Safepre_D$) and a continuous ($Safepre_C$) part. The computation of $Safepre_D$ using boolean operations, substitutions (both for boolean and real variables), and boolean quantification has been already described in [7]. We will explain our new method to cope with continuous-time evolutions (which did not occur in the discrete-time models of the precursor paper) in detail in Sect. 4.

*Termination.* Since the equivalence of state sets (we deal with boolean combinations of linear constraints, as detailed in the following) is decidable, termination of the algorithms enables us to answer the reachability question. However, it should be noted that termination is not guaranteed – otherwise our algorithm would constitute a solution to an undecidable problem[6]. We expect that the algorithm terminates – in theory – for the great majority of problems coming from applications. We consider complexity the much more relevant challenge in practice. Let us also remark that the implemented fixpoint computation is more elaborated in detail than the somewhat simplified version described here (due to lack of space).

*Representation of state sets.* Our algorithm operates on a specific data structure efficiently implementing formulas from $\mathcal{P}(D, C) \cup \mathcal{B}(M)$. These can be seen as boolean combinations over $D$, $M$ and linear constraints $\mathcal{L}(C)$. We use a set of new (boolean) *constraint variables* $Q$ as encodings for the linear constraints, where each occurring $\ell \in \mathcal{L}(C)$ is encoded by some $q_\ell \in Q$. An important characteristic of our procedure is that the set of constraint variables may grow as the step computation continues, so that new variables are introduced continuously.

For the boolean structure we employ Functionally Reduced AND-Inverter Graphs (FRAIGs) [21, 23]. These are a semi-canonical variant of AND-Inverter Graphs (AIGs) [22, 18]. Basically, they are boolean circuits consisting only of AND gates and inverters. Semi-canonical means that no two nodes represent the same boolean function. In the presence of atoms encoding linear constraints, we call them linear constraint AIGs, or shortly LinAIGs. Their structure is illustrated in Fig. 1.

*Efficiency measures.* We have put much effort into the efficiency of our implementation, in particular into the time efficiency of the routines which keep the representations as small as possible. We briefly summarize some techniques described in more detail in [7], while Sec. 5 presents important improvements. Basically, functional reducedness (generalized from FRAIGs to LinAIGs) can be achieved by checking all pairs of nodes for equivalence, taking the interpretation of constraint variables $q_\ell$ by the corresponding linear constraints $\ell$ into account. This task can be performed by an SMT (SAT modulo theories) solver such as HySAT [10], which combines DPLL with linear programming as a decision procedure. However, it would be much too costly to call HySAT every time a new node

[6] Even if the global constraints define a bounded region, one can straightforwardly encode arithmetic on integers represented as fractions $1/2^n$ of continuous values. This is a common integer representation used in the literature for showing undecidabilities in related domains.

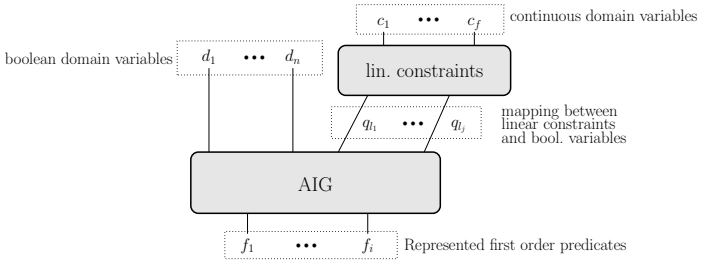


**Fig. 1.** The LinAIG structure

is introduced. Instead, a hierarchy of approximate techniques is used to factor out "easy" problem instances. In first steps purely boolean approximations are employed: If two nodes represent equivalent boolean formulas, we do not need to refer to the definition of the constraint variables. Here we make use of capabilities of FRAIGs, which include local boolean normalization rules, simulation, and SAT checks. Additionally, boolean reasoning is supported by (approximate) knowledge on linear constraints such as implications between constraints. For identifying non-equivalent LinAIG nodes we use test vectors with valuations $\mathbf{c} \in \mathbb{R}^f$, and it proved to be worthwhile to use not only randomly generated test vectors, but also test vectors extracted from failed exact checks done by HySAT (*learning* test vectors). All of these techniques are arranged in a carefully designed and tested strategy of when to apply which technique.

## 4 Flow extrapolation

*Continuous transitions.* In our system model, the time steps only concern the evolutions of continuous variables and leave the discrete part unchanged. For each mode, the continuous safe pre-image $\mathit{Safepre}_C$ can be expressed as a formula with one quantified real variable (time). We will show how to eliminate this quantifier to arrive at a formula which can again be represented by a LinAIG.

Let $\phi(D, M, Q)$ be a representation of a state set. Each valuation $\mathbf{m}_i$ of the mode variables in $M$ encodes a concrete mode with an associated evolution $\mathbf{v}_i$ of $C$ and boundary condition $\beta_i$. Let $\phi_i$ be the cofactor of $\phi$ w. r. t. mode $\mathbf{m}_i$. Thus we have $\phi \Leftrightarrow \bigvee_{i=1}^{k} \phi_i \wedge (m_1, \ldots, m_l) = \mathbf{m}_i$, where each $\phi_i$ is a boolean formula over $D$ and $Q$. For each mode $\mathbf{m}_i$, we must now determine the set of all valuations for which every (arbitrarily long) evolution along $\mathbf{v}_i$ remains in the set of valuations satisfying $\phi_i$, either forever or until it meets a point that satisfies the boundary condition $\beta_i$ or violates the global constraints $GC$. We denote this set by $\mathit{Safepre}_C(\phi_i, \mathbf{v}_i, \beta_i)$. Logically, it can be described by the formula

$$\forall\lambda.\,\big(\lambda < 0 \vee \phi_i(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \neg GC(\mathbf{c}{+}\lambda\mathbf{v}_i) \\ \vee \exists\lambda'.\,(\lambda' \geq 0 \wedge \lambda' < \lambda \wedge (\beta_i(\mathbf{c}{+}\lambda'\mathbf{v}_i) \vee \neg GC(\mathbf{c}{+}\lambda'\mathbf{v}_i)))\big).$$

Under the assumption that the set described by $GC$ is convex, and using the fact that we are only interested in states satisfying $GC$, this formula can be

simplified (modulo $GC$) to

$$\forall\lambda.\,\big(\lambda < 0 \vee \phi_i(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \neg GC(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \exists\lambda'.\,(\lambda' \geq 0 \wedge \lambda' < \lambda \wedge \beta_i(\mathbf{c}{+}\lambda'\mathbf{v}_i))\big).$$

Our task is now to convert this formula over $\lambda$, $\lambda'$, $C$, and $D$ into an equivalent formula over the original variables in $C$ and $D$. If the variables in $C$ occur in $\phi$ and $\beta$ only within linear constraints, then this amounts to variable elimination for linear real arithmetic.[7]

*Test points.* The Loos-Weispfenning test point method [19, 9] eliminates universal quantifiers by converting them into finite conjunctions (and dually, existential quantifiers into finite disjunctions). The method is based on the following observation: Assume that a formula $\psi(x, \vec{y})$ is written as a positive boolean combination of linear constraints $x \sim_i t_i(\vec{y})$ and $0 \sim'_j t'_j(\vec{y})$, where $\sim_i, \sim'_j \in \{=, \neq, <, \leq, >, \geq\}$. Let us keep the values of $\vec{y}$ fixed for a moment. If the set of all $x$ such that $\psi(x, \vec{y})$ does not hold is non-empty, then it can be written as a finite union of (possibly unbounded) intervals, whose boundaries are among the $t_i(\vec{y})$. To check whether $\forall x.\,\psi(x, \vec{y})$ holds, it is therefore sufficient to test $\psi(x, \vec{y})$ for either all upper or all lower boundaries of these intervals. The test values may include $+\infty$, $-\infty$, or a positive infinitesimal $\varepsilon$, but these can easily be eliminated from the substituted formula. For instance, if $x$ is substituted by $t_j(\vec{y}) - \varepsilon$, then both the linear constraints $x \leq t_i(\vec{y})$ and $x < t_i(\vec{y})$ are turned into $t_j(\vec{y}) \leq t_i(\vec{y})$, and both $x \geq t_i(\vec{y})$ and $x > t_i(\vec{y})$ are turned into $t_j(\vec{y}) > t_i(\vec{y})$.

There are two possible sets of test points, depending on whether we consider upper or lower boundaries:

$$TP_1 = \{+\infty\} \cup \{\, t_i(\vec{y}) \mid \sim_i \in \{\neq, >\} \,\} \cup \{\, t_i(\vec{y}) - \varepsilon \mid \sim_i \in \{=, \geq\} \,\}$$
$$TP_2 = \{-\infty\} \cup \{\, t_i(\vec{y}) \mid \sim_i \in \{\neq, <\} \,\} \cup \{\, t_i(\vec{y}) + \varepsilon \mid \sim_i \in \{=, \leq\} \,\}.$$

Let $TP$ be the smaller one of the two sets and let $T$ be the set of all symbolic substitutions $x/t$ for $t \in TP$. Then the formula $\forall x.\,\psi(x, \vec{y})$ can be replaced by an equivalent finite conjunction $\bigwedge_{\sigma \in T} \psi(x, \vec{y})\sigma$. The size of $TP$ is in general linear in the size of $\psi$, so the size of the resulting formula is quadratic in the size of $\psi$. This is independent of the boolean structure of $\psi$ – conversion to CNF is not required. On the other hand, if $\psi$ is a conjunction $\bigwedge \psi_i$, then the test point method can also be applied to each of the formulas $\psi_i$ individually, leading to a smaller number of test points. Moreover, when the test point method transforms each $\psi_i$ into a finite conjunction $\bigwedge \psi_i^j$, then each $\psi_i^j$ contains at most as many linear constraints as the original $\psi_i$, and only the length of the outer conjunction increases.

*Applying the test point method to flow extrapolation.* We have demonstrated above that the safe pre-image $\mathit{Safepre}_C(\phi_i, \mathbf{v}_i, \beta_i)$ of the formula $\phi_i$ is

$$\forall\lambda.\,\big(\lambda < 0 \vee \phi_i(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \neg GC(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \exists\lambda'.\,(\lambda' \geq 0 \wedge \lambda' < \lambda \wedge \beta_i(\mathbf{c}{+}\lambda'\mathbf{v}_i))\big).$$

[7] The variables in $D$ are assumed to remain constant during mode $\mathbf{m}_i$, so boolean expressions over $D$ behave like propositional variables. For simplicity, we will ignore them in the rest of this section.

Assuming that $\phi_i$ equals $\bigwedge_k \phi_{ik}$ and that $\beta_i$ equals $\bigvee_j \beta_{ij}$, we obtain

$$\bigwedge_k \forall\lambda.\,\big(\lambda < 0 \vee \phi'_{ik}(\mathbf{c}{+}\lambda\mathbf{v}_i) \vee \bigvee_j \exists\lambda'.\,(\lambda' \geq 0 \wedge \lambda' < \lambda \wedge \beta_{ij}(\mathbf{c}{+}\lambda'\mathbf{v}_i))\big).$$

where $\phi'_{ik}$ abbreviates $\phi_{ik} \vee \neg GC$. Applying the test point method, we replace the universal and the existential quantifier by a finite conjunction or disjunction using a set of symbolic substitutions $T'_j$ for $\lambda'$ (which depends on $\beta_{ij}$ and $\mathbf{v}_i$) and a set of symbolic substitutions $T_k$ for $\lambda$ (which depends on $\phi_{ik}$, the $\beta_{ij}$, and $\mathbf{v}_i$):

$$\begin{aligned} \mathit{Safepre}_C(\phi_i, \mathbf{v}_i, \beta_i) \;=\; & \bigwedge_k \bigwedge_{\sigma\in T_k} \big((\lambda < 0 \vee \phi'_{ik}(\mathbf{c} + \lambda\mathbf{v}_i))\sigma \\ & \qquad \vee \bigvee_j \bigvee_{\tau\in T'_j} (\lambda' \geq 0 \wedge \lambda' < \lambda \wedge \beta_{ij}(\mathbf{c} + \lambda'\mathbf{v}_i))\tau\sigma\big). \end{aligned}$$

Note that the test point method can work directly on the internal formula representation of LinAIGs – in contrast to the classic Fourier-Motzkin algorithm, there is no need for a costly CNF or DNF conversion before eliminating quantifiers. Moreover, the resulting formulas preserve most of the boolean structure of the original ones: the method behaves largely like a generalized substitution.

*Convexity.* It should be noted that some of the complexity of the general case disappears automatically if the complement of the boundary conditions is convex, that is, if every $\beta_{ij}$ is a single linear inequation. Consider the formula $\bigvee_{\tau\in T'_j} (\lambda' \geq 0 \wedge \lambda' < \lambda \wedge \beta_{ij}(\mathbf{c} + \lambda'\mathbf{v}_i))\tau$. If $\beta_{ij}$ is a single linear inequation, then two test points are always sufficient:[8] (a) If $\beta_{ij}(\mathbf{c} + \lambda'\mathbf{v}_i)$ has the form $\lambda' \leq t(\mathbf{c})$ or $\lambda' < t(\mathbf{c})$, then the test points are $-\infty$ and 0, (b) otherwise, if $\beta_{ij}(\mathbf{c} + \lambda'\mathbf{v}_i)$ has the form $\lambda' \geq t(\mathbf{c})$ or $\lambda' > t(\mathbf{c})$, or if $\lambda'$ is cancelled out completely in $\beta_{ij}(\mathbf{c} + \lambda'\mathbf{v}_i)$, then the test points are $+\infty$ and $\lambda - \varepsilon$. Moreover, if $+\infty$ or $-\infty$ is substituted for $\lambda'$, the conjunction becomes trivially false, so the whole formula is reduced to $0 < \lambda \wedge \beta_{ij}(\mathbf{c})$ in case (a) and to $\lambda > 0 \wedge \beta_{ij}(\mathbf{c}+(\lambda-\varepsilon)\mathbf{v}_i)$ in case (b).

## 5 Redundancy Elimination

Our earlier experiments demonstrated that LinAIGs form an efficient data structure for boolean combinations of boolean variables and linear constraints over real variables [7]. However, in connection with flow extrapolation using Loos-Weispfennig quantifier elimination, one observes that the number of "redundant" linear constraints grows rapidly during the fixpoint iteration of the model checker. For illustration see Fig. 2 and 3, which show a typical example from a model checking run representing a small state set based on two real variables: Lines in Figures 2 and 3 represent linear constraints, and the gray shaded area represents the space defined by some boolean combination of these constraints. Whereas the representation depicted in Fig. 2 contains 24 linear constraints, a closer analysis shows that an optimized representation can be found using only 15 linear constraints as depicted in Fig. 3.

[8] Since we want to eliminate an existential quantifier, we have to use the dual form of the method described above.

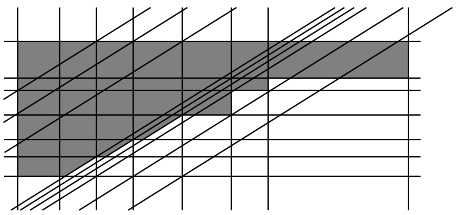

**Fig. 2.** Before redundancy removal

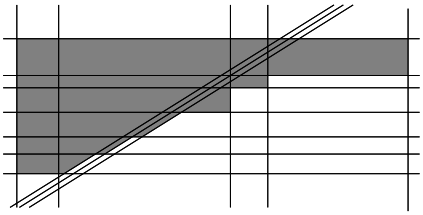

**Fig. 3.** After redundancy removal

*Removing redundant constraints* from our representations turned out to be a crucial task for the success of our methods. It should be noted that, since we represent arbitrary boolean combinations of linear constraints (and boolean variables), this task is not as straightforward as for other approaches such as [14, 11] which represent sets of convex polyhedra, i. e., sets of conjunctions $\ell_1 \wedge \ldots \wedge \ell_n$ of linear constraints. If one is restricted to convex polyhedra, the question whether a linear constraint $\ell_1$ is redundant in the representation reduces to the question whether $\ell_2 \wedge \ldots \wedge \ell_n$ represents the same polyhedron as $\ell_1 \wedge \ldots \wedge \ell_n$, or equivalently, whether $\overline{\ell_1} \wedge \ell_2 \wedge \ldots \wedge \ell_n$ represents the empty set. This question can simply be answered by a linear constraint solver.

For redundancy elimination in our context consider a predicate $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ (represented by a LinAIG) where $b_1, \ldots, b_k$ are boolean variables, $\ell_1, \ldots, \ell_n$ are linear constraints over $C$, and $F$ is a boolean function.

**Definition 4 (Redundancy of linear constraints).** *The linear constraints $\ell_1, \ldots, \ell_r$ $(1 \leq r \leq n)$ are called redundant in the representation of $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ iff there is a boolean function $G$ with the property that $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ and $G(b_1, \ldots, b_k, \ell_{r+1}, \ldots, \ell_n)$ represent the same predicates.*

In order to be able to check for redundancy, we need a disjoint copy $C' = \{c'_1, \ldots, c'_f\}$ of the continuous variables $C = \{c_1, \ldots, c_f\}$. Moreover, for each linear constraint $\ell_i$ $(1 \leq i \leq n)$ we introduce a corresponding linear constraint $\ell'_i$ which coincides with $\ell_i$ up to replacement of variables $c_j \in C$ by variables $c'_j \in C'$. Our check for redundancy is based on the following theorem:

**Theorem 5 (Redundancy check).** *The linear constraints $\ell_1, \ldots, \ell_r$ $(1 \leq r \leq n)$ are redundant in the representation of $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ iff the predicate*

$$F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n) \oplus F(b_1, \ldots, b_k, \ell'_1, \ldots, \ell'_n) \wedge \textstyle\bigwedge_{i=r+1}^{n} (\ell_i \equiv \ell'_i) \qquad (1)$$

*(where $\oplus$ denotes exclusive-or) is not satisfiable by any assignment of boolean values to $b_1, \ldots, b_k$ and real values to the variables $c_1, \ldots, c_f$, $c'_1, \ldots, c'_f$.*

Note that the check from Thm. 5 can be performed by an SMT solver such as HySAT [10]. For completeness a proof of Thm. 5 is given in Appendix A. Here we just give a sketch of the intuition behind Thm. 5.

According to Def. 4 linear constraints $\ell_1, \ldots, \ell_n$ are redundant iff there is a boolean function $G$ such that $G(b_1, \ldots, b_k, \ell_{r+1}, \ldots, \ell_n)$ and $F(b_1, \ldots, b_k, \ell_1, \ldots,$

$\ell_n$) represent the same predicates. Now let us look at $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ as a boolean function $F(b_1, \ldots, b_k, q_{\ell_1}, \ldots, q_{\ell_n})$ with (new) boolean constraint variables $q_{\ell_1}, \ldots, q_{\ell_n}$ and a mapping connecting $q_{\ell_i}$ to $\ell_i$ (just as in our definition of LinAIGs). In comparison to $F$ the required boolean function $G$ must depend only on variables $b_1, \ldots, b_k, q_{\ell_{r+1}}, \ldots, q_{\ell_n}$.

If formula (1) is satisfied by some assignment $\mathbf{d} \in \{0,1\}^k$ to the boolean variables $b_1, \ldots, b_k$, $\mathbf{c} \in \mathbb{R}^f$ to the real variables $c_1, \ldots, c_f$ (which are inputs of linear constraints $\ell_i$), and $\mathbf{c}' \in \mathbb{R}^f$ to the copied real variables $c'_1, \ldots, c'_f$ (which are inputs of copied linear constraints $\ell'_i$), then the first part of formula (1), i. e. $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n) \oplus F(b_1, \ldots, b_k, \ell'_1, \ldots, \ell'_n)$ enforces that the predicate $F$ changes its value if input $\mathbf{c}$ is replaced by input $\mathbf{c}'$ in the corresponding linear constraints. On the other hand, the second part $\bigwedge_{i=r+1}^{n} (\ell_i \equiv \ell'_i)$ enforces that the truth assignment to linear constraints $\ell_{r+1}, \ldots, \ell_n$ does not change when replacing $\mathbf{c}$ by $\mathbf{c}'$. However, since $G$ only depends on variables $b_1, \ldots, b_k$, $q_{\ell_{r+1}}, \ldots, q_{\ell_n}$ (whose truth assignments are not changed), function $G$ "cannot see" the effect of changing $\mathbf{c}$ to $\mathbf{c}'$. Thus $G$ is not able to change its value like $F$ when replacing $\mathbf{c}$ by $\mathbf{c}'$ and therefore it is not able to represent the same predicate as $F$.

Conversely, it can be seen that an appropriate function $G$ can be constructed, when formula (1) is unsatisfiable. When constructing $G$, we use the notion of the *don't care set DC induced by linear constraints* $\ell_1, \ldots, \ell_n$: This don't care set $DC := \{(v_{b_1}, \ldots v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_n}) \mid \nexists (v_{c_1}, \ldots v_{c_f}) \in \mathbb{R}^f \text{ with } \ell_i(v_{c_1}, \ldots, v_{c_f}) = v_{\ell_i} \forall 1 \leq i \leq n\}$ contains all boolean combinations which can not occur due to inconsistent assignments to boolean constraint variables. Whereas for all $(\mathbf{d}, \mathbf{c}) \in \overline{DC} := \{0,1\}^{k+n} \setminus DC$ we have to postulate $G(\mathbf{d}, \mathbf{c}) = F(\mathbf{d}, \mathbf{c})$, the value of $G$ may be chosen arbitrarily for all $(\mathbf{d}, \mathbf{c}) \in DC$, since these values can not occur due to inconsistencies between linear constraints. A closer analysis shows that – under assumption of unsatisfiability of formula (1) – it is indeed possible to define the function values of $G(\mathbf{d}, \mathbf{c})$ for $(\mathbf{d}, \mathbf{c}) \in DC$ in such a way that $G$ will not depend on variables $q_{\ell_1}, \ldots, q_{\ell_r}$. This proves that linear constraints $\ell_1, \ldots, \ell_r$ are then redundant. (For details see Appendix A.)

A straightforward realization of this approach would need a (compact) representation of the don't care set $DC$ in order to compute an appropriate boolean function $G$. However, two interesting observations turn the basic idea into a feasible approach:

1. In general, we do not need the complete set $DC$ for the definition of the boolean function $G$.
2. A representation of a subset of $DC$ which is needed for removing the redundant constraints $\ell_1, \ldots, \ell_r$ is already computed by an SMT solver when checking satisfiability of formula (1).

Details on how the SMT solver internally computes a representation of a sufficient subset of $DC$ and on the actual removal of redundant constraints are included in Appendix B. Our ideas for redundancy detection and removal have been implemented based on the SMT solver HySAT. Experiments given in Section 6 show that integrating redundancy removal is crucial for the success of our methods.

## 6 Experimental Results

Our sample application is derived from a case study for Airbus, a controller for the flaps of an aircraft [13]. The flaps are extended during take-off and landing to generate more lift at low velocity. They are not robust enough for high velocity, so they must be retracted for other periods. It is the controller's task to correct the pilot's commands if he endangers the flaps. Additionally, there is also an extensive monitoring of the health of its sub-systems, checking for instance for hardware failures. The health monitoring system interacts with the flap control by enforcing a more conservative behavior of the control when errors are supposed to be in the system.

The benchmark used here is a simplified version of the full system including the flap controller and a health monitoring system, which is triggered by a timer. The model has three continuous variables: the velocity, the flap angle, and the timer value. Discrete states of the controller and of the health monitoring system contribute to the discrete state space. The discrete state space contains $2^{20}$ discrete states. This size is clearly out of reach for hybrid verification tools known from the literature, which do not scale in the discrete dimension, since modes – the only discrete states considered – are represented *explicitly* when performing reachability analysis.

The safety property to be established for our model is "For the current flap setting, the aircraft's velocity shall not exceed the nominal velocity (w. r. t. the flap position) plus 7 knots". Whether this requirement holds for our model depends on a "race" between flap retraction and speed increase. The controller is correct, if it initiates flap retraction (by correcting the pilot) early enough.

Based on the ideas presented in the previous sections we implemented a prototype model checker using LinAIGs for representing sets of states. Our experiments were run on an AMD Opteron with 2.6 GHz and 16 GB RAM.

Our model checker was able to prove the given safety invariant for the case study in 888.6 CPU seconds. The LinAIG representation had a maximum number of 30887 nodes and a maximum number of 80 linear constraints. The number of flow extrapolation steps using Loos-Weispfennig quantifier elimination was 6, the number of discrete image computation steps performed until reaching the fixpoint was 20. This result clearly demonstrates that our approach is able to successfully verify hybrid systems including discrete parts with state spaces of considerable sizes.

In the following we analyze how the individual ingredients of our method contribute to its overall success. Redundancy elimination turned out to be absolutely necessary to make flow extrapolation using Loos-Weispfennig quantifier elimination feasible. Fig. 4 illustrates the difference between the model checking runs for our case study with and without redundancy removal by plotting the numbers of linear constraints used during the model checking run. Without redundancy removal (dotted line), the number of linear constraints is rapidly increasing up to a number of 1000 linear constraints and 150000 LinAIG nodes in the fourth flow extrapolation.[9] On the other hand, redundancy elimination

[9] Without redundancy removal the remaining two flow extrapolations could not be performed within our timeout of 24 hours.

detects many of the linear constraints to be redundant in our LinAIG representations. Having a closer look at the solid line in Fig. 4 one can identify six groups of three peaks in the number of linear constraints corresponding to six flow extrapolations for three modes, respectively. One notices that redundancy elimination is able to keep the numbers of linear constraints small after Loos-Weispfennig quantifier elimination, so that the number of linear constraints does not exceed 80 during the model checking run. Redundancy elimination removes redundant constraints early and has thus the additional effect that the number of constraints does not blow up due to a series of further substitutions into the removed constraints in following flow extrapolation steps.

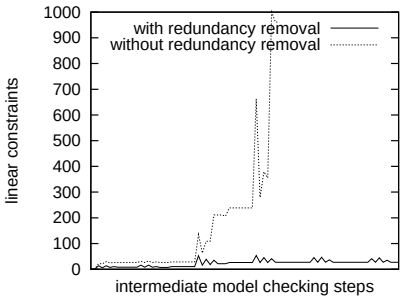


**Fig. 4.** Comparision of the LinAIG evolution with and without redundancy removal

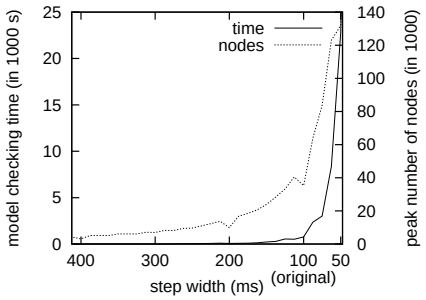


**Fig. 5.** Discrete time example with different time steps

Finally, we want to compare the results for our current system model, which includes continuous evolution of variable length in one operation based on flow extrapolation, to results for a corresponding model with discrete time semantics as presented in [7]. The system model from [7] has no continuous evolution, and discrete steps take fixed time $\delta$. We emphasize that, although time discretized models are widely used in practical applications, they have the problem that unsafe states may be reachable from the initial state, but reachability of these states is not observed due to the time discrete nature of steps. Reducing the width of the discrete time steps can alleviate this problem, but it comes at the cost of a larger number of steps for fixpoint iterations and a larger number of LinAIG nodes for representing sets of states. Our continuous time approach does not show this problem. An analysis of this issue (here done for a flap controller without health monitoring system) is given in Fig. 5. It shows the peak numbers of LinAIG nodes (dotted line) and the run times (solid line) for the example. Here the width of the discrete time step varies between 400 ms and 50 ms. Our analysis clearly shows that run times in the discrete time model largely depend on the width of the time discretization step. At a time step of 400 ms the fixpoint iteration took 5.4 CPU seconds for 9 steps, at 100 ms 763.8 CPU seconds for 33 steps, and at 50 ms 22497 CPU seconds for 65 steps.

This demonstrates a dilemma of the time discretized version: We have to keep the time step small both to be sure not to miss relevant reachable states and to be able to model the system correctly (of course, with discrete time steps of 400 ms we are not able to model realistic controllers sampling every 100

ms). However, decreasing the time step too much may turn the model checking problem intractable. In contrast, in our novel approach we do not work with time discretizations, but we are able to compute continuous evolutions of variable lengths in one operation based on flow extrapolation. Sequences of discrete steps of the previous version [7] where no mode switches are triggered are collapsed into a single symbolic substitution in this way. Note that in the example without health monitoring system only five flow extrapolation steps are needed to reach the fixpoint within a runtime of 27.7 s (whereas for the discrete time model with a time step of 50 ms, e. g., the number of steps amounts to 65 with a run time of 22497 s).

## 7 Conclusion

We consider the tight integration of LinAIGs and HySAT in backward reachability analysis a core technology to address scalability of hybrid system verification methods with large discrete state spaces, and have demonstrated the relevance of the approach using a benchmark derived from an Airbus flap controller. The redundancy elimination technique presented in Section 5 is of independent value and could be integrated in other hybrid verification tools. Next imminent extensions of our approach cover differential inclusions and continuous inputs. We will experiment with incorporating orthogonal extensions to our approach such as exploiting robustness, over-approximation, and counterexample guided abstraction refinement to address richer dynamics and achieve further scalability.

## References


1. M. Agrawal and P. S. Thiagarajan. Lazy rectangular hybrid automata. In *7th Workshop on Hybrid Systems: Computation and Control*, 2004, *LNCS 2993*, pp. 1–15. Springer.
2. M. Agrawal and P. S. Thiagarajan. The discrete time behavior of lazy linear hybrid automata. In *8th Workshop on Hybrid Systems: Computation and Control*, 2005, *LNCS 3414*, pp. 55–69. Springer.
3. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
4. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of non-linear systems. *Acta Informatica*, 43(7):451–476, 2007.
5. E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of the hybrid systems. In *14th Conference on Computer Aided Verification*, 2002, *LNCS 2404*, pp. 365–370. Springer.
6. B. Boigelot and F. Herbreteau. The power of hybrid acceleration. In *18th Conference on Computer Aided Verification*, 2006, *LNCS 4144*, pp. 438–451. Springer.
7. W. Damm, S. Disch, H. Hungar, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Automatic verification of hybrid systems with large discrete state space. In *4th Symposium on Automated Technology for Verification and Analysis*, 2006, *LNCS 4218*, pp. 276–291.
8. W. Damm, G. Pinto, and S. Ratschan. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. *Journal of Foundations of Computer Science*, 18(1):63–86, 2007.

9. A. Dolzmann. *Algorithmic Strategies for Applicable Real Qunantifier Elimination*. PhD thesis, Universität Passau, 2000.
10. M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.
11. G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, 2005.
12. A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.
13. H3 FOMC Team. The flap controller description. `http://www.avacs.org/Benchmarks/Open/flapcontroller.pdf`.
14. T. A. Henzinger. The theory of hybrid automata. In *11th IEEE Symposium on Logic in Computer Science*, 1996, pp. 278–292. IEEE Press.
15. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
16. C. A. R. Hoare. An axiomatic basis for computer programming. *Communication of the ACM*, 12:576–583, 1969.
17. S. Jha, B. Brady, and S. Seshia. Symbolic reachability analysis of lazy linear hybrid automata. Technical report, EECS Dept., UC Berkeley, 2007.
18. A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design*, 21(12):1377–1394, 2002.
19. R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.
20. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
21. A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, EECS Dept., UC Berkeley, 2005.
22. V. Paruthi and A. Kuehlmann. Equivalence checking combining a structural SAT-solver, BDDs, and simulation. In *18th IEEE Conference on Computer Design*, 2000, pp. 459–464. IEEE Press.
23. F. Pigorsch, C. Scholl, and S. Disch. Advanced unbounded model checking by using AIGs, BDD sweeping and quantifier scheduling. In *6th Conference on Formal Methods in Computer Aided Design*, 2006, pp. 89–96. IEEE Press.
24. A. Platzer and E. Clarke. The image computation problem in hybrid systems model checking. In *10th Workshop on Hybrid Systems: Computation and Control*, 2007, *LNCS 4416*, pp. 473–486. Springer.
25. M. Segelken. Abstraction and counterexample-guided construction of $\omega$-automata for model checking of step-discrete linear hybrid models. In *19th Conference on Computer Aided Verification*, 2007, LNCS. Springer. To appear.
26. B. I. Silva, K. Richeson, B. H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using CheckMate. In *4th Conference on Automation of Mixed Processes*, 2000.
27. The VIS Group. VIS: A system for verification and synthesis. In *8th Conference on Computer Aided Verification*, 1996, *LNCS 1102*, pp. 428–432. Springer.
28. F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Transactions on Software Engineering*, 31(1):38–52, 2005.

# A Appendix: Proof of Theorem 5

*"if-part":* For the proof of the "if-part" of Theorem 5 (see page 10) we assume that the predicate from formula (1) is satisfiable and under this assumption we prove that it can not be the case that all linear constraints $\ell_1, \ldots, \ell_r$ are redundant.

Now consider some satisfying assignment to the predicate from formula (1) as follows: $b_1 = v_{b_1}, \ldots, b_k = v_{b_k}$ with $(v_{b_1}, \ldots v_{b_k}) \in \{0,1\}^k$, $c_1 = v_{c_1}, \ldots, c_f = v_{c_f}$ with $(v_{c_1}, \ldots v_{c_f}) \in \mathbb{R}^f$, $c'_1 = v_{c'_1}, \ldots, c'_f = v_{c'_f}$ with $(v_{c'_1}, \ldots v_{c'_f}) \in \mathbb{R}^f$. This satisfying assignment implies a corresponding truth assignment to the linear constraints by $\ell_i(v_{c_1}, \ldots v_{c_f}) = v_{\ell_i}$ $(1 \leq i \leq n)$ with $v_{\ell_i} \in \{0,1\}$ and $\ell'_i(v_{c'_1}, \ldots v_{c'_f}) = v_{\ell'_i}$ $(1 \leq i \leq n)$ with $v_{\ell'_i} \in \{0,1\}$. Since the assignment satisfies formula (1), it holds that

$$F(v_{b_1}, \ldots, v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_n}) \neq F(v_{b_1}, \ldots, v_{b_k}, v_{\ell'_1}, \ldots, v_{\ell'_n}), \quad (a)$$
$$v_{\ell_i} = v_{\ell'_i} \text{ for all } r+1 \leq i \leq n. \quad (b)$$

If the linear constraints $\ell_1, \ldots, \ell_r$ $(1 \leq r \leq n)$ would be redundant in the representation of $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$, then there had to be a boolean function $G$ with $G(b_1, \ldots, b_k, \ell_{r+1}, \ldots, \ell_n)$ and $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ representing the same predicates, and thus

$$G(v_{b_1}, \ldots, v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) = F(v_{b_1}, \ldots, v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_n}), \quad (c)$$
$$G(v_{b_1}, \ldots, v_{b_k}, v_{\ell'_{r+1}}, \ldots, v_{\ell'_n}) = F(v_{b_1}, \ldots, v_{b_k}, v_{\ell'_1}, \ldots, v_{\ell'_n}). \quad (d)$$

Altogether we have

$$\begin{aligned} G(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) &\overset{(c)}{=} F(v_{b_1}, \ldots, v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \\ &\overset{(a)}{\neq} F(v_{b_1}, \ldots, v_{b_k}, v_{\ell'_1}, \ldots, v_{\ell'_r}, v_{\ell'_{r+1}}, \ldots, v_{\ell'_n}) \\ &\overset{(d)}{=} G(v_{b_1}, \ldots, v_{b_k}, v_{\ell'_{r+1}}, \ldots, v_{\ell'_n}) \\ &\overset{(b)}{=} G(v_{b_1}, \ldots, v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \end{aligned}$$

which is a contradiction. Thus, it is not true that all linear constraints $\ell_1, \ldots, \ell_r$ are redundant. □

*"then-part":* Here we assume that there is no satisfying assignment to formula (1) and we show how to construct a boolean function $G$ with $G(b_1, \ldots, b_k, \ell_{r+1}, \ldots, \ell_n)$ and $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ representing the same predicates. To do so, we look at $F(b_1, \ldots, b_k, \ell_1, \ldots, \ell_n)$ as a boolean function $F(b_1, \ldots, b_k, q_{\ell_1}, \ldots, q_{\ell_n})$ with (new) boolean constraint variables $q_{\ell_1}, \ldots, q_{\ell_n}$ and a mapping connecting $q_{\ell_i}$ to $\ell_i$ (just as in our definition of LinAIGs). In comparison to $F$ the required boolean function $G$ depends only on variables $b_1, \ldots, b_k, q_{\ell_{r+1}}, \ldots, q_{\ell_n}$. For defining $G$ we will need the notion of the *don't care set DC induced by linear constraints* $\ell_1, \ldots, \ell_n$: This don't care set

$$DC := \{(v_{b_1}, \ldots v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_n}) \mid \nexists (v_{c_1}, \ldots v_{c_f}) \in \mathbb{R}^f \text{ with } \ell_i(v_{c_1}, \ldots, v_{c_f}) = v_{\ell_i} \forall 1 \leq i \leq n\}$$

contains all boolean combinations which can not occur due to inconsistent assignments to boolean constraint variables.

Whereas for all $(v_{b_1}, \dots v_{b_k}, v_{\ell_1}, \dots, v_{\ell_n}) \in \overline{DC} := \{0,1\}^{k+n} \setminus DC$ we have to postulate $G(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) = F(v_{b_1}, \dots v_{b_k}, v_{\ell_1}, \dots, v_{\ell_n})$ in order to achieve that $G(b_1, \dots, b_k, \ell_{r+1}, \dots, \ell_n)$ and $F(b_1, \dots, b_k, \ell_1, \dots, \ell_n)$ represent the same predicates, the value of $G$ may be chosen arbitrarily for all $\epsilon \in DC$, since these values can not occur due to inconsistencies between linear constraints. Now the idea is to define the function values of $G(\epsilon)$ for $\epsilon \in DC$ in such a way that $G$ will not depend on variables $q_{\ell_1}, \dots, q_{\ell_r}$.

For the definition of $G$ we additionally consider for arbitrary values $(v_{b_1}, \dots, v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \in \{0,1\}^{k+n-r}$ the sets

$$orbit(v_{b_1}, \dots, v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) := \{(v_{b_1}, \dots, v_{b_k}, v_{\ell_1}, \dots, v_{\ell_r}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \mid (v_{\ell_1}, \dots, v_{\ell_r}) \in \{0,1\}^r\}.$$

For each $(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \in \{0,1\}^{k+n-r}$ we define

- If $orbit(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \setminus DC \neq \emptyset$, $G(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) = \delta$ with $F(orbit(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \setminus DC) = \{\delta\}$, $\delta \in \{0,1\}$.
- Otherwise $G(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n})$ is chosen arbitrarily.

It is clear that $G$ does not depend on variables $q_{\ell_1}, \dots, q_{\ell_r}$ by this definition. However it remains to be shown that $G$ is well-defined, i. e., that

$$|F(orbit(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \setminus DC)| = 1.$$

Suppose that $G$ is not well-defined, i. e., there is some set

$$orbit(v_{b_1}, \dots v_{b_k}, v_{\ell_{r+1}}, \dots, v_{\ell_n}) \setminus DC$$

which contains two different elements

$$v^{(1)} := (v_{b_1}, \dots, v_{b_k}, v_{\ell_1}, \dots, v_{\ell_r}, v_{\ell_{r+1}}, \dots, v_{\ell_n})$$

and

$$v^{(2)} := (v_{b_1}, \dots, v_{b_k}, v'_{\ell_1}, \dots, v'_{\ell_r}, v_{\ell_{r+1}}, \dots, v_{\ell_n})$$

with $F(v^{(1)}) \neq F(v^{(2)})$. Since $v^{(1)} \notin DC$, we know that there are $(v_{c_1}, \dots, v_{c_f}) \in \mathbb{R}^f$ with $\ell_i(v_{c_1}, \dots v_{c_f}) = v_{\ell_i}$ for all $1 \leq i \leq n$ and since $v^{(2)} \notin DC$ there are $(v'_{c_1}, \dots, v'_{c_f}) \in \mathbb{R}^f$ with $\ell_i(v'_{c_1}, \dots v'_{c_f}) = v'_{\ell_i}$ for all $1 \leq i \leq r$, $\ell_i(v'_{c_1}, \dots v'_{c_f}) = v_{\ell_i}$ for all $r+1 \leq i \leq n$.

It is easy to see that in this case $b_1 = v_{b_1}, \dots, b_k = v_{b_k}$, $c_1 = v_{c_1}, \dots, c_f = v_{c_f}$, and $c'_1 = v'_{c_1}, \dots, c'_f = v'_{c_f}$ forms a satisfying assignment of formula (1) (with $\ell_i(v_{c_1}, \dots v_{c_f}) = v_{\ell_i}$ for all $1 \leq i \leq r$, $\ell'_i(v'_{c_1}, \dots v'_{c_f}) = v'_{\ell_i}$ for all $1 \leq i \leq r$, $\ell_i(v_{c_1}, \dots v_{c_f}) = \ell'_i(v'_{c_1}, \dots v'_{c_f}) = v_{\ell_i}$ for all $r+1 \leq i \leq n$.). This contradicts our assumption that formula (1) is unsatisfiable, i. e., $G$ is well-defined indeed. □

## B Appendix: Redundancy Removal Using an SMT Solver

In Section 5 we described a method for removing redundant linear constraints from a predicate $F(b_1, \dots, b_k, \ell_1, \dots, \ell_n)$ resulting in a predicate $G(b_1, \dots, b_k,$

$\ell_{r+1}, \ldots, \ell_n$). The basic approach relies on a don't care set $DC$ induced by linear constraints $\ell_1, \ldots, \ell_n$ which is given by

$$DC := \{(v_{b_1}, \ldots v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_n}) \mid \nexists (v_{c_1}, \ldots v_{c_f}) \in \mathbb{R}^f \text{ with } \ell_i(v_{c_1}, \ldots, v_{c_f}) = v_{\ell_i} \forall 1 \leq i \leq n\}.$$

$DC$ contains all boolean combinations which can not occur due to inconsistent assignments to boolean constraint variables.

Instead of computing the complete don't care set $DC$ we rather compute representations of subsets of $DC$ which are needed for removing the redundant constraints $\ell_1, \ldots, \ell_r$. Here we show how these sets can be computed by an SMT solver during the check for satisfiability of formula (1) (see page 10).

In order to explain how the appropriate subset of $DC$ is computed by the SMT solver we need to have a closer look at the function of an SMT solver like HySAT [10]:

Just as in LinAIGs the SMT solver introduces constraint variables $q_{\ell_i}$ for linear constraints $\ell_i$. First, the SMT solver looks for satisfying assignments to the boolean variables (including the constraint variables). Whenever the SMT solver detects a satisfying assignment to the boolean variables, it checks whether the assignment to the constraint variables is consistent, i. e., whether it can be produced by replacement of real variables by reals in linear constraints. This task is performed by a linear constraint solver. If the assignment is consistent, then the SMT solver has found a satisfying assignment, otherwise it continues searching for satisfying assignments to the boolean variables. If some assignment $\epsilon_1, \ldots, \epsilon_m$ to constraint variables $q_{\ell_1}, \ldots, q_{\ell_m}$ was found to be inconsistent, then the boolean "conflict clause" $(q_{\ell_1}^{\overline{\epsilon_1}} + \ldots + q_{\ell_m}^{\overline{\epsilon_m}})$ is added to the set of clauses in the SMT solver to avoid running into the same conflict again. The negation of this conflict clause describes a set of don't cares due to an inconsistency of linear constraints.

Now consider formula (1) (see page 10) which has to be solved by an SMT solver and suppose that the solver introduces boolean constraint variables $q_{\ell_i}$ for linear constraints $\ell_i$ and $q_{\ell'_i}$ for $\ell'_i$ $(1 \leq i \leq n)$. According to Theorem 5, formula (1) is unsatisfiable when linear constraints $\ell_1, \ldots, \ell_r$ are redundant. This means that whenever there is some satisfying assignment to boolean variables (including constraint variables) in the SMT solver, it will be necessarily shown to be inconsistent. The most important observation is now that the negations of conflict clauses due to these inconsistencies include the don't cares needed to compute a boolean function $G$ which is appropriate in the sense of Definition 4:

If there is an orbit $orbit(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ containing two different elements $(v_{b_1}, \ldots, v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ and $(v_{b_1}, \ldots, v_{b_k}, v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ with $F(v_{b_1}, \ldots v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \neq F(v_{b_1}, \ldots v_{b_k}, v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$, then the following assignment to the boolean variables obviously satisfies the boolean abstraction of formula (1) in the SMT solver:
$b_1 = v_{b_1}, \ldots, b_k = v_{b_k}, q_{\ell_1} = v_{\ell_1}, \ldots, q_{\ell_r} = v_{\ell_r}, q_{\ell'_1} = v'_{\ell_1}, \ldots, q_{\ell'_r} = v'_{\ell_r}$,
$q_{\ell_{r+1}} = q_{\ell'_{r+1}} = v_{\ell_{r+1}}, \ldots, q_{\ell_n} = q_{\ell'_n} = v_{\ell_n}$.

However, since we assumed that formula (1) is not satisfiable, this assignment can not be consistent wrt. the interpretation of constraint variables by linear constraints. So there must be an inconsistency in the truth assignment to some linear constraints $\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_n$. Since the linear constraints $\ell_i$ and $\ell'_j$ are

based on disjoint sets of real variables $C = \{c_1, \ldots, c_f\}$ and $C' = \{c'_1, \ldots, c'_f\}$, respectively, it is easy to see that a minimal number of assignments which are already inconsistent performs only assignments to a subset of $\ell_1, \ldots, \ell_n$ or a subset of $\ell'_1, \ldots, \ell'_n$. For this reason, when using the option of minimizing conflict clauses, the SMT solver HySAT will learn a conflict clause whose negation either contains the don't care $(v_{b_1}, \ldots, v_{b_k}, v_{\ell_1}, \ldots, v_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ or the don't care $(v_{b_1}, \ldots, v_{b_k}, v'_{\ell_1}, \ldots, v'_{\ell_r}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$. Since this consideration holds for all pairs of elements in some orbit $\mathit{orbit}(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ for which $F$ produces different values, this means for the subset $DC' \subseteq DC$ of don't cares detected during the run of the SMT solver: $|F(\mathit{orbit}(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \setminus DC')| = 1$, if $\mathit{orbit}(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n}) \setminus DC' \neq \emptyset$. Considering the definition of $G$ from Sect. 5 it is clear that the SMT solver HySAT thus computes all don't cares which are needed to generate the boolean function $G$ not depending on $q_{\ell_1}, \ldots, q_{\ell_r}$.

A representation of the don't cares needed may be extracted as a disjunction of negated conflict clauses which record inconsistencies between linear constraints. In the end we obtain a boolean function $dc$ from the SMT solver with $ON(dc) \subseteq DC$[10] and $ON(dc)$ contains all needed don't cares.

Finally, an appropriate function $G$ is computed in the following way:

- At first by $G' = F \cdot \overline{dc}$ all don't cares represented by $dc$ are mapped to the function value 0.
- Secondly, we perform a existential quantification of the variables $q_{\ell_1}, \ldots, q_{\ell_r}$ in $G'$: $G = \exists q_{\ell_1}, \ldots, q_{\ell_r} G'$. This existential quantification maps all elements of an orbit $\mathit{orbit}(v_{b_1}, \ldots v_{b_k}, v_{\ell_{r+1}}, \ldots, v_{\ell_n})$ to 1, whenever the orbit contains an element $\epsilon$ with $dc(\epsilon) = 0$ and $F(\epsilon) = 1$. Since due to the argumentation above there is no element $\delta$ in such an orbit with $dc(\delta) = 0$ and $F(\delta) = 0$, $G$ eventually differs from $F$ only for don't cares defined by $dc$.

Altogether $G$ does not depend on variables $q_{\ell_1}, \ldots, q_{\ell_r}$ and it is computed from $F$ only by changes of don't cares defined by $dc$. This proves the correctness of the redundancy removal described above.

[10] $ON(dc) := \{\epsilon \mid dc(\epsilon) = 1\}$