Abstraction and Counterexample-Guided Construction of ω-Automata for Model Checking of Step-Discrete Linear Hybrid Models[⋆]

Marc Segelken

OFFIS e.V.

Abstract. For the verification of reactive hybrid systems existing approaches do not scale well w.r.t. large discrete state spaces, since their excellence mostly applies to data computations. However, especially control dominated models of industrial relevance in which computations on continuous data are comprised only of subsidiary parts of the behavior, these large discrete state spaces are not uncommon. By exploiting typical characteristics of such models, the herein presented approach addresses step-discrete linear hybrid models with large discrete state spaces by introducing an iterative abstraction refinement approach based on learning reasons of spurious counterexamples in an ω-automaton. Due to the resulting exclusion of comprehensive classes of spurious counterexamples, the algorithm exhibits relatively few iterations to prove or disprove safety properties. The implemented algorithm was successfully applied to parts of industrial models and shows promising results.

Keywords: automata construction, counterexample guidance, iterative abstraction refinement, model-checking, step-discrete hybrid systems.

1 Introduction

For the analysis of discrete control systems, formal verification has already been successfully applied in recent years on industrial-sized controllers. However, the analysis of hybrid systems still represents a challenge, particularly with regard to controller models modeled and validated with CASE tools such as Statemate, Scade, Ascet and Simulink, which are typically open-loop discrete-time models combining a large discrete state space with a nontrivial number of floating point variables.

Among other approaches, a rich set of different abstraction techniques were developed for verifying hybrid models, transforming the inherently infinite state system into a finite-state model. The more sophisticated ones are usually based on iterative refinement techniques eliminating spurious counterexamples by refining the abstracted model for subsequent iterations, and by thus making the observed counterexample impossible to occur again in future runs. A prominent representative is, e.g., [CFH+03] where path fragments in the discrete state space are excluded. Other techniques limit the continuous dynamics to simple abstractions based on rectangular inclusions or polyhedrons such

⋆ This research was partially supported by the German Research Foundation (DFG) under contract SFB/TR 14 AVACS, see www.avacs.org.

Table 1. Open-loop industrial versus closed-loop academic models

Industrial models	discrete states	continuous variables *total/input/state*	regulation laws
Window lifting system (Ascet,BMW)	2^{26}	27/2/5	~ 60
Flight controller (Scade,Verilog)	2^{51}	423/7/25	~ 80
Desante, casts abstracted (Scade,Hispano-Suiza)	2^{1055}	358/14/0	8
Academic models			
Cruise Control System [SFHK04]	2^{4}	6/0/6	~ 24
Distributed Robot Control [AHH96]	2^{9}	12/0/12	< 1000
Mutual Exclusion example [ADI02]	2^{6}	3/0/3	~ 16

as in HyTech [HH94], PHAVer [Fre05], Checkmate [SK00] or d/dt [ADM02]. Their typical target models are hybrid systems where the continuous computations dominate while the discrete part of the system is only in charge of distinguishing between different modes such that the system can react by, e.g., applying different continuous control laws. Consequently, the existing approaches reflect these characteristics by focusing on the continuous items only, not considering the discrete fragment as a problem.

However, as Table 1 shows, industrial hybrid models might comprise considerable discrete fragments as well. A huge number of discrete states is to be seen alongside of only few different applied regulation laws. This effect is inevitably connected to the usage of discrete timers, validation- and error counters, different clocks and especially the parallel composition of interacting subcomponents including discrete ones such as state machines or communication protocols, which every bigger model naturally consists of. Such industrial models require algorithms capable of large discrete systems as well, an aspect that has been neglected by most research activities.

The approach presented in the following deals with such models by exploiting the relatively small number of different regulation laws. This is done by applying an iterative abstraction refinement that eliminates a comprehensive class of counterexamples represented by the spurious one by generalizing regulation law violations, leading to a considerable amount of refinement in each step and keeping the overall number of iterations needed to confirm or reject a safety property quite small. Since many different traces are spurious for equal reasons being the same or similar continuous computation sequences only starting in different discrete states, this is possible by excluding these continuous computation sequences in general, not only single discrete path fragments. The abstraction technique is conservative, meaning that no property gets a wrong affirmative result. The procedure is a semi-decision one, i.e. it might fail to prove a property in a bounded number of iteration steps, whereas bounded counterexamples can always be found.

As shown schematically in Figure 1, the procedure starts with a simple abstraction, a discrete automaton A_0 having the same structure as the hybrid automaton H. In each iteration, the spurious counterexample is analyzed, and minimal infeasible subsets (conflicts) of the computation on continuous items being implied by the counterexamples projection on the concrete hybrid model are determined. These subsets are sequences of applied regulation laws consisting of conjunctions of formulas guarding and describing the continuous state space transformations of transitions. By incrementally

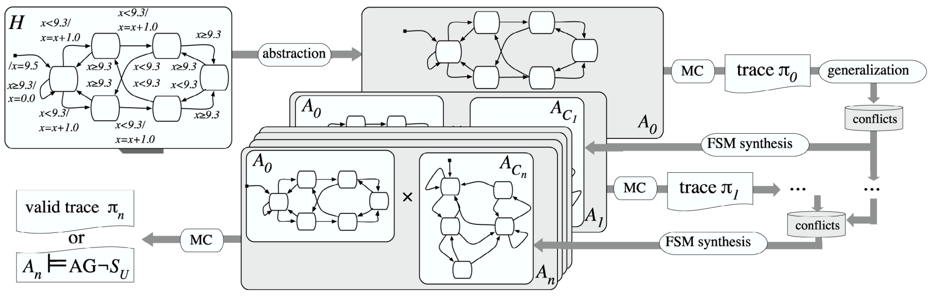


Fig. 1. Schematic overview on iterative refinement process

constructing a simple structured ω-automaton A_C with no fairness constraints that allows all runs except the ones containing any of the infeasible subsets detected so far, we get an automaton that prohibits all classes of known spurious counterexamples. With the parallel composition of $A_{C_{i+1}}$ being constructed based on the known conflicts in the i^{th} step and A_0 being the starting point of the iteration, we get the automaton $A_{i+1} = A_{C_{i+1}} \times A_0$ to be checked in the next iteration. Thus, we directly refine only $A_{C_{i+1}}$ and create the parallel composition A_{i+1} in each step, refining the overall model A_{i+1} indirectly.

The presented technique called ω-CEGAR (Counter-example guided abstraction refinement) was developed and advanced in the industrial context of the SafeAir project [GGB+03], which motivated the specialization to the practically important step-discrete hybrid automata, i.e. classical automata controlling continuous state variables without time-continuous evolution, thus following the synchrony hypothesis. Such automata are modeled by industrially applied CASE-tools such as SCADE, STATEMATE, ASCET, SILDEX, etc., and the herein presented abstraction refinement approach has already been extended to these as well. The abstraction approach shows promising results in parts of industrial case studies.

The paper is organized as follows: In Section 2 some mathematical definitions are introduced. Section 3 describes in detail the basic approach of the abstraction refinement based on ω-automata construction, followed by Section 4 presenting an enhancement of the approach. After presenting experimental results and discussing related work in Section 5 the paper is concluded with Section 6.

2 Preliminaries

2.1 Step-Discrete Hybrid Automata

Models developed with the previously mentioned CASE tools follow the synchrony hypothesis and assume that all computations are instantaneous. Therefore we consider step-discrete hybrid systems in the following. The definitions in this section originate from [CFH+03] and were adapted to step-discrete systems accordingly.

Definition 1 (Step-discrete Hybrid Automaton). A step-discrete hybrid automaton *is a tuple* $H = (Z, z_0, X, X_0, T, g, j)$ *where*

- *Z is a finite set of locations.*
- *$z_0 \in Z$ is an initial location.*
- *$X \subseteq \mathbb{R}^n$ is the continuous state space.*
- *$X_0 \subseteq X$ is the set of initial continuous states. The set of initial hybrid states of H is thus given by the set of states $\{z_0\} \times X_0$.*
- *$T \subseteq Z \times Z$ is the set of discrete transitions between locations*[1]*.*
- *$g : T \rightarrow 2^X$ assigns a guard set $g((z_1, z_2)) \subseteq X$ to $(z_1, z_2) \in T$.*
- *$j : T \rightarrow (X \rightarrow 2^X)$ assigns to each pair $(z_1, z_2) \in T$ a jump function that assigns to each $x \in g((z_1, z_2))$ a jump set $j((z_1, z_2))(x) \subseteq X$.*

We denote the set of all guard sets with $G = \{g(t)|t \in T\}$ and the set of all jump set functions with $J = \{j(t)|t \in T\}$. Note that both G and J are finite.

2.2 Semantics

The corresponding semantics is defined with the notion of transition systems:

Definition 2 (Transition System). *A* transition system *is a triple $TS = (S, S_0, E)$ with a (possibly infinite) state set S, an initial set S_0 and a set of transitions $E \subseteq S \times S$. We denote the set of all transition systems as $\mathfrak{T}$.*

Definition 3 (Path). *A* path *π of a transition system $TS = (S, S_0, E)$ is a (possibly finite) sequence $(s_0, s_1, s_2, ...)$ with $s_0 \in S_0$, each $s_i \in S$ and each pair of successive states $(s_i, s_{i+1}) \in E$. We denote the set of all paths of a transition system TS with $\overrightarrow{TS} := \bigcup_{m \in \mathbb{N}} \{(s_0, s_1, s_2, ..., s_m)|s_0 \in S_0, s_i \in S, (s_i, s_{i+1}) \in E\}$.*

During the iterative refinement itself only finite paths can occur as false negatives, since we restrict ourselves to safety properties. Thus infinite paths do not have to be considered in this paper.

Definition 4 (Semantics). *The* translational semantics *of a step-discrete hybrid automaton H is a transition system $TS_H = (S, S_0, E)$ with:*

- *$S = Z \times X$ being set of all hybrid states (z, x) of H,*
- *$S_0 = \{z_0\} \times X_0$ being the set of initial hybrid states and*
- *$E = (Z \times X) \times (Z \times X)$ being the set of transitions with $((z_1, x_1), (z_2, x_2)) \in E$, iff $\exists (z_1, z_2) \in T : x_1 \in g((z_1, z_2)) \wedge x_2 \in j((z_1, z_2))(x_1)$.*

2.3 Safety Properties

The presented procedure aims at the verification of safety properties, i.e. computes the reachability of a subset of states that are not considered safe. Let $S_U \subseteq S$ denote the unsafe states within a transition system $TS = (S, S_0, E)$. Then the model-checker has to compute whether

[1] For simplicity reasons, only one transition between two states is allowed. By doubling states, multiple transitions can easily be projected on such a restricted model.

- the system is safe w.r.t. S_U ($TS \models \mathbf{AG}\neg S_U$), formally $\nexists \pi \in \overrightarrow{TS} : \pi = (s_0, \ldots, s_m)$, $s_0 \in S_0, s_i \in S, s_m \in S_U$ or
- the system is unsafe w.r.t. S_U ($TS \not\models \mathbf{AG}\neg S_U$), formally $\exists \pi \in \overrightarrow{TS} : \pi = (s_0, \ldots, s_m)$, $s_0 \in S_0, s_i \in S, s_m \in S_U$.

If the model-checker is able to find an answer, it is either a path π showing a simulation run leading to an unsafe state $s \in S_U$, or the confirmation of TS to be safe w.r.t. the unsafe states S_U.

2.4 Abstraction

We use abstraction to get a purely discrete model to be checked by a finite state model-checker. In general an abstraction of a transition system TS is a transition system A that allows at least as much behavior as TS:

Definition 5 (Abstraction). *A transition system $A = (\hat{S}, \hat{S}_0, \hat{E})$ is an* abstraction *of a system $TS = (S, S_0, E)$, denoted $A \geq TS$, iff there exists a relation $\alpha \subseteq S \times \hat{S}$ such that:*

- $\hat{S}_0 = \{\hat{s}_0 | \exists s_0 \in S_0 : (s_0, \hat{s}_0) \in \alpha\}$ *and*
- $\hat{E} = \{(\hat{s}_1, \hat{s}_2) | \exists s_1, s_2 \in S : (s_1, s_2) \in E \wedge \{(s_1, \hat{s}_1), (s_2, \hat{s}_2)\} \subseteq \alpha\}$

Lemma 1. *For a transition system TS and its abstraction A, formally $A \geq TS$, the following condition always holds, if $\forall s_0 \in s_0 : \exists \hat{s}_0 \in \hat{S}_0 : \alpha(s_0, \hat{s}_0)$:*

$$\forall \pi = (s_0, s_1, \ldots, s_n) : \pi \in \overrightarrow{TS} \rightarrow \exists \hat{\pi} = (\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_n) \in \overrightarrow{A}, \forall_{0 \leq i \leq n} (s_i, \hat{s}_i) \in \alpha$$

This entails $A \models \mathbf{AG}\neg\hat{S}_U \implies TS \models \mathbf{AG}\neg S_U, \hat{S}_U = \{\hat{s} \in \hat{S} | \exists s \in S_U : (s, \hat{s}) \in \alpha\}$.

The previous lemma directly follows from the property of α. However, we cannot conclude $A \not\models \mathbf{AG}\neg\hat{S}_U \implies TS \not\models \mathbf{AG}\neg S_U$.

3 The ω-Automaton Based Iterative Abstraction Approach

3.1 Path Projection

During the analysis phase we need to retrieve the guard sets and jump set functions that are to be applied to the continuous state space if a path found in the abstract transition system is to be concretized. We achieve this by ensuring that any state $\hat{s} \in \hat{S}$ of our abstract transition system A can be projected to a discrete location $z \in Z$ of H by a function $\tilde{\alpha}^{-1} : \hat{S} \rightarrow Z$ which allows to reconstruct the transitions along with their associated guard- and jump set functions such that we can project paths of A to sequences of guard-/jump set function pairs.

Definition 6 (Guard-/Jump-set Sequence). *A* guard-/jump-set sequence *(abbrev. GJ-*sequence*) is defined by $((\gamma_0, \zeta_0), (\gamma_1, \zeta_1), \ldots, (\gamma_n, \zeta_n)), \gamma_i \in G, \zeta_i \in J$. We denote the set of all finite guard-/jump-set sequences with $C = \bigcup_{n \in \mathbb{N}} (G \times J)^n$.*

Definition 7 (Projecting Paths to GJ-sequences). *From a path $\hat{\pi} = (\hat{s}_0, ..., \hat{s}_n)$ of $A = (\hat{S}, \hat{s}_0, \hat{T})$ derived from TS_H by an abstraction relation α we compute the underlying GJ-sequence $c \in C$ with $\theta : \overrightarrow{A} \to C$:*

$$c = \theta(\hat{\pi}) = ((\gamma_1, \zeta_1), ..., (\gamma_n, \zeta_n)) \text{ with}$$
$$\gamma_i = g(t_i), \zeta_i = j(t_i), t_i = \tilde{\alpha}^{-1}((\hat{s}_i, \hat{s}_{i+1})) := (\tilde{\alpha}^{-1}(\hat{s}_i), \tilde{\alpha}^{-1}(\hat{s}_{i+1}))$$

In the following we refine the iterative abstraction process in Figure 1.

3.2 Initial Abstraction

Definition 8 (Initial Abstraction α_0). *The* initial abstraction *$A_0 = (\hat{Z}, \hat{z}_0, \hat{E})$ of TS_H of H with $\hat{Z} \cong Z$, $\hat{z}_0 \cong z_0$ and $\hat{E} \cong T$ is defined by a function $\alpha_0 : S \to \hat{Z}$ such that for any state $z_k \in Z$ there exists a state $\hat{z}_k \in \hat{Z}$ with*

$$\alpha_0((z_k, x)) = \hat{z}_k$$

The structure of transition system A_0 is isomorphic to the structure of H w.r.t. discrete locations and transitions while any conditions or operations on the continuous state space are omitted. Trivially by definition of A_0, $A_0 \succeq TS_H$.

Now A_0 can be analyzed by any standard model checker such as the *vis* model-checker [RGA+96] in our case, to check if a given safety property as defined in Section 2.3 is fulfilled. If no bad state in $\hat{S}_U$ is reachable we can conclude that also in TS_H no bad state in S_U is reachable, according to Lemma 1. Otherwise if we get a path $\hat{\pi}$, we proceed with the following analysis phase.

3.3 Analyzing Counterexamples

Given a path $\hat{\pi}$ we need to analyze whether it is a valid or a spurious counterexample and in the latter case we need to refine our transition system.

For this analysis, we first convert $\hat{\pi} = (\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n)$ into a guard-/jump-set sequence $c = \theta(\hat{\pi}) = ((\gamma_1, \zeta_1), ..., (\gamma_n, \zeta_n))$, which describes the step-wise transformations on the initial continuous state space X_0. Following the semantical definition of TS_H in Definition 4, the alternating application of an intersection with guard set γ_i and a transformation by jump set ζ_i on the state space X_i in the i^{th} step leads to a sequence $X_{seq} = (X_0, X_1, \ldots, X_n) \in 2^{X^{n+1}}$ of continuous state spaces with $X_i = \{x' | \exists x \in (X_{i-1} \cap \gamma_i) \wedge x' \in \zeta_i(x)\}$. If $X_n \neq \emptyset$ then $\exists (x_0, x_1, \ldots, x_n), x_i \in X_i$ and consequently there exists also a complying trace $\pi \in \overrightarrow{TS_H}$ with $\pi = ((\tilde{\alpha}^{-1}(\hat{z}_0), x_0), (\tilde{\alpha}^{-1}(\hat{z}_1), x_1), \ldots, (\tilde{\alpha}^{-1}(\hat{z}_n), x_n))$ representing a valid counterexample. For subsequent reuse we define the function $\mathcal{L} : C \times 2^X \to 2^X$ to compute X_n for a GJ-sequence c of length n and an initial continuous state space $\tilde{X}$ according to the previous explanation.

In practice we use the solver lp_solve [BEN04] to implement $\mathcal{L}' : C \times 2^X \to \mathbb{B}$ that computes whether $\exists (x_0, x_1, \ldots, x_n) \in X^{n+1}, x_0 \in X_0, x_1 \in X_1, \ldots, x_n \in X_n, (X_0, X_1, \ldots, X_n) = X_{seq}$ and the function $\tilde{\mathcal{L}}' : C \times 2^X \to \mathbb{B} \times X^{n+1}$ to include the discovered solution vector in the results as well. This is the point where we restrict ourselves to linear Hybrid Systems. However instead we could use e.g. flow-pipe approximation approaches to

Algorithm 1. $r: C \rightarrow 2^C \times 2^C$. Computing reduced conflict sets C_{ini} and C_{inv} from a conflict $c_\perp = ((\gamma_0, \zeta_0), \ldots, (\gamma_n, \zeta_n))$.

```
$(i, k, C_{ini}, C_{inv}) := (0, 0, \emptyset, \emptyset)$
while $C_{inv} \cup C_{ini} = \emptyset$ do
    while $i + k \leq n$ do
        if $i = 0 \wedge \mathcal{L}'(((\gamma_0, \zeta_0), \ldots, (\gamma_{i+k}, \zeta_{i+k})), X_0) = false$ then
            $C_{ini} := C_{ini} \cup \{((\gamma_0, \zeta_0), \ldots, (\gamma_{i+k}, \zeta_{i+k}))\}$
        if $i > 0 \wedge \mathcal{L}'(((\gamma_0, \zeta_0), \ldots, (\gamma_{i+k}, \zeta_{i+k})), X) = false$ then
            $C_{inv} := C_{inv} \cup \{((\gamma_i, \zeta_i), \ldots, (\gamma_{i+k}, \zeta_{i+k}))\}$
        $i := i + 1$
    end
    if $C_{inv} \cup C_{ini} = \emptyset$ then $k := k + 1, i := 0$
end
return $(C_{ini}, C_{inv})$                    % $r_{ini}: C \rightarrow 2^C$ returns $C_{ini}$, $r_{inv}: C \rightarrow 2^C$ returns $C_{inv}$
```

address non-linear models as well, without any other impact on the herein presented approach.

If $\hat{\pi}$ was a spurious counterexample indicated by $\mathcal{L}(\theta(\hat{\pi}), X_0) = \emptyset$, we extract conflicts from it as a basis for refining A through A_C as shown in Figure 1.

Definition 9 (Conflict). *A* conflict $c_\perp$ *is a GJ-sequence with* $\mathcal{L}(c_\perp, \tilde{X}) = \emptyset, \tilde{X} \subseteq X$. *If* $\tilde{X} = X$ *the conflict is termed invariant, if* $\tilde{X} = X_0$ *the conflict is termed initial.*

To get more comprehensive classes of conflicts, the shortest guard-/jump-set sequences still being initial or invariant conflicts are isolated by a reduction function $r: C \rightarrow 2^C \times 2^C$ defined by Algorithm 1, resulting in a pair of initial and invariant conflict sets.

3.4 Consideration of Refinement Strategy

As mentioned in the introduction, we construct an automaton A_C to be combined with A_0 in order to rule out comprehensive classes of all previously detected initial and invariant conflicts, $C_{\perp ini}$ and $C_{\perp inv}$, with

$$A_C \models \neg \left(\bigvee_{c_i \in C_{\perp ini}} \lambda(c_i) \right) \wedge \neg \mathbf{F} \left(\bigvee_{c_v \in C_{\perp inv}} \lambda(c_v) \right) \qquad (1)$$

with λ generating the LTL-Formula $\lambda(c) = ((\gamma_0, \zeta_0) \wedge \mathbf{X}((\gamma_1, \zeta_1) \wedge \mathbf{X}(... \wedge \mathbf{X}(\gamma_n, \zeta_n))))$ for a conflict $c = ((\gamma_0, \zeta_0), (\gamma_1, \zeta_1), ..., (\gamma_n, \zeta_n))$, using $(\gamma_i, \zeta_i) \in G \times J$ as atomic names of characters of an alphabet $\Sigma = G \times J$.

Due to the important observation that for industrial models, guard-/jump-set sequences associated with a path $\hat{\pi}$ and even more so smallest parts of them are replicated multiple times on other paths as well, this approach is reasonable. For a hybrid system dominated by discrete transitions, we have a huge state space with only few different guard-/jump-set pairs constituting the regulation laws replicated all over the transition system, formally:

$$\{(g(t), j(t)) | t \in T\}| \ll |Z| \lesssim |T| \qquad (2)$$

Table 1 shows the relationship between the amount of states[2] and guard-/jump-set pairs (regulation laws) for some examples. For the industrial models the number of pairs was determined empirically by observed occurrences in simulation runs and iterative refinements. This property of control dominated systems in practice is fundamental for the presented approach in this paper and is exploited extensively by ruling out all replications of conflicting guard-/jump-set sequences in the abstract model in one sweep.

3.5 Construction of ω-Automaton

To construct an ω-automaton A_C satisfying (1) we could apply existing LTL-to-Bchi translation algorithm such as [SB00]. However, since our formulas have a special structure, we can apply a dedicated incremental algorithm generating a very small co-1-accepting ω-automaton. As table 2 shows later, such a dedicated algorithm is much more efficient and generates significant smaller automata.

We apply the following algorithm for constructing an ω- and a regular automaton A_{C_ω} and A_{C_R} addressing invariant and initial conflicts each and compose the final ω-automaton A_C of both of them afterwards.

The ω-automaton is a Bchi automaton $\mathcal{A}_{C_\omega} = (Q_\omega, q_{\omega_0}, \Sigma, T_\omega, F_\omega) \in \mathfrak{B}$, with Q_ω being the set of states, q_{ω_0} being the initial state, $\Sigma = G \times J$ consisting of all guard-/jump set function pairs, $T_\omega \subseteq Q_\omega \times \Sigma \times Q_\omega$ being the transition relation and $F_\omega \subseteq Q_\omega$ being the set of accepting states. The regular $\mathcal{A}_{C_R} = (Q_R, q_{R_0}, \Sigma, T_R, F_R) \in \mathfrak{R}$ is a similar tuple, applying the classical acceptance condition for final words.

During construction, the automaton will have non-deterministic auxiliary transitions required to inherit transitions from other states tracking shorter words with matching prefixes. To identify such transitions a partial order $<$ on states is introduced based on a distance-to-default-state metrics. Such information can be efficiently locally computed and maintained for each state throughout construction. Based on such information we define the function $\mathit{tgt} : Q \times \Sigma \to Q$ to return the most distant state q reachable by a transition (p, δ, q) for a given δ.

Starting with the automaton $A_{C_\omega} = (\{q_0\}, q_0, \Sigma, \{(q_0, \delta, q_0) | \delta \in \Sigma\}, \{q_0\})$ with the default state q_0 accepting any infinite word, Algorithm 2 is used to incrementally add finite words $(\delta_0, \delta_1, \ldots, \delta_n)$ such that $A_{C_\omega} \models \neg\mathbf{F}(\delta_0 \wedge \mathbf{X}(\delta_1 \wedge \mathbf{X}(\cdots \wedge \mathbf{X}\delta_n)))$.

After all sequences have been added, auxiliary transitions are removed by a function $\mathit{strip} : \mathfrak{B} \to \mathfrak{B}$, keeping only the transitions $\{(p, \delta, q) \in T | q = \mathit{tgt}(p, \delta)\}$.

Finally, the automaton is efficiently minimized by Algorithm 3. The size of this ω-automaton is not monotonically increasing since adding conflicts might enable new minimization possibilities leading even to reduction. An extension of the algorithm not being described in detail due to space constraints exploits this observation by probing potential sequences that would have such a benefit. If confirmed as conflicts, they are added to the automaton as well, reducing its size while covering more conflicts at the same time.

[2] Since guard-/jump-set pairs are replicated over transitions and not states, statistics on transitions would have been more accurate, but are not accessible for technical reasons. However since the number of transitions always outnumbers the number of (reachable) states, the latter is a safe lower bound.

Algorithm 2. $Add_\omega : \mathfrak{B} \times \Sigma^* \to \mathfrak{B}$. Adding $(\delta_0, \delta_1, \ldots, \delta_n)$ to $A_{C_\omega} = (Q, q_0, \Sigma, T, F)$

```
$p := q_0$
for $0 \leq i \leq n$ do
    $q := tgt(p, \delta_i)$
    if $p < q \vee q \notin F$ then
        $p := q$
    else
        $Q := Q \cup \{q'\}$ with $q'$ being a new state
        $H_p := \{h | \exists p' \in Q, \delta \in \Sigma : \{(p', \delta, p), (p', \delta, h)\} \subseteq T, p < h\}$
        $L_{q'} := \{l | (p, \delta_i, l) \in T, l < q'\}$
        $T := T \cup \bigcup_{h \in H_p \cup \{p\}} \{(h, \delta_i, q')\} \cup \bigcup_{l \in L_{q'}} \{(q', \delta, r) | \exists (l, \delta, r) \in T\}$
        if $i \neq n$ then $F := F \cup \{q'\}$
        $p := q'$
end
return $(Q, q_0, \Sigma, T, F)$
```

Algorithm 3. Minimization of regular- and ω-automaton (Q, q_0, Σ, T, F)

```
$M := \{Q \backslash F\}$
foreach $M_k \in M$ do
    foreach $q_i, q_j \in M_k, q_i \neq q_k$ do
        if $\forall p \in Q, \delta \in \Sigma : \exists (q_i, \delta, p) \in T \Leftrightarrow \exists (q_j, \delta, p) \in T$ then
            $T := T \cup \{(p, \delta, q_i) | \exists (p, \delta, q_j) \in T\}$
            $T := T \setminus (\{(p, \delta, q_j) \in T\} \cup \{(q_j, \delta, p) \in T\})$
            $F := F \setminus \{q_j\}$
            $Q := Q \setminus \{q_j\}$
            $M := (M \setminus \{M_k\}) \cup \{p | \exists \delta \in \Sigma : (p, \delta, q_i) \in T\}$
    end
end
return $(Q, \{q_0\}, \Sigma, T, F)$
```

The regular automaton for conflicts of C_{ini} is constructed with a similar algorithm $Add_R : \mathfrak{R} \times \Sigma^* \to \mathfrak{R}$ by starting from $A_{C_R} = (\{q_0, fin\}, q_0, \Sigma, \{(q_0, \delta, fin) | \delta \in \Sigma\}, \{q_0, fin\})$, using $T := (T \backslash \{(p, \delta_i, fin)\}) \cup \{(p, \delta_i, q)\} \cup \{(p, \delta, fin) | \delta \in \Sigma \backslash \{\delta_i\} \wedge \nexists (p, \delta, r) \in T\}$ as transitions computation, making the auxiliary sets H_p and $L_{q'}$ obsolete.

Cross Product. Both automata A_{C_R} and A_{C_ω} are composed in parallel to a cross-product automaton $A_C = (Q, q_0, \Sigma, T_C, F)$ with $Q = Q_R \times Q_\omega$, $q_0 = (q_{R_0}, q_{\omega_0})$, $F = F_R \times F_\omega$ and $T_C = \{((q_{R_1}, q_{\omega_1}), \sigma, (q_{R_2}, q_{\omega_2})) | (q_{R_1}, \sigma, q_{R_2}) \in T_R, (q_{\omega_1}, \sigma, q_{\omega_2}) \in T_\omega\}$, which is the basis for the final composition of A.

Consideration of Partitioning. The partitioning $\mathfrak{P} = \{X_{q_1}, \ldots, X_{q_n}\} \subseteq 2^X$ of the continuous state space X can be envisioned as $n = |Q|$ partitions induced by the states Q of A_C. Let C_q be the set of all GJ-sequences leading to state $q = (q_R, q_\omega) \in Q$. Then each partition $X_q \in \mathfrak{P}$ is described by $\chi : Q \to X$ with

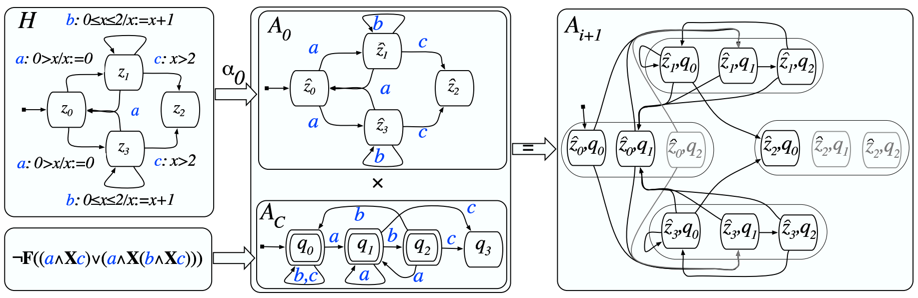


Fig. 2. Simplified example of abstraction refinement. Given an unsafe state z_2, the counterexamples $\hat{\pi} = (\hat{z}_0, \hat{z}_1, \hat{z}_2)$ and $\hat{\pi} = (\hat{z}_0, \hat{z}_1, \hat{z}_1, \hat{z}_2)$ are ruled out in A_{i+1} starting from $(\hat{z}_0, q_0)$. A_C is the conflict Bchi automaton, $A_C \models \neg\mathbf{F}((a \wedge \mathbf{X}c) \vee (a \wedge \mathbf{X}(b \wedge \mathbf{X}c)))$, with $a = (\{x|0 > x\}, x \mapsto 0), b = (\{x|0 \leq x \leq 2\}, x \mapsto x+1), c = (\{x|x > 2\}, x \mapsto x)$.

$$\chi(q) = \begin{cases} \bigcup_{c \in C_q} \mathcal{L}(c, X_0) & \text{if } q_R \neq fin \\ \bigcup_{c \in C_q} \mathcal{L}(c, X) & \text{if } q_R = fin \end{cases}$$

Proof: Let H_C be the isomorphic mapping of A_C to a step-discrete Hybrid automaton. Then its semantics is a transition system TS_{H_C} with states $Q \times X$. It is obvious that any path in TS_{H_C} leading to a state $(q, x) \in Q \times X$ entails a GJ-sequence $c \in C_q$. Since $\mathcal{L}$ was derived from the translational semantics, by its definition $X_q = \mathcal{L}(c, X)$ describes exactly the set of reachable continuous states X_q such that $(q, x), x \in X_q$.

3.6 Refinement of A_{i+1}

For equation (1) to be valid not only for A_{C_j} but also for A_j, we compose $A_0 = (\hat{Z}, \hat{Z}_0, \hat{E})$ and A_{C_j} in parallel by using a cross-product-similar combination of both: $\dot{\times} : \mathfrak{T} \times \mathfrak{B} \to \mathfrak{T}$ such that $A = A_0 \dot{\times} A_C = (\hat{S}, \hat{S}_0, \hat{T})$ with

- $\hat{S} = \hat{Z} \times Q$
- $\hat{S}_0 = \hat{Z}_0 \times \{q_0\}$
- $\hat{T} = \{((\hat{z}_1, q_1), (\hat{z}_2, q_2)) | (\hat{z}_1, \hat{z}_2) \in \hat{E} \wedge (q_1, \sigma, q_2) \in T_C \wedge \exists t \in T : t = \tilde{\alpha}^{-1}((\hat{z}_1, \hat{z}_2)) \wedge \sigma = (g(t), j(t)) \wedge q_2 \in F\}$

Algorithm 4 summarizes all previously detailed steps, and Figure 2 shows a very simple example of the abstraction refinement for one iteration.

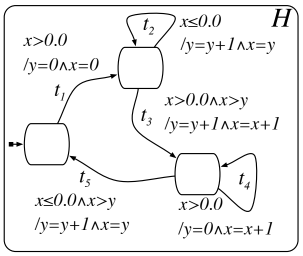
With the previous construction approach for a state $(z, x) \in Z \times X = S$ of the infinite state space of the trace transition system TS_H of H and a state $(\hat{z}, q) \in \hat{Z} \times Q = \hat{S}$ of A_j the *general abstraction* relation α is given by

$$\alpha = \{((z, x), (\hat{z}, q)) \in (Z \times X) \times (\hat{Z} \times Q) | \tilde{\alpha}^{-1}(\hat{z}) = z \wedge x \in \chi(q)\}$$

This follows directly from the construction of A_j and the partitioning χ. It is obvious that this relation fulfills Definition 5, thus $A_j \succeq TS_H$.

Algorithm 4. ω-CEGAR process, returns *true* or path $\pi = (s_0, \ldots, s_n) \in \overrightarrow{TS_H}, s_n \in S_U$

```
$A_{C_R} := (\{q_0, \mathit{fin}\}, q_0, \Sigma, \{(q_0, \delta, \mathit{fin}) | \delta \in \Sigma\}, \{q_0, \mathit{fin}\})$
$A_{C_\omega} := (\{q_0\}, q_0, \Sigma, \{(q_0, \delta, q_0) | \delta \in \Sigma\}, \{q_0\})$
$A := A_0$
while $A \not\models \mathbf{AG}\neg\hat{S}_U$ do                                   % Model-Checker run
    $\hat{\pi} = (\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n), \hat{z}_n \in \hat{S}_U$                        % Path from Model-Checker
    $(result, (x_0, x_1, \ldots, x_n)) := \tilde{\mathcal{L}}'(\theta(\hat{\pi}), X_0)$
    if $result = false$ then                                    % spurious counterexample
        foreach $c_\perp \in r_{ini}(\theta(\hat{\pi}))$ do $A_{C_R} := Add_R(A_{C_R}, c_\perp)$ end
        foreach $c_\perp \in r_{inv}(\theta(\hat{\pi}))$ do $A_{C_\omega} := Add_\omega(A_{C_\omega}, c_\perp)$ end
        $A := (Minimize(strip(A_{C_R})) \times Minimize(strip(A_{C_\omega})) \dot{\times} A_0)$
    else return $\pi := ((\tilde{\alpha}^{-1}(\hat{z}_0), x_0), (\tilde{\alpha}^{-1}(\hat{z}_1), x_1), \ldots, (\tilde{\alpha}^{-1}(\hat{z}_n), x_n))$ % valid path
end
return true                                                     % $TS_H \models \mathbf{AG}\neg S_U$
```



Guard Supersets

$\tilde{\gamma}_1 = \{x | x > 0\}$
$\tilde{\gamma}_2 = \{x | x \leq 0\}$
$\tilde{\gamma}_3 = \{x | x > y\}$

Jump Set Projections

$\tilde{\zeta}_{x_1} = (x \mapsto 0)$ $\quad \tilde{\zeta}_{y_1} = (y \mapsto 0)$
$\tilde{\zeta}_{x_2} = (x \mapsto y)$ $\quad \tilde{\zeta}_{y_2} = (y \mapsto y+1)$
$\tilde{\zeta}_{x_3} = (x \mapsto x+1)$

Grouping

$\tilde{\gamma}_1 \supseteq (g(t_1) \cup g(t_3) \cup g(t_4))$
$\tilde{\gamma}_2 \supseteq (g(t_2) \cup g(t_5))$
$\tilde{\gamma}_2 \supseteq (g(t_3) \cup g(t_5))$

$\tilde{\zeta}_{x_1} \mapsto \{j(t_1)\}$
$\tilde{\zeta}_{x_2} \mapsto \{j(t_2), j(t_5)\}$
$\tilde{\zeta}_{x_3} \mapsto \{j(t_3), j(t_4)\}$
$\tilde{\zeta}_{y_1} \mapsto \{j(t_1), j(t_4)\}$
$\tilde{\zeta}_{y_2} \mapsto \{j(t_2), j(t_3), j(t_5)\}$

Fig. 3. Simple example for syntactic creation of guard supersets and and jump set projections

4 Further Generalization of Conflicts

By using subsets $(\tilde{\gamma}, \tilde{\zeta}) \in \Sigma' \subseteq 2^{G \times J}$ instead of elements $(\gamma, \zeta) \in \Sigma \subseteq G \times J$, we can generalize conflicts having common reasons. Different guard sets $\gamma_1, \ldots, \gamma_m$ are subsumed by guard supersets $\tilde{\gamma}$ such that $\tilde{\gamma} \supseteq (\gamma_1 \cup \cdots \cup \gamma_m)$. Jump sets ζ are generalized by their projection $\tilde{\zeta}$ on fewer or single dimensions. Such $\tilde{\zeta}$ comprise all $\zeta_1, \ldots, \zeta_k$ having the same projection. As the example in figure 3 shows, reasonable guard supersets and jump set projections can even be computed syntactically.

With a function r extended accordingly to further generalize the conflicts with the introduction of $(\tilde{\gamma}, \tilde{\zeta})$ characters as described above, this generalization leads to dramatically reduced iteration numbers, since many similar conflicts are now comprised by one single sequence of sets of guard-/jump sets.

With the previously described construction of A_C, this automaton is no longer deterministic, since each (γ, ζ) might map to several of the sets described above. We determinate it with a transformation intuitively considering A_C as a directed graph with attributed edges with a new operational semantics where each node $q_j \in Q = \{q_0, q_1, \ldots, q_n\}$ represents a boolean variable $b_j \in \mathbb{B}$ being computed by a function $b^* : \{0, \ldots, n\} \times \Sigma \times \mathbb{B}^n \to \mathbb{B}$ with

$$b^*(j, \delta, (b_0, b_1, \dots, b_n)) = \begin{cases} 1 & \text{iff } j = 0 \\ \bigvee_{q_i \in Q:(q_i,\delta',q_j)\in T,[q_j]=j,\delta\in\delta',q_i<q_j} b_{[q_i]} & \text{iff } j > 0 \end{cases}$$

using a function $[] : Q \to \mathbb{N}$ with $[q_i] = i$. Thus we use the structure of $A_C = (Q, q_0, \Sigma, T, F)$ to create a deterministic automaton A^*_C such that $A^*_C = (Q^*, q^*_0, \Sigma, T^*, F^*)$ with $Q^* = \{(1, b_1, b_2, \dots, b_n) | b_i \in \mathbb{B}\}$, $q^*_0 = (1, 0, ..., 0)$, the transitions $T^* := \{((b_0, \dots, b_n), \delta, (b'_0, \dots, b'_n)) \in Q^* \times \Sigma \times Q^* | b'_j = b^*(j, \delta, (b_0, \dots, b_n))\}$ and the accepting states $F^* := \{(1, b_1, \dots, b_k, \dots, b_n) \in Q^* | \forall k \in \mathbb{N}, p \in Q \setminus F : [p] = k \implies b_k = 0\}$.

Table 2. Comparison of ω-automaton construction and general LTL-to-Bchi automata construction implementation Wring 1.1.0 based on [SB00]

	ω-construction		Wring 1.1.0	
LTL formula	states	time[s]	states	time[s]
$\neg\mathbf{F}(a\wedge\mathbf{X}(b\wedge X(b\wedge\mathbf{X}c)))$	8	0.1	20	0.6
$\neg\mathbf{F}(a\wedge\mathbf{X}(b\wedge\mathbf{X}c) \vee b\wedge\mathbf{X}(e\wedge\mathbf{X}(f\wedge\mathbf{X}d)))$	32	0.1	180	100.2
$\neg\mathbf{F}(a\wedge\mathbf{X}(b\wedge\mathbf{X}c) \vee b\wedge\mathbf{X}(e\wedge\mathbf{X}(f\wedge\mathbf{X}d)) \vee x\wedge\mathbf{X}c)$	64	0.1	288	551
$\neg\mathbf{F}(a\wedge\mathbf{X}(b\wedge\mathbf{X}c) \vee b\wedge\mathbf{X}(e\wedge\mathbf{X}(f\wedge\mathbf{X}d)) \vee c\wedge\mathbf{X}(f\wedge\mathbf{X}(g\wedge\mathbf{X}h)))$	256	0.1	2160	161871

This is the automaton referred to in Table 2 being compared to other LTL-to-Bchi translations, which also have non-mutual exclusive atomic propositions. A^*_C is certainly no longer minimal and of considerable size. However, we will see that this conflict generalization dramatically reduces the required iterations.

5 Experimental Results and Related Work

The ω-CEGAR approach was successfully applied to industrial examples ranging up to a hundred state bits and dozens of continuous variables. Table 3 gives an overview for two example models. The car window lifting system is a model from BMW which is modeled in ASCET. Depending on HMI interface and sensors it controls the engine lifting the car window, also maintaining its current position. The reachability of certain window positions was computed. The Flight Controller example is modeled with SCADE and controls the altitude depending on pilot command and sensor readings. The model contains three-dimensional vectors for positions and velocities, including plausibility computation. Here, various reachability analyses refering to expected reactions to pilot commands in a Normal Operations Mode (*NO*) were made. For two of these, Figure 4 (a) and (b) shows typical evolutions of quantities during the iteration process.

Figure 4 (c) shows the process for the same proof as (a), but without using sequences of sets of GJ-pairs as introduced in the previous section. The difference clearly reveals the benefit of such a conflict generalization.

Considering the size of the discrete state space $|Z|$ in the examples, we have remarkably few iterations until getting valid traces. Especially the case where the safety property was fulfilled and the bad state was not reachable as, e.g., in Proof 3 of the Flight Controller system deserves some attention. Here, after only 7 iterations and 13 generalized conflicts, the approach was able to prove the non-reachability. Considering the

Table 3. Experimental results using conflict generalization. Numbers refer to cone-reduced models. Table shows number of discrete locations, number of continuous dimensions (*inputs+state-based*), size of Σ / Σ', number of conflicts, iterations, final path length, size of Q representing state bits of A_C^* and total runtime including integration overhead.

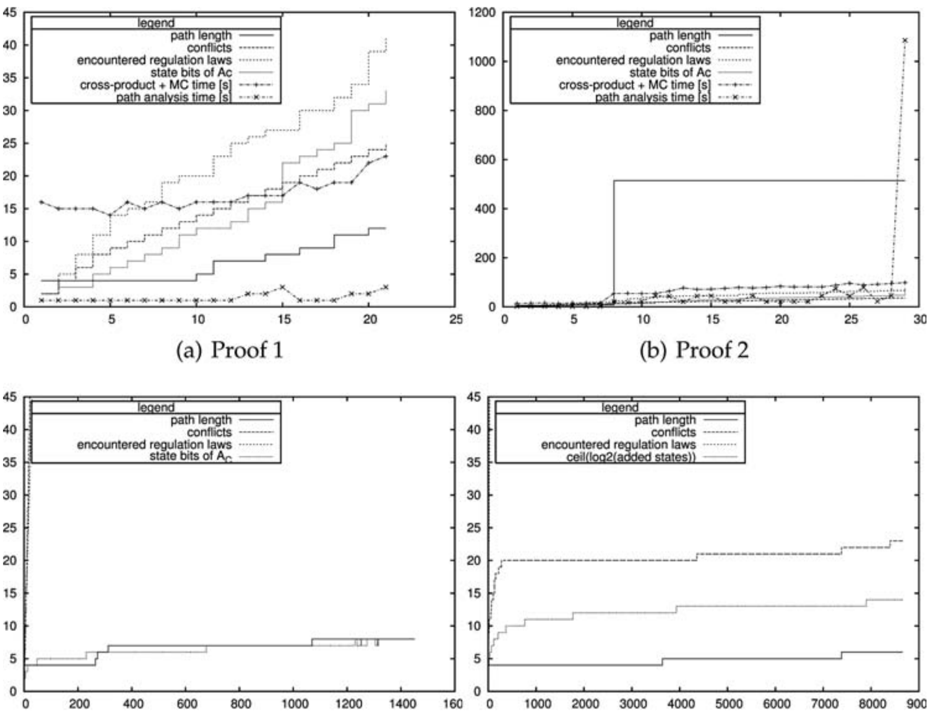
	Model / Proof	$\|Z\|$	dimensions	$\|\Sigma\|/\|\Sigma'_{ini}\|/\|\Sigma'_{inv}\|$	—$C_{ini}\|/\|C_{inv}\|$	iter	$\|\pi\|$	$\|Q\|$	time
	Flight Controller System								
1	$\mathbf{AG}(NO \implies p), p := (\Delta v_x = 0)$	2^{35}	5+18	41/2/43	1/24	21	13	33	16 min
2	$\mathbf{AG}(NO \wedge p \implies \mathbf{X}p)$	2^{35}	5+18	67/2/60	1/36	29	514	78	75 min
3	Proof 2 on corrected model	2^{31}	5+18	26/2/22	1/13	7	$\nexists\pi$	12	4 min
	Car Window Lifting System								
4	$\mathbf{EF}(pos_1 \leq pos_{window} \leq pos_2)$	2^{26}	2+3	92/12/63	14/79	59	10	41	52 min
5	$\mathbf{EF}(pos_{window} > pos_3)$	2^{26}	2+3	59/11/42	13/35	32	$\nexists\pi$	18	19 min

diameter of 513 of that model meaning that internally, the model checker had to analyze that many steps of the model until the fix-point was reached, the result is quite remarkable and demonstrates the power of the approach when it comes to certification issues.

Related Work. Among the various approaches on abstraction refinement to model-check hybrid models, most commonalities with the herein presented approach seem to be shared by two of them. First, there is the INFINITE-STATE-CEGAR algorithm [CFH⁺03] which also uses a fully automated iterative refinement technique. For any spurious counterexample identified as such by a polyhedral over-approximation of successor states, the corresponding path fragment of length n in the abstract model is ruled out by replicating up to $n-1$ states and modifying concerned transitions accordingly such that any other trace not containing the spurious path fragment is still observable. However, this does not prevent false negatives in other areas of the model where the same GJ-sequence is linked to different locations. According to the previously made observation in equation (2), this omission might lead to a huge number of iterations with false negatives caused by reasons already detected. The advantage of a slowly growing abstract model size is easily turned down by the huge number of iterations required for the herein targeted model class.

By including source- and target states in the alphabet characters, $\Sigma = \{(z_1, z_2, g(t), j(t)) | t = (z_1, z_2) \in T\}$, we modify our algorithm to exclude exactly the same path fragments that would be ruled out in [CFH⁺03], making a direct comparison w.r.t. abstraction refinement iterations possible. Figure 4 (d) shows the different evolution of a still uncompleted iteration process. Being compared to (a), it confirms the above statements w.r.t. the given example.

Second, an analysis via predicate abstraction approach described in [ADI02] constructs an automaton with 2^k states which is composed in parallel with the abstract model to rule out spurious counterexamples. Based on a set of k predicates, it is computed in advance which transitions in the added automaton are possible. The fixed size of the automaton results from the reservation of one boolean variable per predicate, encoding its truth-value. Refinement is realized by manually adding additional predicates, making the approach only half-automatic with no counterexample guidance. The



(a) Proof 1 (b) Proof 2

(c) Proof 1, no generalization, incomplete (d) Proof 1, mimicking the strategy of [CFH+03] using ZZGJ-sequences, incomplete

Fig. 4. Evolutions of iterative refinement, x-coordinate shows iteration steps

approach exploits the model characteristics of equation (2), but besides the required manual intervention, the predicates are only state-expressions with no temporal operators, making it impossible to directly rule out spurious counterexamples based on multi-step conflicts. Disregarding problems of manual intervention and automaton construction, an extension with LTL-formula predicates would make that approach more similar to the herein presented one. However, recent research activities described in [ADI03] automate the process of finding new predicates by looking for predicates for separation of polyhedra, thus following a different strategy.

6 Conclusion

In this paper an iterative abstraction refinement approach called ω-CEGAR for verifying step-discrete hybrid models exploiting the characteristics of control dominated models being observed in industrial practice was presented. The small number of applied regulation laws leading to vast cases of recurrence of continuous state space computations throughout different discrete transition sequences is exploited by forbidding impossible continuous computation sequences globally if only a single representative is detected. The construction of an ω-automaton being composed in parallel with the coarsely abstracted original model realizes this in an efficient way. Many iterations of model-checker and path validation runs being the most costly operations can be saved. The application of the ω-CEGAR approach on parts of industrial models already shows its efficacy on the targeted class of models, especially in comparison to the INFINITE-STATE-CEGAR approach.

Acknowledgements

The author likes to thank Werner Damm, Martin Frnzle, Hardi Hungar, Hartmut Wittke, Christian Herde and Tino Teige for many valuable discussions and feedback on previous versions of this paper.

References

[ADI02] Alur, R., Dang, T., Ivančić, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 35–48. Springer, Heidelberg (2002)

[ADI03] Alur, R., Dang, T., Ivancic, F.: Counter-example guided predicate abstraction of hybrid systems. In: 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (April 2003)

[ADM02] Asarin, E., Dang, T., Maler, O.: Eugene Asarin, Thao Dang, and Oded Maler. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 365–370. Springer, Heidelberg (2002)

[AHH96] Alur, R., Henzinger, T.A., Ho, P.: Automatic symbolic verification of embedded systems. In: Real-Time, I.E.E.E. (ed.) IEEE Real-Time Systems Symposium, pp. 2–11 (1996)

[BEN04] Berkelaar, K., Eikland, M., Notebaert, P.: Open source (mixed-integer) linear programming system. In: Eindhoven University of Technology (May 2004)

[CFH$^+$03] Clarke, E.M., Fehnker, A., Han, Z., Krogh, B.H., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. Found. Comput. Sci. 14(4), 583–604 (2003)

[Fre05] Frehse, G.: PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)

[GGB$^+$03] Gaudre, T., Guillermo, H., Baufreton, P., Goshen, D., Cruz, J., Dupont, F., Leviathan, R., Segelken, M., Winkelmann, K., Halbwachs, N.: A methodology and a tool set designed to develop aeronautics, automotive and safety critical embedded control-systems. In: Convergence 2003 (2003)

[HH94] Henzinger, T.A., Ho, P.-H.: Hytech: The cornell hybrid technology tool. In: Hybrid Systems, pp. 265–293 (1994)

[RGA$^+$96] Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S.-T., Edwards, S., Khatri, S., Kukimoto, Y., Pardo, A., Qadeer, S., Ranjan, R.K., Sarwary, S., Shiple, T.R., Swamy, G., Villa, T.: VIS: a system for verification and synthesis. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, Springer, Heidelberg (1996)

[SB00] Somenzi, F., Bloem, R.: Efficient Büchi Automata from LTL Formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)

[SFHK04] Stursberg, O., Fehnker, A., Han, Z., Krogh, B.H.: Verification of a cruise control system using counterexample-guided search. In: Control Engineering Practice, Elsevier, Amsterdam (2004)

[SK00] Silva, B.I., Krogh, B.H.: Formal verification of hybrid systems using checkmate: a case study. In: Proceedings of the American Control Conference, pp. 1679–1683 (2000)