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 generalization 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 as-systems. In such settings, the information exchange among protocols are prominent examples of this particular class of final purposes. The classes of TDMA- [2] and EPL-based [3] where every component uses an independent clock for inter-

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 simulation relations. In particular, Pappas [13] and van der Schaft [14] presented a theoretical framework of bisimulation between variables of a program is discussed and evaluated in Section 4, the transformation for linear systems. The idea of simulation was extended to hybrid automata each with variables \( \mathcal{H} = (\text{Loc}, \text{Var}, \text{Label}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init}) \) where \( \text{Loc} \) is a finite set of locations, \( \text{Var} \) is a finite set of real-valued variables, \( \text{Lab} \) is a set of synchronisation labels including the stutter label \( \tau \in \text{Lab} \), \( \text{Edge} \subseteq \text{Loc} \times \text{Label} \times 2^{\mathbb{R}^2} \times \text{Loc} \) is a finite set of directed edges including a stutter edge \( (\ell, \tau, id, \ell) \) for each location \( \ell \in \text{Loc} \). An edge \((\ell, a, \mu, \ell')\) from location \( \ell \) to \( \ell' \) is labelled with a label \( a \in \text{Lab} \) and a conditional update \( \mu \). \( \text{Act} : \text{Loc} \rightarrow 2^{\mathbb{R}^2} \) is a function which assigns a set of activities \( f : \mathbb{R}^2 \rightarrow V \) to each location. The activity sets are time-invariant, i.e., for each \( \ell' \in \text{Loc}_t^\mathbb{R}^2, f(\ell') \) implies \( f(\ell+\tau) = f(\ell+\tau) \). \( \text{Inv} : \text{Loc} \rightarrow 2^{\mathbb{R}^2} \) is a function which assigns an invariant \( \text{Inv}(\ell) \subseteq V \) to each location \( \ell \). Locations \( \ell \) with \( \text{Inv}(\ell) \neq \emptyset \) are called initial locations.

A network \( N = \{H_1, \ldots, H_n\} \) is a finite set of hybrid automata each with variables \( \text{Var} \) and with pairwise disjoint sets of locations. We write \( \mathcal{H} \in N \) if and only if \( \mathcal{H} \in \{H_1, \ldots, H_n\} \). We use \( \text{Lab}(H), \text{Edge}(H) \), etc. to denote the set of labels, edges, etc. of automaton \( \mathcal{H} \).

The operational semantics of a network \( N \) is defined by the (labelled) transition system

\[
T(N) = (\text{Conf}(N), \Lambda, \xi, \xi_0 | \lambda \in \Lambda, \xi_{\text{init}})
\]

where \( \text{Conf}(N) = \text{Loc} \times V \) and \( \Lambda := (\mathbb{R}_0^+ \times \text{Act}(N)) \cup \bigcup_{1 \leq i < n} \text{Lab}_i \) is the set of transition labels. The set of configurations \( \text{Conf}(N) \) consists of pairs of location vectors \( \xi = (\xi_0, \xi_1, \ldots, \xi_n) \in \mathcal{N}_{\text{Loc}}(\mathcal{H}_i) \) and valuations. We write \( \xi_i \), \( 1 \leq i \leq N \), to denote the location that automaton \( \mathcal{H}_i \) assumes in a configuration \( s = (\xi_i, \nu) \) and \( \nu \) to denote \( \nu_{|\text{Var}} \). The set of initial configurations is \( \xi_{\text{init}} = \{ (\xi_0, \nu) \in \text{Conf}(N) | \nu \in \bigcap_{1 \leq i < n} \text{Inv}(\xi_i) \} \). There is a discrete transition \( s \xrightarrow{a_i} s' \) from configuration \( s = ((\xi_0, \ldots, \xi_n), \nu) \) to configuration \( s' = ((\xi_1', \ldots, \xi_n'), \nu') \) if and only if there are edges \((\ell_i, a_i, \mu_i, \ell_i') \in \text{Edge}_i, 1 \leq i \leq n \), such that for each \( i \), either \( a_i = a \) or \( a_i = \tau \) and \( a \notin \text{Lab}_i \), and the edges are enabled by valuation \( \nu \) and \( \nu' \) is an effect of the updates applied to \( \nu \), i.e., \((\nu, \nu') \in \bigcap_{1 \leq i < n} \text{Inv}(\xi_i) \). There is a time transition \( s \xrightarrow{a_i, \tau} s' \) with delay \( d \in \mathbb{R}_0^+ \) from configuration \( s \) to configuration \( s' \) if and only if action \( f \) is in \( \bigcap_{1 \leq i < n} \text{Act}((\xi_i), a_i) \) and \( f(0) = \nu, f(d) = \nu', \forall 0 \leq t < d \). \( \xi(t) \in \bigcap_{1 \leq i < n} \text{Inv}(\xi_i) \).

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 \( \mathcal{H} \in N \).

Figure 1 shows an example of a network \( N \) 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
semantics of networks of hybrid automata, these updates occur one after the other.

A sequence \(\pi = (\ell_0, v_0), t_0, \lambda_1(\ell_1, v_1), t_2, \lambda_2 \cdots\) of time-stamped configurations is called run of \(\mathcal{H}\) if and only if \((\ell_0, v_0) \in C_{in}, t_0 = 0,\) and for all \(i \in \mathbb{N}_0,\) we have \(\lambda_{i+1} \in \text{Lab} \) and \(t_i = t_{i+1} \) or \(\lambda_{i+1} \in \{t_{i+1} - t_i\} \times \text{Act},\) and the transition relation \((\ell_i, v_i) \xrightarrow{\lambda_{i+1}} (\ell_{i+1}, v_{i+1}).\) We use \(P(\mathcal{H})\) 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 \(\pi = ((\ell_0, t_0), (0, -3)), 0 \xrightarrow{\ell_0} ((\ell_0, t_0), (5, 7)), 5 \xrightarrow{\ell_1} ((\ell_1, t_0), (3, 7)), 5 \xrightarrow{\ell_1} ((\ell_1, t_1), (3, 3)), 5 \xrightarrow{\ell_2} ((\ell_1, t_1), (5, 7)), \) \(\cdots\)

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 \(\pi \in P(\mathcal{H})\) such that \(s\) occurs in \(\pi.\)

**Update Points and Instantaneous Valuations.**

The observable behaviour of run \(\pi\) is a function \(I^\pi : \mathbb{R}_0^+ \to V\) which assigns to each point in time the valuation defined by an action during a non-zero delay in \(\pi,\) i.e. \(I^\pi(t) := f_{\ell_{i+1}}(t - t_i)\) where \(i \in \mathbb{N}_0\) is the (unique) index such that \(t_i < t < t_{i+1}\) 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 \(\pi\) above.

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

\[
\exists i \in \mathbb{N}_0 \cdot (\ell_i, v_i), t_i, a_{i+1} \xrightarrow{\lambda_{i+1}} (\ell_{i+1}, v_{i+1}), t_{i+1} \land t = t_{i+1}.
\]

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

\[
V^\pi(t) := \{I^\pi(t)\} \cup \{v_i \mid t_i = t\}.
\]

We use \(I^\pi : \mathbb{R}_0^+ \to \mathbb{R}\) and \(V^\pi : \mathbb{R}_0^+ \to 2^V \setminus \{\emptyset\}\) to denote the functions which are point-wise defined as \(I^\pi(t) := I^\pi(t)(x)\)

and \(V^\pi(t) := \{v(x) \mid v \in V^\pi(t)\}\) 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 \(\ell \in \ell_{\mathcal{N}}, 1 \leq i \leq n,\) and where \(\varphi\) is any arithmetical constraint over the variables in \(\mathcal{N}.\) A basic formula \(\beta\) is satisfied by a configuration \(s \in \text{Conf}(\mathcal{N}),\) denoted by \(s \models \beta,\) if and only if \(\ell_{s,i} = \ell, \ell_{s,i} \neq \ell,\) and \(v_\ell = v,\) 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 \(x, y \in \text{Var}\) be two variables of the hybrid automaton \(\mathcal{H}.\) We say \(x\) quasi-depends on \(y\) via function \(f,\) denoted by \(x \leadsto_f y,\) 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

\[
\forall \pi \in P(\mathcal{H}) \forall t \in \mathbb{R}_0^+ \cdot I^\pi(t) = f(V^\pi(t)).
\]

\(f\) 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 \(x\) 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 \(x, y \in \text{Var}\) be two variables of the hybrid automaton \(\mathcal{H}\) such that \(x \leadsto_f y\) with dependency function \(f.\) Then \(x \leadsto_f y\) if and only if \(\forall \pi \in P(\mathcal{H})\)

\[
V^\pi|_{\mathbb{R}_0^+} \land I^\pi(t) = f(V^\pi(t))
\]

where \(f \circ V^\pi\) is the set-valued function which yields the set of values obtained by applying \(f\) to all values in \(V^\pi.\)}}

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 \(x, y \in \text{Var}\) be two variables of the hybrid automaton \(\mathcal{H}.\) Let \(\pi \in P(\mathcal{H})\) 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.
\[
\Delta(t) = \begin{cases} 
1, & \text{if there exist } v \in \mathcal{V}^y(t), w \in \mathcal{V}^y(t) \text{ s.t. } v \neq f(w) \\
0, & \text{otherwise.}
\end{cases}
\]

Then \( x \simeq_f y \) implies \( \int_0^\infty \Delta(t) dt = 0 \). \( \diamond \)

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 \( x, y \in \text{Var} \) be variables of hybrid automaton \( \mathcal{H} \). We say \( x \) quasi-
depends on \( y \), denoted by \( \equiv \), 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 \( \text{Var} \) of \( \mathcal{H} \). \( \diamond \)

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

**Lemma 3.4.** Let \( x, y \in \text{Var} \) 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}_+^+ \bullet g(f(\mathcal{I}^x_\pi(t))) = \mathcal{I}^y_\pi(t). \)
\( \diamond \)

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 \((\text{Loc}, \text{Var}, \text{Lab}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init})\).

**Definition 3.** (Symbolic Semantics) The symbolic semantics of a hybrid automaton \( \mathcal{H} \) is defined by the region transition system \( T(\mathcal{H}) = (\text{Conf}(\mathcal{H}), \implies, C_{\text{ini}}) \) where
\[
\begin{align*}
\text{Conf}(\mathcal{H}) &= \text{Loc} \times (2^\text{Var} \setminus \{\emptyset\}) \text{ is the set of configurations,} \\
C_{\text{ini}} &= \{(\ell, Z) \in \text{Conf}(\mathcal{H}) \mid Z \subseteq \text{Init}(\ell)\} \text{ is the set of initial configurations, and}
\end{align*}
\]
\( \implies \) is the transition relation where
\[
\begin{align*}
\bullet & \text{ there is a discrete transition } (\ell_0, Z) \implies (\ell_1, Z') \\
& \text{induced by edge } (\ell_0, \alpha, \ell_1, Z_1) \in \text{Edge} \text{ if } Z' = \{ \nu' \mid \exists \nu \in Z \bullet (\nu, \nu' \in \mu \text{ and } \nu' \in \text{Inv}(\ell_1)\}, \text{ and}
\end{align*}
\]
\[
\begin{align*}
\bullet & \text{ there is a time transition } (\ell_0, Z) \implies (\ell_1, Z') \text{ if } Z' = \{ \nu' \mid \exists f \in \text{Act}(\ell_0) \bullet f(0) = \nu \land f(t) = \nu' \land \forall \nu \in Z \bullet f(t) \in \text{Inv}(\ell_1)\} \text{ if } \forall f \in \text{Act}(\ell_0) \bullet f(0) = \nu \land f(t) = \nu' \land \forall \nu \in Z \bullet f(t) \in \text{Inv}(\ell_1) \implies t = 0. \quad \diamond
\end{align*}
\]

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 \((\ell, Z)\) is called zero time, if and only if the invariant of \( \ell \) does not allow time to elapse from any valuation in \( Z \), i.e. if
\[
\forall f \in \text{Act}(\ell_0) \bullet f(0) = \nu \land f(t) = \nu' \land \forall \nu \in Z \bullet f(t) \in \text{Inv}(\ell_1) \implies t = 0.
\]
We write \( zt(\ell, Z) \) if and only if \((\ell, Z)\) is zero time. \( \diamond \)

Note that the values of quasi-dependent variables are not related by the dependency function at most in zero time con-
fugurations, and that they may assume any value in zero-time configurations without violating quasi-dependency. Stated
the other way around, quasi-dependency is violated in each non-zero time configuration where the values of the variables are
not related by the given dependency function. Thus our analyses 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 con-
fugurations, the relax operator only preserves the information which variables are related by which dependency function.

**Definition 5.** (Relax operator) The relax operator \( \text{rb} \) applied to the configuration \((\ell, Z)\) of \( \mathcal{H} \) over-approximates the region \( Z \) by a conjunction of the quasi-dependent functions it entails. Formally, \( \text{rb}(\ell, Z) = (\ell^{'}, Z^{'}) \) where
\[
Z' = \bigwedge \{ y = f(x) \mid x, y \in \text{Var} \land \forall g \in \text{Act}(\ell) \bullet g(0)(x) = \nu(\ell_0 \bullet g(t)(y) = \nu(y) \implies \forall t \in \mathbb{R}_+^+ \bullet g(t)(y) = f(g(t)(x))) \}.
\]

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-de-
dependency relations. In practice, the number of zero time con-
fugurations 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 \( \alpha_{\text{zt}} \) applied to a configuration \((\ell, Z)\) leaves
the configuration unaffected if the configuration is zero time and it abstracts it using the relax operator otherwise, i.e.

\[ \alpha_\ell(\langle \ell, Z \rangle) := \begin{cases} \langle \ell, Z \rangle & \text{if } zt(\ell, Z) \\ \text{relax}(\ell, Z) & \text{otherwise}. \end{cases} \]

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 \( (\ell, Z) \in \text{Conf}(\mathcal{H}) \) be a configuration.

1. \( Z \subseteq Z' \) if \( (\ell, Z') = \alpha_\ell((\ell, Z)) \), and
2. \( \alpha_\ell(\alpha_\ell((\ell, Z))) = \alpha_\ell((\ell, Z)) \).

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 \( \alpha_\ell \). 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 \( \mathcal{T}^\#, (\mathcal{H}) = (\text{Conf}(\mathcal{H}), \Longrightarrow^\#, C^\#) \) where

- \( \Longrightarrow^\# \subseteq \text{Conf}(\mathcal{H}) \times \text{Conf}(\mathcal{H}) \) is the transition relation.
  - There is a transition \( (\ell, Z) \Longrightarrow^\# (\ell', Z') \) if there is an edge \( (\ell, a, \mu, \ell') \in E \) such that \( Z' = \{ \nu' \mid 3 \nu \in Z \odot (\nu, \nu') \in \mu \odot \nu' \in \text{Inv}(\ell') \} \) is not empty, and
- \( C^\#_0 = \{ \alpha_\ell((\ell, Z)) \mid (\ell, Z) \in \text{Conf}(\mathcal{H}) \land Z = \text{Init}(\ell) \} \) 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 \( \mathcal{T}_i = (\text{Conf}(\mathcal{H}_i), \Longrightarrow_i, C_{\text{init}_i}) \), \( i \in \{1, 2\} \), is a binary relation on \( \text{Conf}(\mathcal{H}) \) satisfying the following properties:

1. For all \( \text{conf}_{\text{init}_1} \in C_{\text{init}_1} \) there exists \( \text{conf}_{\text{init}_2} \in C_{\text{init}_2} \) with \( \text{conf}_{\text{init}_1} \leq \text{conf}_{\text{init}_2} \),
2. \( (\ell_1, Z_1) \leq (\ell_2, Z_2) \) implies \( \ell_1 = \ell_2 \) and \( Z_1 \subseteq Z_2 \), and
3. \( (\ell_1, Z_1) \leq (\ell_2, Z_2) \) and \( (\ell_1, Z_1) \Longrightarrow (\ell'_1, Z'_1) \) implies that there exists a configuration \( (\ell_2, Z'_2) \) such that \( (\ell_2, Z_2) \Longrightarrow (\ell'_2, Z'_2) \) and \( (\ell'_1, Z'_1) \leq (\ell'_2, Z'_2) \).

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 dependency function \( f \) is checked on the abstract region graph \( \mathcal{T}^\# (\mathcal{H}) \) by checking, for each non-zero time configuration \( (\ell, Z) \) 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 \( t = 1 \) 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 \( y < 2x + 3 \lor y > 2x + 3 \) \land 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.
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 $\text{vars}(W) \subseteq \text{Var}$ to denote the set of variables that are constrained by $W$, i.e., $\text{vars}(W) = \{ v \in \text{Var} \mid \exists x \in W \mid v(x) \in W \}$. Analogously, for update $\mu \subseteq \forall \nu \in \text{V} \setminus \text{At}$, we use $\text{vars}(\mu)$ and $\text{vars}(f)$, to denote the set of variables that are constrained by $\mu$ and $f$, respectively. We use $\nu(\text{H})$ to denote the set of variables that are constrained by some initial condition, location invariant, update, or action in $\text{H}$.

We use $\text{SE}_Y(\text{H})$ to denote the set of simple resetting edges of hybrid automaton $\text{H}$ where only quasi-dependent variables from $Y \in \text{QD}_N$ are constrained by the conditional update and which have action $\tau$, i.e., $\text{SE}_Y(\text{H}) = \{(\ell, \tau, \mu, \ell') \in \text{Edge}(\text{H}) \mid \text{vars}(\mu) \subseteq Y\}$. We use $\text{CE}_Y(\text{H})$ to denote the set of complex resetting edges of $\text{H}$ where quasi-dependent and non-quasi-dependent variables occur in the conditional update, or which have a label different from $\tau$, i.e., $\text{CE}_Y(\text{H}) = \{(\ell, \tau, \mu, \ell') \in \text{Edge}(\text{H}) \mid \text{vars}(\mu) \setminus Y \neq \emptyset \vee (\text{vars}(\mu) \cap Y) \neq \emptyset \}$. We use $\text{CE}_Y(\text{H}) = \text{SE}_Y(\text{H}) \cup \text{CE}_Y(\text{H})$ to denote the set of resetting edges of $\text{H}$ wrt. $Y$, and $\text{RSE}_Y(\text{N})$ to denote the set of automata in $\text{N}$ which have a $Y$-resetting edge, i.e., $\text{RSE}_Y(\text{N}) = \{ \text{H} \in \text{N} \mid \text{CE}_Y(\text{H}) \neq \emptyset \}$. We use $\text{SE}_Y(\text{H})$ and $\text{CE}_Y(\text{H})$ to respectively denote the set of source and destination locations of simple and complex resetting edges of $\text{H}$. A location $\ell \in \text{Edge}(\text{H})$ is called reset (successor) location wrt. $Y$ in $\text{H}$ if and only if there is a resetting edge in $\ell \in \text{CE}_Y(\text{H})$ from $(\ell, \ell')$. We use $\text{RSE}_Y(\text{H}) (\text{RSE}_Y(\text{L}))$ to denote the set of reset (successor) locations wrt. $Y$ in $\text{N}$. We define $\text{RSE}_Y(\text{N}) := \bigcup_{\text{H} \in \text{QD}_N} \text{RSE}_Y(\text{H})$ and similarly $\text{RSE}_N$.

Definition 9. (Well-Formed Network) A network $\text{N}$ is called well-formed if and only if it satisfies the following restrictions for each set of quasi-dependent variables $Y \in \text{QD}_N$: (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 $C_Y, O_Y \in \text{R}$, i.e.

$\exists C_Y, O_Y \in \text{R} \forall (\ell, \ell', \mu, \ell'') \in \text{CE}_Y(\text{H}) \exists x \in Y \bullet$

$\forall v, v' \in \mu \bullet v|_{v(x)} = v'|_{v(x)}$\\
$\land \forall v \in (\nu(\ell) \cap \text{Inv}(\ell)) \bullet v(x) = C_Y$\\
$\land \forall v \in (\nu(\ell') \cap \text{Inv}(\ell')) \bullet v(x) = O_Y$,

where $\nu(\ell)$ denotes projection onto the $i$-th component.

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

$\exists e_1 = (\ell_1, a_1, \mu_1, e_1) \in \text{CE}_Y(\text{N}) \bullet e_1 \neq e_2 \implies \ell_1 \neq \ell_2$.

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

$\forall e \in \text{Lab} \bullet (\exists e = (\ell, \mu, \ell') \in \text{Edge}(\text{N}) \bullet e \in \text{CE}_Y(\text{N}))$

$\implies (\forall e = (\ell, \mu, \ell') \in \text{Edge}(\text{N}) \bullet e \in \text{CE}_Y(\text{N}))$.

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

$\forall e \in \text{Lab} \bullet (\exists e = (\ell, \mu, \ell') \in \text{Edge}(\text{N}) \bullet e \in \text{CE}_Y(\text{N}))$

$\implies \forall y \in Y \setminus \{x\} \bullet (\nu(\ell))(y) = \text{R}$.

Note that network $\text{N}$ from Figure 1 is well-formed. In the following we describe our transformation procedure $\text{K}$. It works with two given inputs, a well-formed network $\text{N} = \{ \text{H}_1, \ldots, \text{H}_n \}$ and a set of equivalence classes $\text{QD}_N$ of quasi-dependent variables. The output is the transformed network $\text{N}' = \{ \text{H}_1', \ldots, \text{H}_n' \} \cup \{ \text{CL}_i \mid \text{Y} \in \text{QD}_N \}$ which consists of hybrid automata $\text{H}_i' = \{ \text{Loc}^+_{\text{i}}(\text{H}_i), \text{Var}, \text{Lab}(\text{H}_i), \text{Edge}(\text{H}_i), \text{Act}(\text{H}_i), \text{Inv}(\text{H}_i), \text{Init}(\text{H}_i) \}$ which are obtained as modifications of the $\text{H}_i$ and one resetter automaton for each equivalence class $\text{Y} \in \text{QD}_N$.

The common set of variables $\text{Var}$ consists of the non-quasi-dependent variables in $\text{N}$, one representative $\text{rep}(\text{Y})$ for each equivalence class, and bookkeeping variables $\text{rst}^\text{N}_\text{Y}$ and $\text{rst}^\text{O}_\text{Y}$, $\text{N} \in \text{N}$. For each equivalence class $\text{Y} \in \text{QD}_N$, the set of labels of $\text{H}$ is extended by the fresh label $\text{reset}_Y$ if $\text{H} \in \text{RSE}_Y(\text{N})$.

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

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 $\text{rst}^\text{N}_\text{Y}$ is assigned 1, the assignments of all non-resetting edges with a reset location as source are changed such that $\text{rst}^\text{N}_\text{Y}$ is assigned 0, and the assignments of all resetting edges (including those redirected to have source $\ell_{\text{reset}}$) is changed such that $\text{rst}^\text{O}_\text{Y}$ is assigned 0. Note that $\text{rst}^\text{N}_\text{Y}$ may also be assigned 1 on resetting edges in case of loops. For each location, any action $f \in \text{Act}(\text{H})$ is restricted to $\text{Var}$ and modified such that $\text{rst}^\text{N}_\text{Y}$ and $\text{rst}^\text{O}_\text{Y}$ keep their value over time, and such that the evolution of the representative variables is unconstrained. Their evolution is

<table>
<thead>
<tr>
<th>Cl.</th>
<th>(n)</th>
<th>SMT-calls</th>
<th>states</th>
<th>(t) (s)</th>
<th>SpaceEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0.34</td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>0.64</td>
<td>1503</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>1.00</td>
<td>2003</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1.48</td>
<td>2503</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>17</td>
<td>4</td>
<td>0.91</td>
<td>1502</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>5.12</td>
<td>3502</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>278</td>
<td>16</td>
<td>25.72</td>
<td>7502</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1255</td>
<td>35</td>
<td>142.20</td>
<td>-</td>
</tr>
</tbody>
</table>
determined by the resetter automata. The actions of the locations \( \ell_{x,y} \) 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 \( \kappa \) to network \( \mathcal{N} \) from Figure 1 with the set of quasi-dependent variables \( Y = \{x,y\} \) yields the automaton \( \mathcal{H}_1 \) 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 \( \ell_{\text{inst}, Y} \) and \( \ell_{\text{dest}, Y} \) and two edges \((\ell_{\text{inst}, Y}, \text{reset}, \mu_1, \ell_{\text{dest}, Y})\) and \((\ell_{\text{dest}, Y}, \tau, \mu_2, \ell_{\text{inst}, Y})\).

The first edge obtains a guard \( \mu_1 \) which checks whether the representative variable of equivalence class \( Y \) equals \( C_Y \) and whether the \( rst_{1, Y} \) evaluate to 1. \( \mu_1 \) assigns all \( rst_{1, Y} \) to 0 and the representative variable to \( O_Y \). The second edge obtains a guard \( \mu_2 \) which checks whether \( rst_{2, Y} \) evaluates to 0, and assigns 1 to \( rst_{2, Y} \).

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 \( \ell_{\text{inst}, Y} \) inherits the invariant of the reset location with an outgoing edge updating \( x \).

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

Function \( \Omega \) treats queries for source or destination locations of resetting edges special, by referring these queries to the summary location \( \ell_{\text{dest}, Y} \). For instance, for a simple resetting edge \( e \in \mathcal{E}_Y(H) \) of some \( H \in \mathcal{N} \), the source location \( \ell \) 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 \( H \) did not reset yet, while other automata in \( \mathcal{RSE}_Y(N) \) may have reset their quasi-dependent variables already. In \( \mathcal{N}' \), all edges resulting from simple edges fire at once on the synchronisation on \( \text{reset}_Y \), so all source locations are left together. With this synchronisation, the resetter moves to \( \ell_{\text{dest}, Y} \), 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 \( \ell \) is reachable in \( \mathcal{N} \) if and only if, for \( \mathcal{N}' \), (i) \( \ell_{\text{dest}, Y} \) is reachable, or (ii) \( \ell \) is reached while being stable, i.e., not being in \( \ell_{\text{dest}, Y} \).

Note that \( \Omega(CF) \) consists of two parts: a transformation of \( CF \) by \( \Omega_0 \) 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 \( \ell_{\text{dest}, Y} \). 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 \( RI \), we only need to consider, e.g., \( O_Y \) and \( C_Y \) as values of \( \bar{x} \), 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 \( Y \in \mathcal{QD}_N \) be a set of quasi-dependent variables. A configuration \( s \in Conf(\mathcal{N}) \) is called stable wrt. \( Y \in \mathcal{QD}_N \) if and only if \( \forall x \in Y, x \neg \text{rep}(x) \land \nu(x) = f(\nu_x(\text{rep}(x))) \).

We use \( SC_Y \) to denote the set of all configurations that are stable wrt. \( Y \) and \( SC_Y \) to denote the set \( \bigcap_{Y \in \mathcal{QD}_N} SC_Y \) of globally stable configurations of \( \mathcal{N} \). Configurations not in \( SC_Y \) are called unstable.

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

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 \( \forall s_0 \exists s \in \mathcal{R}_2 \overset{\lambda_j}{\rightarrow} E_i, \ldots, s_{n-1} \overset{\lambda_j}{\rightarrow} E_i, s_n \in \Pi(H) \rightarrow \exists j \in [n] \wedge \lambda_j \in \mathcal{R}_2 \setminus \{0\} \wedge y \leq i < n \wedge \text{Edge}(H) \cap E_i = \emptyset \).

Here, we write \( s_0 \overset{\lambda_j}{\rightarrow} E_i, s_{i+1} \in \mathcal{N}^*, \) to denote that the transition \( s_i \overset{\lambda_j}{\rightarrow} s_{i+1} \) is justified by the set of edges \( E_i \); \( E_i \) is empty for time transitions, i.e., if \( \lambda_j \in \mathcal{R}_2 \times \text{Lab} \).

We say \( \mathcal{QD}_N \)-reset edges are post/pre delayed in well-formed network \( \mathcal{N} \) if and only if all edges originating in
\[ \Omega_0(\beta) = \begin{cases} 
(\ell \wedge \neg \ell_{\text{not}}, Y) \vee (\ell_{\text{not}}, Y \wedge \tilde{\ell}) 
& , \text{if } \beta = \ell, r \in \mathcal{SC}_Y(N) \cup (\mathcal{CL}_Y(N) \cap \mathcal{RL}_Y), Y \in \Omega \mathcal{D}_N. 
(-\ell \wedge -(\ell_{\text{not}}), Y) \vee (\ell_{\text{not}}, Y \wedge \tilde{\ell}) 
& , \text{if } \beta = -\ell, r \in \mathcal{SC}_Y(N) \cup (\mathcal{CL}_Y(N) \cap \mathcal{RL}_Y), Y \in \Omega \mathcal{D}_N. 
(\varphi[x/\tilde{x}] \text{ true }) \wedge \neg \ell_{\text{not}}(Y) \vee (\ell_{\text{not}}, Y \wedge \varphi) 
& , \text{if } \beta = \varphi, \varphi = \varphi[x/\tilde{x}] \text{ true }, Y \in \Omega \mathcal{D}_N. 
\beta 
& , \text{otherwise}.
\end{cases} \]

\[ \Omega(CF) = \exists z_1, \ldots, z_k \exists \ell_1, \ldots, \ell_m \text{ such that } \]
\[ \Omega_0(CF) \wedge \bigcap_{1 \leq i \leq k} (\ell_i \implies \tilde{z}_j = C_Y) \wedge \bigcap_{1 \leq i \leq m} (\ell_i \implies \tilde{z}_j = O_Y) \wedge \bigcap_{1 \leq i \leq m} (\ell_i \implies \ell_i \wedge \tilde{z}_j) \wedge \bigcap_{1 \leq i \leq m} (\ell_i \implies \tilde{z}_j \wedge \ell_{\text{in}, r}) \wedge \bigcap_{1 \leq i \leq m} (\ell_i \implies \ell_{\text{out}, r}). \]

**Table 2**: Formula transformation function \( \Omega \). \( \Omega_0(CF) \) denotes applying \( \Omega_0 \) to each basic formula in \( CF \).

reset (successor) locations are delayed, i.e., if for all \( e = (\ell, a, \mu, \ell') \in \text{Edge}(N), \ell \in \mathcal{R}_N \cup \mathcal{RL}_N \), implies that \( \ell \) is delayed.

There are sufficient syntactic criteria for an edge \( e = (\ell_0, a_0, \mu_0, \ell_1) \) being delayed. For instance, if \( (\ell_0, a_0, \mu_0, \ell_1) \) is the only incoming edge to \( \ell_1 \) and if the condition of \( \mu_0 \) is \((x \geq C \land x \leq C) \) and the condition of \( \mu_1 \) is \((x \geq D \land x \leq D) \) 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 \( \ell_1 \), \( \mu_0 \) updates \( x \), and the condition of \( \mu_1 \) 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 \( \mathcal{QD}_N \)-reset edges are prep/post delayed in FSN.

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

1. \( \forall s \in \text{Conf}(N) \exists r \in S \land \forall r \in S \exists s \in \text{Conf}(N) \).
2. For all configuration formulae \( CF \) over \( N \), \( \forall (s, r) \in S \land r \models \text{Conf}(N) \), \( \exists s' \in \text{Conf}(N) \) such that \( s' \models CF \).
3. For all \( (s, r) \in S \), if \( s \xrightarrow{\lambda} s' \) with \( (a) s, s' \notin \mathcal{SC}_Y \), then \( s \xrightarrow{\lambda} s' \), and justified by a simple resetting edge, or \( s \notin \mathcal{SC}_Y \), \( s' \notin \mathcal{SC}_Y \), then \( s \xrightarrow{\lambda} s' \), and justified by a simple resetting edge, then \( s \xrightarrow{\lambda} s' \) and \((s', r) \in S \).
4. For all \( (s, r) \in S \), if \( s \xrightarrow{\lambda} s' \) with \( (b) s \in \mathcal{SC}_Y \), \( s' \notin \mathcal{SC}_Y \), then \( s \xrightarrow{\lambda} s' \), and justified by a simple resetting edge, then \( s \xrightarrow{\lambda} s' \), and justified by a simple resetting edge, \( s \xrightarrow{\lambda} s' \), and justified by the set \( \mathcal{C}_Y \subseteq \mathcal{CE}_Y(N) \) of complex resetting edges wrt. \( Y \), or \( s, s' \notin \mathcal{SC}_Y \), where \( Y \in \mathcal{QD}_N \), and justified by the set \( \mathcal{CE}_Y \subseteq \mathcal{CE}_Y(N) \) of complex resetting edges wrt. \( Y \), or \( s, s' \notin \mathcal{SC}_Y \), then \( s \xrightarrow{\lambda} s' \).

**Theorem 5.2.** Let \( CF \) be a configuration formula over well-formed network \( N \) with \( \mathcal{QD}_N \)-reset delayed. \( CF \) is reachable in \( N \) iff \( CF \) is reachable in \( N' \).

**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. One of our benchmarks, 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].
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 \(rst_i^{\leq_n}\) and \(rst_i^{>\leq_n}\) by corresponding locations in the resetter automata.

### 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 straightforward, 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

projects/CaseStudyRepository/hybridqdv.
powerlink networks for distributed control and
[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
clock reduction: More networks, more queries. In
quasi-equal clocks in timed automata. In FORMATS,
unified lattice model for static analysis of programs by
construction or approximation of fixpoints. In POPL,
[9] C. Daws and S. Yovine. Reducing the number of clock
variables of timed automata. In RTSS, pages 73–81.
IEEE, 1996.
reachability properties using abstractions. In TACAS,
Schloss Dagstuhl, 2013.
systems by bisimulation. IEEE Trans. Automatic
bisimulation relations for constrained linear systems.
Approximate simulation relations for hybrid systems.
verification of linked data structures. In PLDI, pages
[20] G. Frehse et al. SpaceEx: Scalable verification of
[21] D. Dietsch, S. Feo Arenis, B. Westphal, and
A. Podelski. Disambiguation of industrial standards
through formalization and graphical languages. In RE,
solver for nonlinear theories over the reals. In CADE,
non-linear arithmetic constraint systems with complex