Quasi-Dependent Variables in Hybrid Automata*

Sergiy Bogomolov
Albert-Ludwigs-Universität
Freiburg

Christian Herrera
Albert-Ludwigs-Universität
Freiburg

Marco Muñiz
Albert-Ludwigs-Universität
Freiburg

Bernd Westphal
Albert-Ludwigs-Universität
Freiburg

Andreas Podelski
Albert-Ludwigs-Universität
Freiburg

ABSTRACT

The concept of hybrid automata provides a powerful framework to model and analyze real-world systems. Due to the structural complexity of hybrid systems it is important to ensure the *scalability* of analysis algorithms. We approach this problem by providing an effective generalisation of the recently introduced notion of *quasi-equal* clocks to hybrid systems. For this purpose, we introduce the concept of *quasi-dependent* variables. Our contribution is two-fold: we demonstrate how such variables can be automatically *detected*, and we present a *transformation* leading to an abstraction with a smaller state space which, however, still retains the same properties as the original system. We demonstrate the practical applicability of our methods on a range of industrial benchmarks.

Categories and Subject Descriptors

G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

Keywords

Hybrid systems, quasi-dependent variables, model transformation, abstraction, reachability analysis

1. INTRODUCTION

Real-time systems often employ distributed architectures where every component uses an independent clock for internal purposes. The classes of TDMA- [2] and EPL-based [3] protocols are prominent examples of this particular class of systems. In such settings, the information exchange among the components proceeds periodically; components are assigned to certain time ranges for communication. In the end of every period, the components reset their clocks to ensure correct system behavior. This leads to an exponential blowup of the set of states to be considered due to the interleaving semantics of distributed resets in timed automata.

We have attacked this problem in our previous work [4–6] by proposing the notion of *quasi-equal* clocks for timed automata to efficiently reduce the system complexity. Intuitively, two clocks are *quasi-equal* if they always agree on their values except possibly when those clocks are reset.

Hybrid automata provide an expressive framework to model and analyze real-world systems exhibiting complex continuous behavior described by differential equations. Therefore, the handling of continuous dynamics takes an essential part of the analysis run-time. Due to this reason, the reduction of the number of variables in the system and of interleaving resets might lead to crucial performance improvements. To this end, we propose to move from an *equality* relation between quasi-equal clocks to a general *dependency* relation and thus *quasi-dependent* variables. The notion of quasi-equality is a special case of quasi-dependency of variables.

We present an approach consisting of two parts. First, we detect quasi-dependent variables. Here, in some sense, we transfer the idea of variable relations already explored in program verification [7, 8] into the context of hybrid automata. Second, based on this knowledge, we reduce the original system. This line of research is particularly motivated by industrial time-based protocols where the timers of the individual components advance at *different* rates.

The crux behind our detection algorithm is the application of abstraction techniques while analyzing the relations between variables. Both continuous and discrete evolution can break the quasi-dependency between variables. However, interestingly, we only need to be precise while considering discrete jumps whereas in case of continuous evolution we can just check which quasi-dependencies still hold in the considered location. The intuition is that we can abstract away constraints other than the ones reflecting the quasi-dependencies because if a quasi-dependency holds before computing continuous successors (which is checked without precision loss in the preceding discrete step) then the continuous evolution obeys the quasi-dependency. The quasi-dependent variables can also be detected by a hybrid model checker, however, it would explore the state space of the system in a great detail, whereas our detection algorithm conducts a coarse analysis with the focus on the state space parts relevant for establishing quasi-dependencies. These ideas lead to an abstraction that is rather imprecise, yet

*Partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS (http://www.avacs.org), and by CONACYT (Mexico) and DAAD (Germany).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HSCC'14, April 15–17, 2014, Berlin, Germany.
Copyright 2014 ACM 978-1-4503-2732-9/14/04 ...$15.00.
http://dx.doi.org/10.1145/2562059.2562142.

results in a dramatic performance improvement of quasi-dependent variables detection.

Our transformation replaces quasi-dependent variables by one representative variable and updates the system structure appropriately. Properties of the original system are reflected, that is, a forbidden configuration is reachable in the transformed system if and only if it is reachable in the original system. System complexity is reduced in two ways. First, we reduce the number of variables in the system. This step alone leads to a more efficient system handling as the underlying data structures become more compact. Furthermore, in many cases we can achieve an even larger performance boost by completely avoiding interleavings of resets of quasi-dependent variables.

The paper is organized as follows. Section 2 introduces necessary preliminaries. Section 3 describes and discusses the concept of quasi-dependency. The detection algorithm is discussed and evaluated in Section 4, the transformation in Section 5. We conclude the paper with Section 6.

Related Work.

The state space reduction for different classes of systems by deriving dependencies between their variables has been an active research topic since the inception of the area of verification. In particular, deriving (affine) relationships between variables of a program is classical since [7, 8]. This idea has also been successfully applied to the domain of timed automata. Daws et al. [9] suggest an approach to reduce the number of clocks in a single automaton based on the notion of active and equal clocks. This work was extended by Daws et al. [10] to handle networks of automata. Active clocks were used by Étienne [11] to analyze parametric timed automata. The work [12] proposes an abstraction method for a restricted class of systems with multiple clocks being activated in a sequential manner for some bounded time. Clearly, with our approach we can handle timed automata as a subclass of hybrid automata, however, we impose weaker structural requirements compared to the above mentioned works.

In the scope of hybrid automata, dependency detection and appropriate reduction becomes especially complicated. The main efforts in this area were concentrated towards reducing hybrid automata by finding appropriate (bi-)simulation relations. In particular, Pappas [13] and van der Schaft [14] presented a theoretical framework of bisimulation for linear systems. The idea of simulation was extended to its approximate version by Girard et al. [15] where the distance between the observed behavior of the reduced and original should be bounded. Finally, the notion of approximate simulations was lifted to hybrid automata by Girard et al. [16].

In this paper, we pose weaker requirements on the structure of the reduced system. Departing from strong bisimulation towards *weak* bisimulation between the reduced and original systems allows us to completely eliminate some configurations which are induced by interleavings. In order to ensure the preservation of the property to be checked, we rewrite the property itself. In this way, we access the information which is no longer explicitly encoded in the reduced system.

2. PRELIMINARIES

In the following, we recall the definitions of hybrid automata [17], (non-Zeno) run, and observable behaviour for self-containedness. In addition, we introduce the notions of update point and instantaneous valuations. The latter denotes the non-empty, possibly non-singleton set of valuations observable at a given point in time.

Given a set of real-valued variables Var, we use V to denote the set $(\mathit{Var} \to \mathbb{R})$ of *valuations* of Var. A *hybrid automaton* is a tuple $\mathcal{H} = (\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ where Loc is a finite set of *locations*, Var is a finite set of real-valued variables, Lab is a set of *synchronisation labels* including the *stutter label* $\tau \in \mathit{Lab}$, $\mathit{Edge} \subseteq \mathit{Loc} \times \mathit{Lab} \times 2^{V^2} \times \mathit{Loc}$ is a finite set of directed *edges* including a *stutter edge* (ℓ, τ, id, ℓ) for each location $\ell \in \mathit{Loc}$. An edge (ℓ, a, μ, ℓ') from location ℓ to ℓ' is labelled with a label $a \in \mathit{Lab}$ and a *conditional update* μ. $\mathit{Act} : \mathit{Loc} \to 2^{\mathbb{R}_0^+ \to V}$ is a function which assigns a set of *activities* $f : \mathbb{R}_0^+ \to V$ to each location. The activity sets are time-invariant, i.e., for each $t' \in \mathbb{R}_0^+$, $f \in \mathit{Act}(\ell)$ implies $(f + t) \in \mathit{Act}(\ell)$, where $(f + t)(t') = f(t + t')$. $\mathit{Inv} : \mathit{Loc} \to 2^V$ is a function which assigns an *invariant* $\mathit{Inv}(\ell) \subseteq V$ to each location ℓ, and $\mathit{Init} : \mathit{Loc} \to 2^V$ is a function which assigns an *initial value invariant* $\mathit{Init}(\ell) \subseteq V$ to each location ℓ. Locations ℓ with $\mathit{Init}(\ell) \neq \emptyset$ are called *initial locations*.

A *network* $\mathcal{N} = \{\mathcal{H}_1, \dots, \mathcal{H}_n\}$ is a finite set of hybrid automata each with variables Var and with pairwise disjoint sets of locations. We write $\mathcal{H} \in \mathcal{N}$ if and only if $\mathcal{H} \in \{\mathcal{H}_1, \dots, \mathcal{H}_n\}$. We use $\mathit{Lab}(\mathcal{H}), \mathit{Edge}(\mathcal{H})$, etc. to denote the set of labels, edges, etc. of automaton $\mathcal{H}$.

The *operational semantics* of a network $\mathcal{N}$ is defined by the (labelled) transition system

$$\mathcal{T}(\mathcal{N}) = (\mathit{Conf}(\mathcal{N}), \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$$

where $\mathit{Conf}(\mathcal{N}) = \mathit{Loc} \times V$, and $\Lambda := (\mathbb{R}_0^+ \times \mathit{Act}(\mathcal{N})) \cup \bigcup_{1 \le i \le n} \mathit{Lab}_i$ is the set of transition labels. The set of configurations $\mathit{Conf}(\mathcal{N})$ consists of pairs of *location vectors* $\vec{\ell} = \langle \ell_1, \dots, \ell_N \rangle \in \times_{i=1}^{n} \mathit{Loc}(\mathcal{H}_i)$ and valuations. We write $\ell_{s,i}$, $1 \le i \le N$, to denote the location that automaton $\mathcal{H}_i$ assumes in a configuration $s = \langle \vec{\ell}_s, \nu_s \rangle$ and ν_s to denote $\nu_s|_{\mathit{Var}}$. The set of initial configurations is $C_{ini} = \{\langle \vec{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \mid \nu \in \cap_{1 \le i \le n} \mathit{Init}_i(\ell_i)\}$. There is a *discrete transition* $s \xrightarrow{a} s'$ from configuration $s = \langle (\ell_1, \dots, \ell_n), \nu \rangle$ to configuration $s' = \langle (\ell'_1, \dots, \ell'_n), \nu' \rangle$ if and only if there are edges $(\ell_i, a_i, \mu_i, \ell'_i) \in \mathit{Edge}_i$, $1 \le i \le n$, such that for each i, either $a_i = a$, or $a_i = \tau$ and $a \notin \mathit{Lab}_i$, and the edges are enabled by valuation ν and ν' is an effect of the updates applied to ν, i.e. $(\nu, \nu') \in \bigcap_{1 \le i \le n} \mu_i$, $\nu' \in \bigcap_{1 \le i \le n} \mathit{Inv}(\ell'_i)$. There is a *time transition* $s \xrightarrow{d,f} s'$ with delay $d \in \mathbb{R}_0^+$ from configuration s to configuration s' if and only if action f is in $\bigcap_{1 \le i \le n} \mathit{Act}(\ell_{s,i})$ and $f(0) = \nu_s$, $f(d) = \nu_{s'}$, $\forall 0 \le t \le d \bullet f(t) \in \bigcap_{1 \le i \le n} \mathit{Inv}(\ell_{s,i})$.

Note that multiple automata in a network can take labelled edges simultaneously; asynchronous labelled edges are only possible for local labels, i.e., labels that only occur in the set of labels of a single hybrid automaton $\mathcal{H} \in \mathcal{N}$.

Figure 1 shows an example of a network $\mathcal{N}$ consisting of the hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ with variables x and y. Variables x and y with respective initial values 0 and -3 are set to 3 at the point in time 5. In the strict interleaving

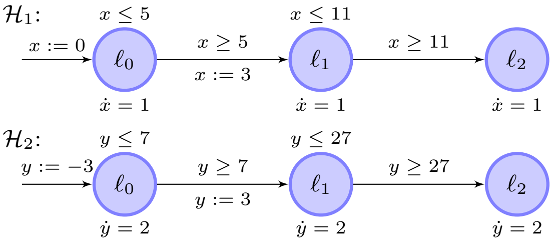

Figure 1: Example of a network $\mathcal{N}$ of hybrid automata.

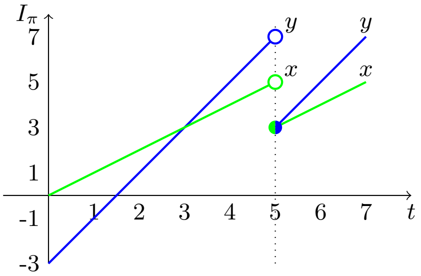

Figure 2: Observable behavior of variables x and y from $\mathcal{N}$.

semantics of networks of hybrid automata, these updates occur one after the other.

A sequence $\pi = \langle \ell_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle, t_1 \xrightarrow{\lambda_2} \cdots$ of time-stamped configurations is called *run* of $\mathcal{H}$ if and only if $\langle \ell_0, \nu_0 \rangle \in C_{ini}$, $t_0 = 0$, and for all $i \in \mathbb{N}_0$, we have $\lambda_{i+1} \in Lab$ and $t_i = t_{i+1}$ or $\lambda_{i+1} \in \{t_{i+1} - t_i\} \times Act$, and the transition relation $\langle \ell_i, \nu_i \rangle \xrightarrow{\lambda_{i+1}} \langle \ell_{i+1}, \nu_{i+1} \rangle$. We use $\Pi(\mathcal{H})$ to denote the set of all runs of $\mathcal{H}$; a hybrid automaton need not have a run.

One run of network $\mathcal{N}$ in Figure 1 is $\pi = \langle (\ell_0, \ell_0), (0, -3) \rangle$, $0 \xrightarrow{5,f} \langle (\ell_0, \ell_0), (5, 7) \rangle, 5 \xrightarrow{\tau} \langle (\ell_1, \ell_0), (3, 7) \rangle$, 5 $\xrightarrow{\tau} \langle (\ell_1, \ell_1), (3, 3) \rangle, 5 \xrightarrow{2,f} \langle (\ell_1, \ell_1), (5, 7) \rangle, 7 \cdots$

In the following, we assume infinite runs without Zeno behaviour, i.e. where the sequence of time stamps diverges. A configuration s is called *reachable* in $\mathcal{H}$ if and only if there is a run $\pi \in \Pi(\mathcal{H})$ such that s occurs in π.

Update Points and Instantaneous Valuations.

The *observable behaviour* of run π is a function $\mathcal{I}^\pi : \mathbb{R}_0^+ \to V$ which assigns to each point in time the valuation defined by an action during a non-zero delay in π, i.e. $\mathcal{I}^\pi(t) := f_{i+1}(t - t_i)$ where $i \in \mathbb{N}_0$ is the (unique) index such that $t_i \le t < t_{i+1}$ and $\lambda_{i+1} = (d_{i+1}, f_{i+1})$. Figure 2 shows the observable behaviour of the network in Figure 1 corresponding to the example run π above.

A point in time $t \in \mathbb{R}_0^+$ is called *update point* of π if and only if a discrete transition occurs in π at t, i.e. if

$$\exists\, i \in \mathbb{N}_0 \bullet \langle \ell_i, \nu_i \rangle, t_i \xrightarrow{a_{i+1}} \langle \ell_{i+1}, \nu_{i+1} \rangle, t_{i+1} \wedge t = t_{i+1}.$$

We use $\mathcal{U}^\pi$ to denote the set of update points of π, the index may be omitted if clear from context. The *instantaneous valuations* of π is a function $\mathcal{V}^\pi : \mathbb{R}_0^+ \to 2^V \setminus \{\emptyset\}$ which assigns to each point in time the (non-empty) set of valuations that occur at that point in time, i.e.

$$\mathcal{V}^\pi(t) := \{\mathcal{I}^\pi(t)\} \cup \{\nu_i \mid t_i = t\}.$$

We use $\mathcal{I}^\pi_x : \mathbb{R}_0^+ \to \mathbb{R}$ and $\mathcal{V}^\pi_x : \mathbb{R}_0^+ \to 2^{\mathbb{R}} \setminus \{\emptyset\}$ to denote the functions which are point-wise defined as $\mathcal{I}^\pi_x(t) := \mathcal{I}^\pi(t)(x)$ and $\mathcal{V}^\pi_x(t) := \{\nu(x) \mid \nu \in \mathcal{V}^\pi(t)\}$ for $t \in \mathbb{R}_0^+$, respectively. We may omit the superscript if it is clear from the context.

In the following we assume that a set of *forbidden configurations* for network $\mathcal{N}$ is given in form of a *configuration formula CF*. *CF* is any logical connection of *basic formulae* over $\mathcal{N}$ which are given by the grammar $\beta ::= \ell \mid \neg\ell \mid \varphi$ where $\ell \in L_i$, $1 \le i \le n$, and where φ is any arithmetical constraint over the variables in Var. A basic formula β is *satisfied* by a configuration $s \in Conf(\mathcal{N})$, denoted by $s \models \beta$, if and only if $\ell_{s,i} = \ell$, $\ell_{s,i} \ne \ell$, and $\nu_s \models \varphi$, respectively. We say *CF* is *reachable* in $\mathcal{N}$ if only if there is a configuration s reachable in $\mathcal{T}(\mathcal{N})$ which satisfies *CF*.

3. QUASI-DEPENDENT VARIABLES

In the following, we define quasi-dependency between variables in terms of the *observable behaviour* of a hybrid automaton. We discuss two alternative characterisations of quasi-dependency and show how quasi-dependency induces an equivalence relation.

Definition 1. (Quasi-f-Dependent Variables) Let $x, y \in Var$ be two variables of the hybrid automaton $\mathcal{H}$. We say x *quasi-depends on* y via function f, denoted by $x \simeq_f y$, if and only if for each run of $\mathcal{H}$ and for each point in time, the value of x in the observable behaviour is the value of f applied to the value of y in the observable behaviour, i.e. if

$$\forall\, \pi \in \Pi(\mathcal{H})\ \forall\, t \in \mathbb{R}_0^+ \bullet \mathcal{I}^\pi_x(t) = f(\mathcal{I}^\pi_y(t)).$$

f is called *dependency function* for x wrt. y. ◇

In the example of the network $\mathcal{N}$ from Figure 1, the variable x quasi-depends on y via dependency function $f(z) = 2z - 3$. At time 5, there are configurations where their values don't satisfy $f(y) = 2x - 3$, but these configurations do not contribute to the observable behaviour as there is no delay possible afterwards (cf. Figure 2).

Note that dependency functions are in general not unique. For example, if only value a occurs in the observable behaviour of variable x and only value b for variable y, then x and y are quasi-dependent via any function f with $f(a) = b$.

We can alternatively state quasi-dependency using instantaneous valuations and update points. The instantaneous values of quasi-dependent variables are related by the dependency function f everywhere except for update points. At update points, the values given by the observable behaviour have to adhere to f.

Lemma 3.1. *Let $x, y \in Var$ be two variables of the hybrid automaton $\mathcal{H}$ such that $x \simeq_f y$ with dependency function f. Then $x \simeq_f y$ if and only if $\forall\, \pi \in \Pi(\mathcal{H}) \bullet$*

$$\mathcal{V}^\pi_x|_{\mathbb{R}_0^+ \setminus \mathcal{U}} = (f \circ \mathcal{V}^\pi_y)|_{\mathbb{R}_0^+ \setminus \mathcal{U}} \wedge \forall\, t \in \mathcal{U} \ \bullet\ \mathcal{I}^\pi_x(t) = f(\mathcal{I}^\pi_y(t))$$

where $f \circ \mathcal{V}^\pi_y$ is the set-valued function which yields the set of values obtained by applying f to all values in $\mathcal{V}_y$. ◇

The following lemma states a third approach to quasi-dependency: two variables are quasi-dependent if the accumulated duration where their values are not related by the dependency function is 0.

Lemma 3.2. *Let $x, y \in Var$ be two variables of the hybrid automaton $\mathcal{H}$. Let $\pi \in \Pi(\mathcal{H})$ be a run of $\mathcal{H}$. Let $\Delta : \mathbb{R}_0^+ \to \{0, 1\}$ be the characteristic function of the points* *in time where the value of x is* not *obtained by applying the dependency function f to the value of y, i.e.*

$$\Delta(t) = \begin{cases} 1, \textit{ if there exist } v \in \mathcal{V}_x^\pi(t), w \in \mathcal{V}_y^\pi(t) \textit{ s.t. } v \neq f(w) \\ 0, \textit{ otherwise.} \end{cases}$$

Then $x \simeq_f y$ *implies* $\displaystyle\int_0^\infty \Delta(t)dt = 0.$ ◇

Note that Lemma 3.2 is only an implication. The opposite direction does not hold, e.g., with actions that non-continuously change their value at isolated points during a time transition. The integral is blind for such changes, in the observable behaviour (and thus for Definition 1) they are present.

Definition 2. (Quasi-Dependent Variables) Let $x, y \in \mathit{Var}$ be variables of hybrid automaton $\mathcal{H}$. We say x *quasi-depends on* y, denoted by $x \equiv y$, if and only if there exist dependency functions f and g such that $x \simeq_f y$ and $y \simeq_g x$. ◇

Quasi-dependency induces an equivalence relation as follows.

Lemma 3.3 (Equivalence Relation). *Let $\mathcal{H}$ be a hybrid automaton. Quasi-dependency $\equiv$ is an equivalence relation on the variables Var of $\mathcal{H}$.* ◇

In the following, we use $\mathcal{QD}_\mathcal{N}$ to denote the set $\{Y \in \mathit{Var}/{\equiv} \mid 1 < |Y|\}$ of equivalence classes of quasi-dependent variables of $\mathcal{N}$ with at least two elements. For each $Y \in \mathcal{QD}_\mathcal{N}$, we assume a designated representative variable denoted by $\mathit{rep}(Y)$. For each $x \in Y$, we use $\mathit{rep}(x)$ to denote $\mathit{rep}(Y)$.

Lemma 3.4. *Let $x, y \in \mathit{Var}$ be variables of hybrid automaton $\mathcal{H}$ such that $x \simeq_f y$ and $y \simeq_g x$. Then f is the inverse of g, i.e.*

$$\forall\, \pi \in \Pi(\mathcal{H})\ \forall\, t \in \mathbb{R}_0^+ \bullet g(f(\mathcal{I}_y^\pi(t))) = \mathcal{I}_y^\pi(t). \qquad ◇$$

Note that the notion of quasi-dependency of variables substantially generalizes the existing notion of quasi-equal clocks from [4]. Firstly, timed automata are a subclass of hybrid automata where all continuous variables increase with rate 1 and can only be assigned to 0. Two clocks in a timed automaton are quasi-equal if and only if they are quasi-dependent via the identity function.

4. QUASI-DEPENDENCY DETECTION

For our detection of quasi-dependent variables in hybrid automata, we consider the well-known symbolic semantics of hybrid automata where sets of valuations are represented by so called *regions*. For the rest of this section, let $\mathcal{H}$ denote the hybrid automaton $(\mathit{Loc}, \mathit{Var}, \mathit{Lab}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$.

Definition 3. (Symbolic Semantics) The *symbolic semantics* of a hybrid automaton $\mathcal{H}$ is defined by the region transition system $\mathcal{T}(\mathcal{H}) = (\mathit{Conf}'(\mathcal{H}), \Longrightarrow, C_{ini})$ where

- $\mathit{Conf}'(\mathcal{H}) = \mathit{Loc} \times (2^V \setminus \{\emptyset\})$ is the set of configurations, consisting of pairs of a location and a set of valuations,
- $C_{ini} = \{\langle \ell, Z\rangle \in \mathit{Conf}'(\mathcal{H}) \mid Z \subseteq \mathit{Init}(\ell)\}$ is the set of initial configurations, and
- $\Longrightarrow$ is the transition relation where
  - there is a *discrete transition* $\langle \ell, Z\rangle \Longrightarrow \langle \ell', Z'\rangle$ induced by edge $(\ell, a, \mu, \ell') \in \mathit{Edge}$ if $Z' = \{\nu' \mid \exists\, \nu \in Z \bullet (\nu, \nu') \in \mu$ and $\nu' \in \mathit{Inv}(\ell')\}$, and
  - there is a *time transition* $\langle \ell, Z\rangle \Longrightarrow \langle \ell, Z'\rangle$ if $Z' = \{\nu' \mid \exists\, f \in \mathit{Act}(\ell), \nu \in Z, t \in \mathbb{R}_0^+ \bullet f(0) = \nu \wedge f(t) = \nu' \wedge \forall\, 0 \leq t' \leq t \bullet f(t') \in \mathit{Inv}(\ell)\}$. ◇

We call configurations where no flow of positive duration is possible, i.e. the only possible flow has duration zero, zero time configurations.

Definition 4. (Zero time configuration) A configuration $\langle \ell, Z\rangle$ is called *zero time*, if and only if the invariant of ℓ does not allow time to elapse from any valuation in Z, i.e. if

$$\begin{aligned} &\forall\, f \in \mathit{Act}(\ell), \nu \in Z, t \in \mathbb{R}_0^+ \bullet f(0) = \nu \wedge f(t) \in \mathit{Inv}(\ell) \\ &\quad \wedge \forall\, 0 \leq t' \leq t \bullet f(t') \in \mathit{Inv}(\ell) \implies t = 0. \end{aligned}$$

We write $\mathsf{zt}(\ell, Z)$ if and only if $\langle \ell, Z\rangle$ is zero time. ◇

Note that the values of quasi-dependent variables are *not* related by the dependency function at most in zero time configurations, and that they may assume any value in zero-time configurations without violating quasi-dependency. Stated the other way around, quasi-dependecy is violated in each non-zero time configuration where the values of the variables are not related by the given dependency function. Thus our analysis tries to establish that for each transition sequence involving only zero-time configurations, whenever two variables are related by the quasi-dependency function when entering a zero-time configuration, they will be related by the quasi-dependency function again when entering a non zero-time configuration.

To this end, our analysis uses precise values for variables only during zero-time configurations. For non zero-time configurations, the *relax operator* only preserves the information which variables are related by which dependency function.

Definition 5. (Relax operator) The *relax operator* rlx applied to the configuration $\langle \ell, Z\rangle$ of $\mathcal{H}$ over-approximates the region Z by a conjunction of the quasi-dependent functions it entails. Formally, $\mathsf{rlx}(\ell, Z) := \langle \ell, Z'\rangle$ where

$$\begin{aligned} Z' = \bigwedge\{y = f(x) \mid\ & x, y \in \mathit{Var} \text{ and } \forall\, g \in \mathit{Act}(\ell), \nu \in Z \bullet \\ & g(0)(x) = \nu(x) \wedge g(0)(y) = \nu(y) \\ & \implies \forall\, t \in \mathbb{R}_+^0 \bullet g(t)(y) = f(g(t)(x))\}. \end{aligned}$$

Our abstraction function (see below) depends on whether the given configuration is zero time. If it is zero time, then the abstraction goes as precise as possible by preserving all information in the given region. If the configuration is not zero time, then the abstraction will preserve only quasi-dependency relations. In practice, the number of zero time configurations is often small compared to the number of non-zero time configurations. Since for every non-zero time configuration the abstraction function maps given regions to much bigger ones, the size of the resulting abstract system is then much smaller than the corresponding concrete one.

Definition 6. (Zero time abstraction function) The abstraction function α_{zt} applied to a configuration $\langle \ell, Z\rangle$ leaves

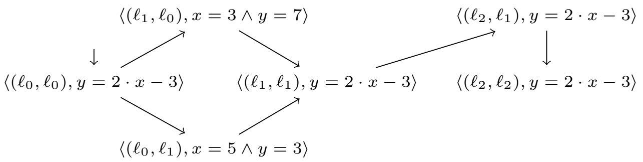
$\langle(\ell_1,\ell_0), x=3\wedge y=7\rangle$
$\langle(\ell_2,\ell_1), y=2\cdot x-3\rangle$
$\langle(\ell_0,\ell_0), y=2\cdot x-3\rangle$
$\langle(\ell_1,\ell_1), y=2\cdot x-3\rangle$
$\langle(\ell_2,\ell_2), y=2\cdot x-3\rangle$
$\langle(\ell_0,\ell_1), x=5\wedge y=3\rangle$

Figure 3: Abstract region graph of network $\mathcal{N}$ from Figure 1.

the configuration unaffected if the configuration is zero time and it abstracts it using the relax operator otherwise, i.e.

$$\alpha_{\mathsf{zt}}(\langle\ell,Z\rangle) := \begin{cases}\langle\ell,Z\rangle & \text{if } \mathsf{zt}(\ell,Z)\\ \mathsf{rlx}(\ell,Z) & \text{otherwise.}\end{cases}$$

The following lemma ensures that the abstraction function applied to a configuration produces a configuration with a bigger or equal region.

Lemma 4.1. *The abstraction function is increasing and idempotent, i.e., let $\langle\ell,Z\rangle \in Conf'(\mathcal{H})$ be a configuration.*

1. *$Z \subseteq Z'$ if $\langle\ell,Z'\rangle = \alpha_{\mathsf{zt}}(\langle\ell,Z\rangle)$, and*
2. *$\alpha_{\mathsf{zt}}(\alpha_{\mathsf{zt}}(\langle\ell,Z\rangle)) = \alpha_{\mathsf{zt}}(\langle\ell,Z\rangle)$.* ◇

Our technique generates an abstract region graph, where quasi-dependent variables can be soundly detected by traversing the graph. The following definition is constructive in the sense that it tells how to construct the abstract region graph: compute the initial configurations firstly and then for each configuration, compute its successors using the abstraction function α_{zt}. It is important to note that there are only discrete transitions in the abstract region graph. This is because, by definition of the abstraction function there are two possibilities: either a configuration is zero time, meaning it does not have time successors, or the configuration is non-zero time. In the latter case the relax operator is applied, and yields a region which includes all time successors.

Definition 7. (Abstract region graph) $\mathcal{H}$ induces the abstract region graph $\mathcal{T}^{\#}(\mathcal{H}) = (Conf'(\mathcal{H}), \Longrightarrow^{\#}, C_0^{\#})$ where

- $\Longrightarrow^{\#} \subseteq Conf'(\mathcal{H}) \times Conf'(\mathcal{H})$ is the transition relation. There is a transition $\langle\ell,Z\rangle \Longrightarrow^{\#} \alpha_{\mathsf{zt}}(\langle\ell',Z'\rangle)$ if there is an edge $(\ell,a,\mu,\ell') \in E$ such that $Z' = \{\nu' \mid \exists\,\nu \in Z \bullet (\nu,\nu') \in \mu \text{ and } \nu' \in Inv(\ell')\}$ is not empty, and
- $C_0^{\#} = \{\alpha_{\mathsf{zt}}(\langle\ell,Z\rangle) \mid \langle\ell,Z\rangle \in Conf'(\mathcal{H}) \wedge Z = Init(\ell)\}$ is the set of initial configurations. ◇

Figure 3 shows the abstract region graph for the running example. The abstract region graph simulates the region transition system thus quasi-dependency detection on the abstract region graph is sound.

Definition 8. (Simulation relation) A simulation relation $\leq$ for two region transition systems $\mathcal{T}_i = (Conf'(\mathcal{H}), \Longrightarrow_i, C_{ini_i})$, $i \in \{1,2\}$, is a binary relation on $Conf'(\mathcal{H})$ satisfying the following properties:

1. for all $c_{ini_1} \in C_{ini_1}$ there exists c_{ini_2} with $c_{ini_1} \leq c_{ini_2}$,
2. $\langle\ell_1,Z_1\rangle \leq \langle\ell_2,Z_2\rangle$ implies $\ell_1 = \ell_2$, $Z_1 \subseteq Z_2$, and
3. $\langle\ell_1,Z_1\rangle \leq \langle\ell_2,Z_2\rangle$ and $\langle\ell_1,Z_1\rangle \Longrightarrow_1 \langle\ell_1',Z_1'\rangle$ implies that there exists a configuration $\langle\ell_2',Z_2'\rangle$ such that $\langle\ell_2,Z_2\rangle \Longrightarrow_2 \langle\ell_2',Z_2'\rangle$ and $\langle\ell_1',Z_1'\rangle \leq \langle\ell_2',Z_2'\rangle$.

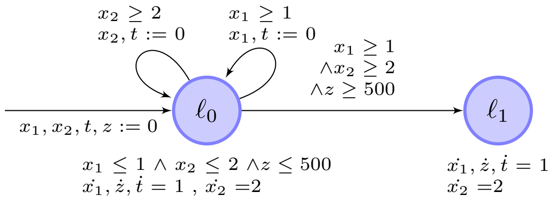

Figure 4: Class B benchmark: $x_1 \simeq_f x_2$ via $f(x) = 0.5 \cdot x$.

We say that $\mathcal{T}_2$ *simulates* $\mathcal{T}_1$ if and only if there exists a simulation relation for $\mathcal{T}_1, \mathcal{T}_2$. ◇

Theorem 4.2. *$\mathcal{T}^{\#}(\mathcal{H})$ simulates $\mathcal{T}(\mathcal{H})$.* ◇

Quasi-dependency of two variables x and y via depedency function f is checked on the abstract region graph $\mathcal{T}^{\#}(\mathcal{H})$ by checking, for each non-zero time configuration $\langle\ell,Z\rangle$ of $\mathcal{T}^{\#}(\mathcal{H})$ whether Z entails $x = f(y)$. If there is a (possibly spurious) counter-example for the quasi-dependency, the corresponding application of the relax operator removes this dependency from the region.

Note that Definition 5 needs a given dependency function candidate f. For *linear* hybrid automata, also known as *multi-rate* automata, dependency function candidates can easily be constructed from the locations' actions (rates) and initial values of variables. Further note that two variables x, y having the same rate in each location does not imply quasi-dependency between x and y; similarly, having different rates for x and y in some (even reachable) location does not imply that there is no quasi-dependency between them.

4.1 Experimental Results

We have implemented our approach in our tool sAsEt. We have implemented regions as predicates over reals and use the SMT solver Z3 [18] to compute the result of the relax operator, whether a configuration is zero time, and whether guards are satisfied. As Z3 is restricted to linear arithmetics, the scope of our implementation is restricted to *linear* hybrid automata. sAsEt is based on the Jahob system [19]. It takes as input a network of hybrid automata in the SpaceEx [20] format and computes successors on the fly.

Table 1 compares the run-time of sAsEt with SpaceEx on two artificial benchmarks which represent two typical cases for quasi-dependent variables in hybrid models. As SpaceEx does not support the detection of quasi-dependency directly, we introduced an auxiliary continuous variable t with rate $\dot{t} = 1$ in all locations which is set to 0 on all edges. Then checking for quasi-equality, e.g., for the running example amounts to checking $(y < 2x+3 \vee y > 2x+3) \wedge t > 0$.

In the Class A benchmark, there exists an ordering on the assignments of quasi-dependent variables. This yields a linear increase of the number of zero time configurations. Both sAsEt and SpaceEx scale, but the non quasi-dependent variables are early abstracted away by our approach yielding a very small abstract region graph in comparison with the one obtained by SpaceEx. In the Class B benchmark (cf. Figure 4), there is non-determinism between the assignments of quasi-dependent variables which yields an exponential growth of the number of zero time configurations in

[1] AMD Opteron 6174, 2.2GHz, 16GB, Linux 2.6.32-5-amd64; SpaceEx 0.9.7c (32bit); t(s) gives user+system time.

Cl.	n	sAsEt			SpaceEx	
		SMT-calls	states	t (s)	iterat.	t (s)
A	1	4	3	0.34	1003	2.35
	2	5	4	0.64	1503	4.21
	3	6	5	1.00	2003	6.79
	4	7	6	1.48	2503	10.23
B	1	17	4	0.91	1502	7.52
	2	71	8	5.12	3502	49.14
	3	278	16	25.72	7502	283.70
	4	1255	35	142.20	-	-

Table 1: $x_0 \simeq_{f_i} x_i$ via $f_i(x) = x/i + 1$. Timeout 1200 sec.[1]

the interleaving semantics. Hence the number of SMT calls grows exponentially. Yet in addition of being faster, the abstraction is more space efficient than SpaceEx. Note that the use of SMT calls amounts to 80% of the computation time.

5. TRANSFORMATION

In this section we present our transformation for networks of hybrid automata which reduces a given set of quasi-dependent variables and reflects reachability of forbidden configurations. For simplicity, we impose a set of syntactical criteria called well-formedness rules on networks.

Firstly, we introduce some notions which we need to state our rules. Let $W \subseteq V$ be a set of valuations. We use $\mathit{vars}(W) \subseteq \mathit{Var}$ to denote the set of variables that are constrained by W, i.e. $\mathit{vars}(W) = \{x \in \mathit{Var} \mid \{\nu(x) \mid \nu \in W\} \neq \mathbb{R}\}$. Analogously, for update $\mu \subseteq V \times V$ and action f, we use $\mathit{vars}(\mu)$ and $\mathit{vars}(f)$, to denote the set of variables that are constrained by μ and f, respectively. We use $\mathcal{V}(\mathcal{H})$ to denote the set of variables that are constrained by some initial condition, location invariant, update, or action in $\mathcal{H}$.

We use $\mathcal{SE}_Y(\mathcal{H})$ to denote the set of *simple resetting edges* of hybrid automaton $\mathcal{H}$ where only quasi-dependent variables from $Y \in \mathcal{QD}_\mathcal{N}$ are constrained by the conditional update and which have action τ, i.e., $\mathcal{SE}_Y(\mathcal{H}) = \{(\ell, \tau, \mu, \ell') \in \mathit{Edge}(\mathcal{H}) \mid \mathit{vars}(\mu) \subseteq Y\}$. We use $\mathcal{CE}_Y(\mathcal{H})$ to denote the set of *complex resetting edges* of $\mathcal{H}$ where quasi-dependent and non-quasi-dependent variables occur in the conditional update, or which have a label different from τ, i.e., $\mathcal{CE}_Y(\mathcal{H}) = \{(\ell, a, \mu, \ell') \in \mathit{Edge}(\mathcal{H}) \mid \mathit{vars}(\mu) \cap Y \neq \emptyset \wedge (\mathit{vars}(\mu) \setminus Y \neq \emptyset \vee a \neq \tau)\}$. We use $\mathcal{E}_Y(\mathcal{H}) = \mathcal{SE}_Y(\mathcal{H}) \cup \mathcal{CE}_Y(\mathcal{H})$ to denote the set of resetting edges of $\mathcal{H}$ wrt. Y, and $\mathcal{RES}_Y(\mathcal{N})$ to denote the set of automata in $\mathcal{N}$ which have a Y-resetting edge, i.e., $\mathcal{RES}_Y(\mathcal{N}) = \{\mathcal{H} \in \mathcal{N} \mid \mathcal{E}_Y(\mathcal{H}) \neq \emptyset\}$.

We use $\mathcal{SL}_Y(\mathcal{H})$ and $\mathcal{CL}_Y(\mathcal{H})$ to respectively denote the set of source and destination locations of simple and complex resetting edges of $\mathcal{H}$. A location ℓ (ℓ') is called *reset (successor) location* wrt. Y in $\mathcal{N}$ if and only if there is a resetting edge in $\mathcal{E}_Y(\mathcal{H})$ from (to) ℓ (ℓ'). We use $\mathcal{RL}_Y^-$ ($\mathcal{RL}_Y^+$) to denote the set of reset (successor) locations wrt. Y in $\mathcal{N}$. We define $\mathcal{RL}_\mathcal{N}^- := \bigcup_{Y \in \mathcal{QD}_\mathcal{N}} \mathcal{RL}_Y^-$ and similarly $\mathcal{RL}_\mathcal{N}^+$.

Definition 9. (Well-formed Network) A network $\mathcal{N}$ is called *well-formed* if and only if it satisfies the following restrictions for each set of quasi-dependent variables $Y \in \mathcal{QD}_\mathcal{N}$:

(R1) All resetting edges update at most one variable $x \in Y$, and, given $x \simeq_f y$, the guard of the edge and the invariant of its source location and the assignment, resp., determine unique values $C_Y, O_Y \in \mathbb{R}$, i.e.

$$\begin{aligned} &\exists\, C_Y, O_Y \in \mathbb{R}\ \forall\, (\ell, a, \mu, \ell') \in \mathcal{E}_Y(\mathcal{N})\ \exists\, x \in Y \bullet \\ &\quad \forall\, (\nu, \nu') \in \mu \bullet \nu|_{\mathcal{V}\setminus\{x\}} = \nu'|_{\mathcal{V}\setminus\{x\}} \\ &\quad \wedge \forall\, \nu \in (\mu \downarrow_1) \cap \mathit{Inv}(\ell) \bullet \nu(x) = C_Y \\ &\quad \wedge \forall\, \nu \in (\mu \downarrow_2) \bullet \nu(x) = O_Y, \end{aligned}$$

where $\mu \downarrow_i$ denotes projection onto the i-th component.

(R2) There are no two resetting edges from one location, i.e.

$$\begin{aligned} &\forall\, e_1 = (\ell_1, a_1, \mu_1, \ell_1'), e_2 = (\ell_2, a_2, \mu_2, \ell_2') \in \mathcal{E}_Y(\mathcal{N}) \bullet \\ &\quad e_1 \neq e_2 \implies \ell_1 \neq \ell_2. \end{aligned}$$

(R3) All edges that synchronise on some label either all reset a clock from Y or none does, i.e.,

$$\begin{aligned} &\forall\, a \in \mathit{Lab} \bullet (\exists\, e = (\ell, a, \mu, \ell') \in \mathit{Edge}(\mathcal{N}) \bullet e \in \mathcal{E}_Y(\mathcal{N})) \\ &\quad \implies (\forall\, e = (\ell, a, \mu, \ell') \in \mathit{Edge}(\mathcal{N}) \bullet e \in \mathcal{E}_Y(\mathcal{N})). \end{aligned}$$

(R4) No guard relates two or more variables from Y, i.e.

$$\begin{aligned} &\forall\, e = (\ell, a, \mu, \ell') \in \mathit{Edge}(\mathcal{N})\ \forall\, x \in Y \bullet (\mu \downarrow_1)|_{\{x\}} \subsetneq \mathbb{R} \\ &\quad \implies \forall\, y \in Y \setminus \{x\} \bullet (\mu \downarrow_1)|_{\{y\}} = \mathbb{R}. \end{aligned}$$

Note that network $\mathcal{N}$ from Figure 1 is well-formed.

In the following we describe our transformation procedure $\mathcal{K}$. It works with two given inputs, a well-formed network $\mathcal{N} = \{\mathcal{H}_1, \dots, \mathcal{H}_n\}$ and a set of equivalence classes $\mathcal{QD}_\mathcal{N}$ of quasi-dependent variables. The output is the transformed network $\mathcal{N}' = \{\mathcal{H}_1', \dots, \mathcal{H}_n'\} \cup \{\mathcal{R}_Y \mid Y \in \mathcal{QD}_\mathcal{N}\}$ which consists of hybrid automata $\mathcal{H}_i' = (\mathit{Loc}'(\mathcal{H}_i), \mathit{Var}', \mathit{Lab}'(\mathcal{H}_i), \mathit{Edge}'(\mathcal{H}_i), \mathit{Act}'(\mathcal{H}_i), \mathit{Inv}'(\mathcal{H}_i), \mathit{Init}'(\mathcal{H}_i))$ which are obtained as modifications of the $\mathcal{H}_i$, and one *resetter* automaton for each equivalence class $Y \in \mathcal{QD}_\mathcal{N}$.

The common set of variables Var' consists of the non-quasi-dependent variables in $\mathcal{N}$, one representative $\mathit{rep}(Y)$ for each equivalence class, and bookkeeping variables $\mathit{rst}_Y^{I_\mathcal{H}}$ and $\mathit{rst}_Y^{O_\mathcal{H}}$, $\mathcal{H} \in \mathcal{N}$. For each equivalence class $Y \in \mathcal{QD}_\mathcal{N}$, the set of labels of $\mathcal{H}$ is extended by the fresh label reset_Y if $\mathcal{H} \in \mathcal{RES}_Y(\mathcal{N})$.

For each automaton $\mathcal{H} \in \mathcal{N}$, and for each complex edge $e \in \mathcal{CE}_Y(\mathcal{H})$, the set of locations is extended by one fresh location $\ell_{\xi_{Y,e}}$ and the set of edges is extended by one edge with label reset_Y and no guard and no assignment from the source of e to $\ell_{\xi_{Y,e}}$. The source location of e is changed to be $\ell_{\xi_{Y,e}}$, any guard on and any assignment of quasi-dependent variables is removed from its μ. For each simple edge, its label τ is replaced by reset_Y and any guard on and assignment of quasi-dependent variables is removed from its μ.

In order to keep track of whether a hybrid automaton is ready to reset a quasi-dependent variable or has just reset its variable, the assignments of all edges with a reset location as destination are changed such that $\mathit{rst}_Y^{I_\mathcal{H}}$ is assigned 1, the assignments of all non-resetting edges with a reset location as source are changed such that $\mathit{rst}_Y^{I_\mathcal{H}}$ is assigned 0, and the assignments of all resetting edges (including those redirected to have source $\ell_{\xi_{Y,e}}$) is changed such that $\mathit{rst}_Y^{O_\mathcal{H}}$ is assigned 0. Note that $\mathit{rst}_Y^{I_\mathcal{H}}$ may also be assigned 1 on resetting edges in case of loops.

For each location, any action $f \in \mathit{Act}(\mathcal{H})$ is restricted to Var' and modified such that $\mathit{rst}_Y^{I_\mathcal{H}}$ and $\mathit{rst}_Y^{O_\mathcal{H}}$ keep their value over time, and such that the evolution of the representative variables is unconstrained. Their evolution is

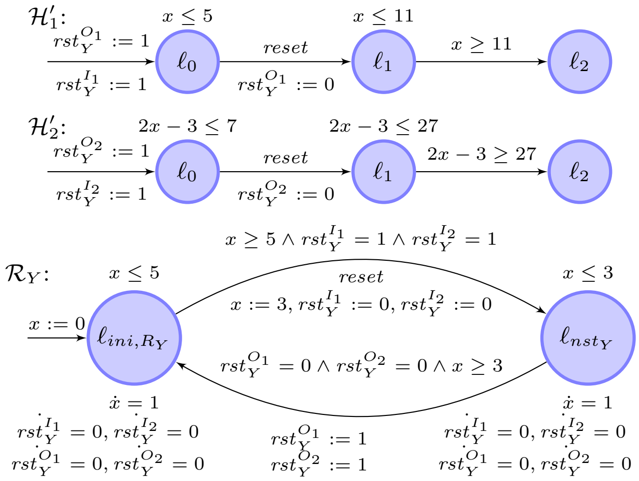

Figure 5: Network $\mathcal{N}'$ after applying the transformation.

determined by the resetter automata. The actions of the locations $\ell_{\xi_{Y,e}}$ keep all values constant. For each location, the valuations in the invariant and the initial valuations set are modified such that they express the original guard in terms of the representative variable and the dependency function.

For example, applying $\mathcal{K}$ to network $\mathcal{N}$ from Figure 1 with the set of quasi-dependent variables $Y = \{x, y\}$ yields the automata $\mathcal{H}_1'$ and $\mathcal{H}_2'$ as shown in Figure 5. Note that only the representative variable of Y remains, namely variable x.

The *resetter* for equivalence class Y is the hybrid automaton $\mathcal{R}_Y$ with two locations $\ell_{ini,\mathcal{R}_Y}$ and $\ell_{nst,Y}$ and two edges $(\ell_{ini,\mathcal{R}_Y}, reset_Y, \mu_1, \ell_{nst,Y})$ and $(\ell_{nst,Y}, \tau, \mu_2, \ell_{ini,\mathcal{R}_Y})$. The first edge obtains a guard μ_1 which checks whether the representative variable of equivalence class Y equals C_Y and whether the $rst_Y^{I_\mathcal{H}}$ evaluate to 1. μ_1 assigns all $rst_Y^{I_\mathcal{H}}$ to 0 and the representative variable to O_Y. The second edge obtains a guard μ_2 which checks whether $rst_Y^{O_\mathcal{H}}$ evaluates to 0, and assigns 1 to $rst_Y^{O_\mathcal{H}}$.

Both locations of $\mathcal{R}_Y$ inherit the actions for the representative variable x from the automaton constraining x. Additionally, from the same automaton, location $\ell_{ini,\mathcal{R}_Y}$ inherits the invariant of the reset location with an outgoing edge updating x. Location $\ell_{ini,\mathcal{R}_Y}$ is set as the initial location and in $\ell_{nst,Y}$ we require that the value of the representative is O_Y. The variables $rst_Y^{I_\mathcal{H}}$ and $rst_Y^{O_\mathcal{H}}$ keep their value over time. $rst_Y^{I_\mathcal{H}}$ is initialised to 1 on initial locations which are reset locations and to 0 otherwise; $rst_Y^{O_\mathcal{H}}$ is initialised to 1.

See Figure 5 for an example resetter. Note that the guard and the update operation are delegated to the resetter. Further note that well-formedness together with the counter variables $rst_Y^{I_\mathcal{H}}$ enforces blocking multicast synchronisation, i.e., always all automata from $\mathcal{RES}_Y(\mathcal{N})$ participate in the reset.

To support all possible queries, we use the location $\ell_{nst,Y}$ to summarise information from configurations that are induced by interleaving updates of quasi-dependent variables. The variables $rst_Y^{O_\mathcal{H}}$ are introduced to indicate how many automata still need to take their reset edges. The following function Ω (cf. Table 2) syntactically transforms properties over a well-formed network $\mathcal{N}$ into equivalent properties over $\mathcal{N}'$ (cf. Theorem 5.2).

Function Ω treats queries for source or destination locations of resetting edges special, by referring these queries to the summary location $\ell_{nst,Y}$. For instance, for a simple resetting edge $e \in \mathcal{SE}_Y(\mathcal{H})$ of some $\mathcal{H} \in \mathcal{N}$, the source location ℓ of e can be assumed in $\mathcal{N}$ in different configurations: either the reset time is not yet reached, or the reset time is reached but $\mathcal{H}$ did not reset yet, while other automata in $\mathcal{RES}_Y(\mathcal{N})$ may have reset their quasi-dependent variables already. In $\mathcal{N}'$, all edges resulting from simple edges fire at once on the synchronisation on $reset_Y$, so all source locations are left together. With this synchronisation, the resetter moves to $\ell_{nst,Y}$, which represents all configurations of $\mathcal{N}$ where all simple edges are in their source or destination location and the variables updated on these edges have value C_Y or O_Y. Thus the location ℓ is reachable in $\mathcal{N}$ if and only if, for $\mathcal{N}'$, (i) $\ell_{nst,Y}$ is reachable, or (ii) ℓ is reached while being stable, i.e., not being in $\ell_{nst,Y}$.

Note that $\Omega(CF)$ consists of two parts: a transformation of CF by Ω_0 which uses fresh existentially quantified variables and a feasibility condition for these fresh variables. Thereby we represent that, although involving two choices of location and variable value each, there are actually only two cases (not four) summarised by $\ell_{nst,Y}$. Namely, assuming the source location of a simple resetting edge implies that the representative has value C_Y, and assuming the destination location implies the value O_Y. Furthermore, we can only assume either the source or the destination location of the simple resetting edge. By *R1*, we only need to consider, e.g., O_Y and C_Y as values of $\tilde{x}$, thus the existential quantification can be rewritten into a big disjunction, and hence is a proper configuration formula.

In the following, we observe that our transformation yields a network whose stable configurations (see below) directly relate to the stable configurations of $\mathcal{N}$ (one-to-one). For unstable configurations from both networks the relation between them is more involved (one-to-many), i.e., this relation depends on information from simple and complex resetting edges, as well as on values from *rst*-variables.

Definition 10. (Stable and Unstable Configurations) Let $\mathcal{N}$ be a network and let $Y \in \mathcal{QD}_\mathcal{N}$ be a set of quasi-dependent variables. A configuration $s \in \mathit{Conf}(\mathcal{N})$ is called *stable wrt.* $Y \in \mathcal{QD}_\mathcal{N}$ if and only if $\forall x \in Y, x \simeq_f rep(x) \bullet \nu_s(x) = f(\nu_s(rep(x)))$. We use $\mathcal{SC}_\mathcal{N}^Y$ to denote the set of all configurations that are stable wrt. Y and $\mathcal{SC}_\mathcal{N}$ to denote the set $\bigcap_{Y \in \mathcal{QD}_\mathcal{N}} \mathcal{SC}_\mathcal{N}^Y$ of *globally stable* configurations of $\mathcal{N}$. Configurations not in $\mathcal{SC}_\mathcal{N}$ are called *unstable*.

A configuration $r \in \mathit{Conf}(\mathcal{N}')$ of $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{QD}_\mathcal{N})$ is called *stable wrt.* Y if and only if the initial location $\ell_{ini,\mathcal{R}_Y}$ of resetter $\mathcal{R}_Y \in \mathcal{N}'$ occurs in r, i.e., if $\ell_r = \ell_{ini,\mathcal{R}_Y}$. ♢

The following definition allows us to define a weak bisimulation relation between $\mathcal{N}$ and $\mathcal{N}'$.

Definition 11. (Delayed Edge) An edge e of a hybrid automaton $\mathcal{H}$ in network $\mathcal{N}$ is called *delayed* if and only if time must pass before e can be taken, i.e., if
$\forall s_0 \xrightarrow{\lambda_1}_{E_1} \dots s_{n-1} \xrightarrow{\lambda_n}_{E_n} s_n \in \Pi(\mathcal{H}) \bullet e \in E_n \implies$ $\exists 0 \leq j < n \bullet \lambda_j \in \mathbb{R}_0^+ \setminus \{0\} \wedge \forall j \leq i < n \bullet \mathit{Edge}(\mathcal{H}) \cap E_i = \emptyset$.
Here, we write $s_i \xrightarrow{\lambda_i}_{E_i} s_{i+1}$, $i \in \mathbb{N}^+$, to denote that the transition $s_i \xrightarrow{\lambda_i} s_{i+1}$ is justified by the set of edges E_i; E_i is empty for time transitions, i.e., if $\lambda_i \in \mathbb{R}_0^+ \times \mathit{Lab}$.

We say *$\mathcal{QD}_\mathcal{N}$-reset edges are pre/post delayed in well-formed network $\mathcal{N}$* if and only if all edges originating in

$$\Omega_0(\beta) = \begin{cases} (\ell \wedge \neg\ell_{nst,Y}) \vee (\ell_{nst,Y} \wedge \tilde{\ell}) & \text{, if } \beta = \ell, \ell \in \mathcal{SL}_Y(\mathcal{N}) \cup (\mathcal{CL}_Y(\mathcal{N}) \cap \mathcal{RL}_Y^-), Y \in \mathcal{QD}_\mathcal{N}. \\ (\neg\ell \wedge \neg\ell_{nst,Y}) \vee (\ell_{nst,Y} \wedge \neg\tilde{\ell}) & \text{, if } \beta = \neg\ell, \ell \in \mathcal{SL}_Y(\mathcal{N}) \cup (\mathcal{CL}_Y(\mathcal{N}) \cap \mathcal{RL}_Y^-), Y \in \mathcal{QD}_\mathcal{N}. \\ \big(\varphi[x/rep(x) \mid x \in \mathit{Var}] \wedge \neg\ell_{nst,Y}\big) \vee \big(\ell_{nst,Y} \wedge \tilde{\varphi}\big) & \text{, if } \beta = \varphi, \tilde{\varphi} = \varphi[x/\tilde{x} \mid x \in \mathit{Var}], Y \in \mathcal{QD}_\mathcal{N}. \\ \beta & \text{, otherwise} \end{cases}$$

$$\Omega(CF) = \exists\, \tilde{x}_1, \dots, \tilde{x}_k\ \exists\, \tilde{\ell}_1, \dots, \tilde{\ell}_m \bullet$$

$$\Omega_0(CF) \wedge \bigwedge_{\substack{1 \le i \le k, 1 \le j \le m, \\ x_j \in \mathcal{X}_p \cap Y, 1 \le p \le n, \\ \ell_i \in L_p \cap (\mathcal{RL}_Y^- \setminus \mathcal{RL}_Y^+)}} (\tilde{\ell}_i \implies \tilde{x}_j = C_Y) \wedge \bigwedge_{\substack{1 \le i \le k, 1 \le j \le m, \\ x_j \in \mathcal{X}_p \cap Y, 1 \le p \le n, \\ \ell_i \in L_p \cap (\mathcal{RL}_Y^+ \setminus \mathcal{RL}_Y^-)}} (\tilde{\ell}_i \implies \tilde{x}_j = O_Y) \wedge \bigwedge_{\substack{(\ell, a, \mu, \ell') \\ \in \mathcal{SE}_Y(\mathcal{H}), \\ \ell_i \in \{\ell, \ell'\}}} (\tilde{\ell}_i \implies \ell') \wedge \bigwedge_{\substack{(\ell, a, \mu, \ell') \\ \in \mathcal{CE}_Y(\mathcal{H}), \\ \ell_i = \ell}} (\tilde{\ell}_i \implies \ell_{\xi_{Y,e}}) \wedge \bigwedge_{\substack{1 \le i \ne j \le m, \\ 1 \le p \le n \\ \ell_i, \ell_j \in L_p,}} \neg(\tilde{\ell}_i \wedge \tilde{\ell}_j)$$

Table 2: Formula transformation function Ω. $\Omega_0(CF)$ denotes applying Ω_0 to each basic formula in CF.

reset (successor) locations are delayed, i.e., if for all $e = (\ell, a, \mu, \ell') \in \mathit{Edge}(\mathcal{N})$, $\ell \in \mathcal{RL}_\mathcal{N}^- \cup \mathcal{RL}_\mathcal{N}^+$ implies that e is delayed. $\diamondsuit$

There are *sufficient* syntactic criteria for an edge $e = (\ell_1, a_1, \mu_1, \ell_2)$ being delayed. For instance, if $(\ell_0, a_0, \mu_0, \ell_1)$ is the only incoming edge to ℓ_1 and if the condition of μ_0 is $(x \ge C \wedge x \le C)$ and the condition of μ_1 is $(x \ge D \wedge x \le D)$ and $C < D$, then e is delayed. It is also delayed if $(\ell_0, a_0, \mu_0, \ell_1)$ is the only incoming edge to ℓ_1, μ_0 updates x, and the condition of μ_1 is $(x > 0)$.

Both patterns occur, e.g., in the FSN case-study (cf. Section 5.1). There, the reset location is entered via an edge following the former pattern, and the edges originating at the reset successor location follow the latter pattern. Thus $\mathcal{QD}_\mathcal{N}$-reset edges are pre/post delayed in FSN.

LEMMA 5.1. *(Weak Bisimulation) Any well-formed network $\mathcal{N}$ where $\mathcal{QD}_\mathcal{N}$-reset edges are pre/post delayed, is* weakly bisimilar *to $\mathcal{N}' = \mathcal{K}(\mathcal{N}, \mathcal{QD}_\mathcal{N})$, i.e., there is a* weak bisimulation relation *$\mathcal{S} \subseteq \mathit{Conf}(\mathcal{N}) \times \mathit{Conf}(\mathcal{N}')$ such that*

1. *$\forall\, s \in \mathit{Conf}(\mathcal{N})\ \exists\, r \bullet (s, r) \in \mathcal{S}$ and $\forall\, r \in \mathit{Conf}(\mathcal{N})\ \exists\, s \bullet (s, r) \in \mathcal{S}$.*

2. *For all configuration formulae CF over $\mathcal{N}$, $\forall\, (s, r) \in \mathcal{S} \bullet s \models CF \implies r \models \Omega(CF)$ and $\forall\, r \in \mathit{CONS}_{\mathcal{QD}_\mathcal{N}} \bullet r \models \Omega(CF) \implies \exists\, s \in \mathit{Conf}(\mathcal{N}) \bullet (s, r) \in \mathcal{S} \wedge s \models CF$. Where $r \in \mathit{CONS}_{\mathcal{QD}_\mathcal{N}}$ iff*

   $$\forall\, Y \in \mathcal{QD}_\mathcal{N} \bullet \nu_r(rst_Y^{I_\mathcal{H}}) = |\{\ell_{r,1}, \dots, \ell_{r,n}\} \cap \mathcal{RL}_Y^-| \wedge \ell_{r,\mathcal{R}_Y} = \ell_{ini,\mathcal{R}_Y} \implies \nu_r(rst_Y^{O_\mathcal{H}}) = |Y| \wedge \ell_{r,\mathcal{R}_Y} = \ell_{nst,\mathcal{R}_Y} \implies \nu_r(rst_Y^{O_\mathcal{H}}) = |\{\ell_{r,1}, \dots, \ell_{r,n}\} \cap \Xi_Y|.$$

3. *For all $(s, r) \in \mathcal{S}$, if $s \xrightarrow{\lambda} s'$ with*

   (a) *$s, s' \notin \mathcal{SC}_\mathcal{N}^Y$, where $Y \in \mathcal{QD}_\mathcal{N}$, and justified by a simple resetting edge, or $s \notin \mathcal{SC}_\mathcal{N}^Y$, $s' \in \mathcal{SC}_\mathcal{N}^Y$, where $Y \in \mathcal{QD}_\mathcal{N}$, and justified by a simple resetting edge, then $r \xrightarrow{0} r$ and $(s', r) \in \mathcal{S}$.*

   (b) *$s \in \mathcal{SC}_\mathcal{N}^Y$, $s' \notin \mathcal{SC}_\mathcal{N}^Y$, where $Y \in \mathcal{QD}_\mathcal{N}$, and justified by the set $CE_Y^1 \subseteq \mathcal{CE}_Y(\mathcal{N})$ of complex resetting edges wrt. Y, or $s, s' \in \mathcal{SC}_\mathcal{N}^Y$, where $Y \in \mathcal{QD}_\mathcal{N}$, and justified by $CE_Y \subseteq \mathcal{CE}_Y(\mathcal{N})$, or $s, s' \in \mathcal{SC}_\mathcal{N}^Y$, $\ell_r = \ell_{nst,\mathcal{R}_Y}$ for some $Y \in \mathcal{QD}_\mathcal{N}$, and $\lambda = d > 0$, then there exist r', r'' such that $r \xrightarrow{\tau} r' \xrightarrow{\lambda} r''$ and (s, r'), $(s', r'') \in \mathcal{S}$.*

   (c) *Otherwise there is r' s.t. $r \xrightarrow{\lambda} r'$ and $(s', r') \in \mathcal{S}$.*

   and if $r \xrightarrow{\lambda} r'$ with

   (a) *$r \in \mathcal{SC}_{\mathcal{N}'}^Y, r' \notin \mathcal{SC}_{\mathcal{N}'}^Y$, where $Y \in \mathcal{QD}_\mathcal{N}$, $\nu_{r'}$ $(rst_Y^{O_\mathcal{H}}) < N$, where $N = \nu_r(rst_Y^{O_\mathcal{H}})$, there exist $s_1, \dots, s_n$ where $n = N - \nu_{r'}(rst_Y^{O_\mathcal{H}})$, such that $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_n$ and $(s_i, r') \in \mathcal{S}$, $1 \le i \le n$.*

   (b) *$r \in \mathcal{SC}_{\mathcal{N}'}^Y, r' \notin \mathcal{SC}_{\mathcal{N}'}^Y$, $\nu_{r'}(rst_Y^{O_\mathcal{H}}) = \nu_r(rst_Y^{O_\mathcal{H}})$, where $Y \in \mathcal{QD}_\mathcal{N}$, or $\ell_r = \ell_{nst,\mathcal{R}_Y}$, $\ell_{r'} \ne \ell_{nst,\mathcal{R}_Y}$, $Y \in \mathcal{QD}_\mathcal{N}$, then $s \xrightarrow{0} s$ and $(s, r') \in \mathcal{S}$.*

   (c) *Otherwise there is s' s.t. $s \xrightarrow{\lambda} s'$ and $(s', r') \in \mathcal{S}$.*

THEOREM 5.2. *Let CF be a configuration formula over well-formed network $\mathcal{N}$ with pre/post delayed $\mathcal{QD}_\mathcal{N}$-resets. CF is reachable in $\mathcal{N}$ iff $\Omega(CF)$ is reachable in $\mathcal{K}(\mathcal{N}, \mathcal{QD}_\mathcal{N})$.*

5.1 Experimental Results

For our experimental evaluation[1] we have chosen two industrial time-based protocols for wireless sensor networks where the timers of the individual sensors advance at different rates (each rate being constant). In our case studies we have verified queries which were proposed by the respective authors. Our first benchmark, the FSN fire alarm protocol from [21] belongs to the class of TDMA protocols. It consists of one controller and, in our setting, from 10 to 130 sensors. Each sensor has only one (simple) resetting edge. We verified that the communication between controller and sensors is successful, i.e., each message received by the controller is replied to and reaches the transmitting sensor. The results are reported in Table 3; we have used the same machine as in Section 4.1. We compare the number of iterations and analysis runtime needed by SpaceEx for the original and the transformed systems (the latter denoted in the table by the suffix K). Furthermore, we report time spent in the detection phase. It is important to note that the detection can be done in a compositional manner by considering only *pairs* of sensors and their variables. These detections can run in parallel; the final set of quasi-dependent variables can be derived using transitivity. Therefore, in column "Detect" of Table 3, we show the results of pair-wise detection. Overall, our approach shows better performance starting from the smallest instances. Furthermore, our combined approach

[1] Tools and benchmarks are available for download [1].

	Detect		Check		
	pairs	Σ t(s)	iter.	t(s)	Σ
10			2086	47.18	47.18
10_K	9	27.68	41	0.14	27.82
11			4138	134.53	134.53
11_K	10	30.74	45	0.17	30.91
12			-	-	-
12_K	11	33.78	49	0.19	33.97
110_K	109	334.28	441	46.35	380.63
120_K	119	364.93	481	59.72	424.65
130_K	129	395.48	521	75.55	471.03

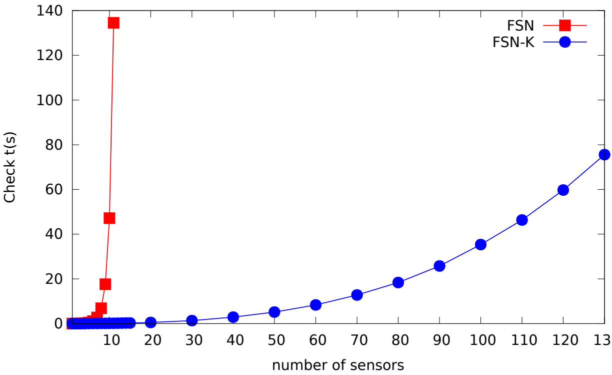

Table 3: Benchmark FSN: For each pair of quasi-dependent variables, detect needs 41 SMT calls, 14 states.[1]

uses much less memory thus we can treat larger system instances. Our approach scales up to 130 sensors whereas the analysis of the original system runs out of memory already for the instance with 12 sensors.

Our second case study [3] belongs to the class of EPL protocols. Once again there is one controller with multiple sensors, each sensor has a single (complex) resetting edge where quasi-dependent and non-quasi-dependent variables are updated. We verified that all sensors receive and acknowledge each message sent by the controller. The results for our settings range from 7 to 11 sensors and are presented in Table 4. Although it always takes SpaceEx less time to analyse the transformed system than to analyse the original system, we observe that the speed-up is smaller than for the FSN benchmark. This can be explained by the fact that we are able to completely remove the interleaving "diamond" in case of FSN because of only simple resetting edges, whereas for EPL we only reduce the number of variables while still having to analyze the interleavings induced by the update of non-quasi-dependent variables in complex resetting edges. Those interleavings also have an impact on the detection phase performance as we need to consider more paths in the abstract region graph. Still, we can observe that the analysis of the original system runs out of memory for 10 sensors, our approach can analyze larger instances.

As SpaceEx supports only continuous variables, we have, for the sake of efficiency, represented values of the discrete counter variables $rst_Y^{I_\mathcal{H}}$ and $rst_Y^{O_\mathcal{H}}$ by corresponding locations in the resetter automata.

	Detect		Check		
	pairs	Σ t(s)	iter.	t(s)	Σ
7			396	5.45	5.45
7_K	5	48.53	408	2.05	50.58
8			781	15.51	15.51
8_K	6	58.21	794	5.28	63.49
9			1550	43.75	43.75
9_K	7	68.25	1564	14.27	82.52
10			-	-	-
10_K	8	77.85	3102	41.89	119.74
11_K	9	87.49	6176	139.91	227.40

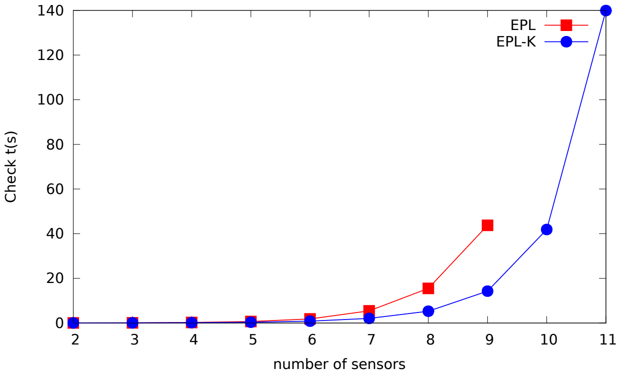

Table 4: Benchmark EPL: For each pair of quasi-dependent variables, detect needs 120 SMT calls, 34 states.[1]

6. CONCLUSION

In this paper, we have presented techniques to efficiently exploit the dependencies between continuous variables in hybrid automata.

The technical contribution comprises two methods: an automatic detection of quasi-dependent variables and a method to automatically transform and reduce the original system. The current prototypical implementation of our detection algorithm is backed by a linear SMT solver. Although using a rather straight-forward, constraint-based representation of regions in the abstract region graph, we have been able to achieve a significant performance speed-up. Thus, we can expect to achieve even better performance when using elaborated data structures to represent regions. Our transformation technique shrinks the set of states emerging during discrete updates of quasi-dependent variables due to the interleaving semantics. This leads to an abstracted hybrid automaton with a much smaller state space, which, however, reflects all properties of the original system.

Overall, a combination of our methods, i.e. the detection part and consequent transformation, shows very promising results compared to the application of the hybrid model checker SpaceEx to original models. Although we have only evaluated our approach on linear hybrid automata motivated by industrial, time-based protocols where the timers of the individual sensors advance at measurably different rates, our theory is generic as it uses the SMT solver (for the detecting part) and SpaceEx (for the analysis of a reduced hybrid automaton) as black-boxes. In particular, our detection algorithm leverages the power of the underlying SMT-solver, thus using dReal [22] or HySAT [23], which can handle non-linear constraints, would immediately support detection for systems with non-linear behaviour. The transformation does not depend on the class of continuous dynamics at all.

Acknowledgments.

We thank Goran Frehse for his assistance with SpaceEx during the preparation of our benchmark suite.

7. REFERENCES

[1] `http://swt.informatik.uni-freiburg.de/projects/CaseStudyRepository/hybridqdv`.

[2] T.S. Rappaport. *Wireless communications*, volume 2. Prentice Hall, 2002.

[3] G. Cena et al. Performance analysis of ethernet powerlink networks for distributed control and automation systems. *CSI*, 31(3):566–572, 2009.

[4] C. Herrera, B. Westphal, S. Feo-Arenis, M. Muñiz, and A. Podelski. Reducing quasi-equal clocks in networks of timed automata. In *FORMATS*, pages 155–170. Springer, 2012.

[5] C. Herrera, Westphal, and A. Podelski. Quasi-equal clock reduction: More networks, more queries. In *TACAS*. Springer, 2014.

[6] M. Muñiz, B. Westphal, and A. Podelski. Detecting quasi-equal clocks in timed automata. In *FORMATS*, pages 198–212. Springer, 2013.

[7] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6(2):133–151, 1976.

[8] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM, 1977.

[9] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *RTSS*, pages 73–81. IEEE, 1996.

[10] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, pages 313–329. Springer, 1998.

[11] A. Étienne. Dynamic clock elimination in parametric timed automata. In *FSFMA*, volume 31, pages 18–31. Schloss Dagstuhl, 2013.

[12] R. Ben Salah et al. Compositional timing analysis. In *EMSOFT*, pages 39–48. ACM, 2009.

[13] G. J. Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.

[14] A. J. Van der Schaft. Equivalence of dynamical systems by bisimulation. *IEEE Trans. Automatic Control*, 49(12):2160–2172, 2004.

[15] A. Girarda and G. J. Pappas. Approximate bisimulation relations for constrained linear systems. *Automatica*, 43:1307–1317, 2007.

[16] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008.

[17] R. Alur, C. Courcoubetis, et al. The algorithmic analysis of hybrid systems. *TCS*, 138(3):34, 1995.

[18] L. De Moura and N. Bjørner. Z3: An efficient smt solver. *TACAS*, pages 337–340, 2008.

[19] K. Zee, V. Kuncak, and M. Rinard. Full functional verification of linked data structures. In *PLDI*, pages 349–361. ACM, 2008.

[20] G. Frehse et al. SpaceEx: Scalable verification of hybrid systems. In *CAV*, LNCS. Springer, 2011.

[21] D. Dietsch, S. Feo Arenis, B. Westphal, and A. Podelski. Disambiguation of industrial standards through formalization and graphical languages. In *RE*, pages 265–270. IEEE, 2011.

[22] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE*, pages 208–214. Springer, 2013.

[23] M. Fränzle, C. Herde, et al. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.