Scalable Static Hybridization Methods for Analysis of Nonlinear Systems*

Stanley Bak
Air Force Research Laboratory
Information Directorate, USA

Sergiy Bogomolov
IST Austria

Thomas A. Henzinger
IST Austria

Taylor T. Johnson
University of Texas at
Arlington, USA

Pradyot Prakash
IIT Bombay, India

ABSTRACT

Hybridization methods enable the analysis of hybrid automata with complex, nonlinear dynamics through a sound abstraction process. Complex dynamics are converted to simpler ones with added noise, and then analysis is done using a reachability method for the simpler dynamics. Several such recent approaches advocate that only "dynamic" hybridization techniques—i.e., those where the dynamics are abstracted on-the-fly during a reachability computation—are effective. In this paper, we demonstrate this is not the case, and create static hybridization methods that are more scalable than earlier approaches.

The main insight in our approach is that quick, numeric simulations can be used to guide the process, eliminating the need for an exponential number of hybridization domains. Transitions between domains are generally time-triggered, avoiding accumulated error from geometric intersections. We enhance our static technique by combining time-triggered transitions with occasional space-triggered transitions, and demonstrate the benefits of the combined approach in what we call mixed-triggered hybridization. Finally, error modes are inserted to confirm that the reachable states stay within the hybridized regions.

The developed techniques can scale to higher dimensions than previous static approaches, while enabling the parallelization of the main performance bottleneck for many dynamic hybridization approaches: the nonlinear optimization required for sound dynamics abstraction. We implement our method as a model transformation pass in the HYST tool, and perform reachability analysis and evaluation using an unmodified version of SpaceEx on nonlinear models with up to six dimensions.

1. INTRODUCTION

A hybrid automaton [7] is an expressive mathematical model useful for describing complex dynamic processes involving both continuous and discrete states and their evolution. Efficient algorithms and analysis tools for linear and affine systems have recently emerged [24]. However, the behaviour of many real-world systems can only be modeled with nonlinear differential equations.

Hybridization methods attempt to address this issue, enabling the application of existing algorithms for simpler dynamics (such as constant or affine dynamics) on the analysis of hybrid automata with nonlinear differential equations. Alternative recent approaches for analyzing nonlinear systems include simulation-based verification [22] or using efficient representations such as Taylor models [17]. Most hybridization methods work by dividing the state space into a set of domains. In each domain, the nonlinear dynamics are then converted to simpler ones with added noise to account for the abstraction error within the domain. Hybridization is also known as *conservative approximation* [8], which illustrates that it is a sound (or conservative) abstraction. Hybridization has been used to verify properties for several types of systems, from analog/mixed-signal circuits [19] to autonomous satellite maneuvers in space [14, 31].

We classify existing hybridization approaches along two axes as shown in Table 1: static versus dynamic, and space-triggered versus time-triggered. *Static* hybridization approaches use a fixed partitioning, and can make use unmodified, off-the-shelf analysis tools. In contrast, *dynamic* methods exploit runtime information to perform hybridization, and therefore must be tightly integrated within an analysis tool. On the other axis, *space-triggered* techniques perform geometric intersections along hybridization domain boundaries. Time-triggered hybridization, on the other hand, avoids this operation by creating a series of overlapping domains, and switches between them at specific points in time.

Based on this classification, a gap exists in existing research: no methods exist that perform static, time-triggered hybridization. The main contribution of this paper is the investigation of this category, and demonstrating that such methods can overcome some of the drawbacks of existing hybridization methods. Notably, the new hybridization methods are more scalable than existing space-triggered approaches. Furthermore, the expensive dynamics abstraction step, which is generally a global optimization problem, is easily parallelizable, which is not the case in dynamic approaches. We further enhance our static technique by combining time-triggered transitions with occasional space-triggered tran-

*DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA #88ABW-2016-0181, 28 JAN 2016)

HSCC'16, April 12 - 14, 2016, Vienna, Austria

ACM ISBN 978-1-4503-3955-1/16/04…$15.00

DOI: http://dx.doi.org/10.1145/2883817.2883837

	Space-Triggered	Time-Triggered	Mixed-Triggered
Static	[8, 10, 29, 31]	this paper	this paper
Dynamic	[8, 9]	[1–3, 5, 20, 28]	none

Table 1: Breakdown of hybridization approaches into static versus dynamic, and space-triggered versus time-triggered, as well as combinations thereof (mixed-triggered).

sitions, and demonstrate the benefits of the combined approach in what we call *mixed-triggered hybridization*.

The static mixed-triggered hybridization approach works by hybridizing only a part of the state space. We use quick numeric simulations to guide the partitioning process. In this way, we mitigate the problem of exponential growth in the number of partitions. In addition, we generally use time-triggered guards in the transitions between partitions. This prevents costly geometric intersection computations which typically add overapproximation error to the result. We ensure the soundness of the constructed abstraction by adding *error modes* to guarantee that the computed reachable states remain within the hybridized region (which is constructed from simulations that may be imprecise).

We implement the hybridization method described in this paper as a model transformation pass in the HYST source-to-source translation tool. Since it is a static approach, we can use unmodified reachability tools on the hybridized models. We create affine abstractions of nonlinear dynamics, and use to perform reachability analysis.

Contributions and Paper Organization. The main contribution of this paper is the development of the first static time-triggered and mixed-triggered hybridization methods. Of critical importance in the proposed approaches is the choice of hybridization parameters, and a second contribution is an algorithm which uses simulations to generate these values. This algorithm is implemented in the HYST [12] model transformation tool, which allows it to quickly be applied to new systems and with new simulation parameters. Finally, we validate our claims that the method is more scalable than existing static approaches by evaluating it on nonlinear models, including a six-dimensional water tank model, and then using an unmodified version of SpaceEx [13, 15, 24], which does not natively support nonlinear dynamics, to compute the set of reachable states.

The remainder of the paper is organized as follows. First, Section 2 reviews and classifies existing hybridization methods. Section 3 then presents mathematical background and formalisms, which are used in Section 4 to give formal descriptions and correctness arguments for several hybrid automaton transformations. The transformations are combined into our hybridization approaches in Section 5, which also proposes a simulation-based algorithm to create the hybridization parameters. Section 6 discusses the implementation in HYST and experimental reachability results in SpaceEx, followed by a conclusion in Section 7.

2. HYBRIDIZATION METHODS

In this section, we discuss and classify previous research on hybridization. Hybridiziation is the process of using simple dynamics with noise to create an abstraction of a system with more complicated, usually nonlinear, dynamics. This is done to enable the analysis of systems with the more complicated dynamics by methods which work exclusively on the simpler ones.

This process is typically targeted for flow-pipe construction methods, where the set of reachable states is iteratively computed or overapproximated at monotonically increasing instances in time, starting from an initial set of states. Computational approaches maintain some representation of the set of states at each time instances, which we informally refer to as the *currently-tracked* set of states.

Static Space-Triggered Hybridization. Early hybridization methods were both static and space-triggered [29]. In these approaches, the state space is partitioned using a (typically uniform) grid or mesh, and transitions are added along the partition boundaries, resulting in state-dependent switching. The advantage of this approach is that existing termination checking techniques can be used, which is particularly useful in the case of periodic systems where linearizing a bounded subset of the state-space is reasonable [31].

There are, however, three main drawbacks. First, static mesh construction is traditionally done without knowledge of the reachable states. Therefore, it requires computing the mesh over the entire state space (or bounded subset thereof), which scales exponentially with the number of continuous dimensions in the system. Second, the geometric intersections required by space-triggered approaches may introduce error during reachability computation [4, 17]. This is because such intersections can require tools to convert from precise internal representations such as zonotopes [25], support functions [27], or Taylor models [17], to simpler representations where intersection operations can be computed, such as polytopes [6]. After intersection, the simpler representation is then converted back to the internal representation for subsequent computation [26]. These conversions can result in overapproximations of the original currently-tracked set of states, adding error each time they are performed. Since hybridization can be done more accurately when domains are small, many intersection operations may be necessary and this can quickly lead to error explosion, as well as an explosion in the number of modes of the hybrid automaton. Third, the currently-tracked set of reachable states may leave a hybridization domain along multiple facets, requiring splitting and, later, possibly remerging the set of reachable states, which can be both computationally expensive and inaccurate [20].

Dynamic Space-Triggered Hybridization. In order to help increase scalability, methods were developed that closely integrate hybridization with reachability analysis [8]. This results in dynamic methods where the domain construction and the abstraction process is performed on-the-fly and only on states that are reachable [9]. Although dynamic space-triggered methods scale better into higher dimensions, they still suffer from the other two problems mentioned above: error accumulation due to many geometric intersections, and the splitting of the currently-tracked set of states along multiple facets.

Dynamic Time-Triggered Hybridization. To address the other two drawbacks, dynamic time-triggered approaches were developed [5, 20, 28]. These methods avoid geometric intersections by choosing hybridization domains around the currently-tracked set of states. As time is advanced, the hy-

bridization domains are updated to be near the new position of the currently-tracked set of states, without requiring an intersection operation. This can be done at each step [28], or whenever the currently-tracked set of states leaves the hybridization domain [20]. This can be viewed as the mode of the abstract hybrid automaton changing at specific instances in time to a mode with new dynamics, which corresponds to a time-triggered transition.

Although dynamic time-triggered methods perform well, they also suffer from certain drawbacks. The most important drawback is that, in the earlier static approaches, performing the dynamics abstraction step was an embarrassingly parallel problem, so parallelism could be leveraged to reduce total runtime (or equivalently, increase precision for a fixed runtime). In dynamic methods, the bounds of each new abstraction domain depend on the set of reachable states in the previous domain, forcing this expensive step to be performed serially. For example, abstracting nonlinear dynamics using polynomial differential inclusions can yield an accurate hybridization, but it requires bounding the Lagrange remainder of the dynamics' Taylor expansion [1]. In previous work, this step was reported to take 1121 out of 1180 seconds on a nine-dimensional biological aging model (about 95% of the runtime), and 1155 out of 1296 seconds on hybrid variant of the same model (about 89%), although it was mentioned that some implementation optimizations were possible [1]. Some parallelization of reachability computation was considered to enable online reachability of car manoeuvres [2,3]. However, the crucial step of dynamics abstraction (computing the linearization errors) was still performed serially because the overapproximation of the Lagrange remainders of the Taylor expansions of the dynamics at each step was based on the Lagrange remainders at the previous step. This serial step dominated the reported runtime of the technique.

A second drawback of time-triggered approaches is that, if the currently-tracked set of states becomes large (which can be a property of the system regardless of the method used), the domains over which dynamics abstraction is performed also become large. This, in turn, increases the dynamics approximation error that must be added to the simpler dynamics to result in a sound abstraction, increasing error in the overapproximation of the set of reachable states. This can be overcome by splitting the set of reachable states [21], although this may yield an exponential number of sets that need to be tracked, and possibly redundant computation. This problem can be partially mitigated through extra tracking to perform cancellation of redundant sets of reachable states, which requires (expensive and error-introducing) intersection operations on the internal representations [5]. Space-triggered approaches do not suffer from this problem. In fact, introducing *occasional* artificial space-triggered transitions can serve to reduce the size and complexity of the currently-tracked set of reachable states [11].

Novel Hybridization Approaches. A classification of existing hybridization research is shown in Table 1. A research gap is noticeable in the static time-triggered category. This paper attempts to fill this gap by developing, to the best of the authors' knowledge, the first static time-triggered hybridization method. The approach is static, and therefore can perform the bottleneck step of dynamics abstraction in a parallel fashion. Since the approach is time-triggered, it can scale to larger numbers of dimensions while avoiding the accumulation of intersection error. Additionally, as the method is static and modifies the model directly, it can work with unmodified reachability tools, yielding immediate benefit of its application using the latest reachability methods.

There are also no fundamental reasons why a method could not use both time-triggered and space-triggered transitions during analysis. We develop such a *mixed-triggered* hybridization approach, which generally uses time-triggered transitions, but occasionally performs a state-triggered transition to attempt to reduce the size and complexity of the currently-tracked set of states. In our review of existing research, no such approaches currently exist.

Other Hybridization Factors. Research in hybridization also explores other aspects that are important, but less critical to the methods developed in this paper. One choice when performing hybridization is the shape of space-triggered domains. Rectangular domains are simple to reason about, although manual region selection [29], simplexes [9, 21, 31], and nonuniform meshes [8,10,31] have been considered. The sound and tight abstraction of dynamics within each domain is critical to control error when performing hybridization. The main reason to consider alternative domains is in order to reduce this error. For general nonlinear dynamics, this often requires solving constrained nonlinear optimization problems, which can be impossible in theory and expensive in practice. For rectangular domains, interval analysis [30] can be used to provide guaranteed bounds for this problem. For other types of domains, the success of the method depends on the system being analyzed. For example, to perform the nonlinear optimization step for simplicial domains, one can use knowledge of the system's Lipschitz constant (which will be sound but inaccurate), or compute bounds on the second partial derivatives (the elements of the Hessian matrix) [8, 9, 21]. In general, this is a nonlinear optimization problem with linear constraints, but for specific cases it can be efficiently solved. For example, for quadratic dynamics [20, 21], the Hessian matrix is constant. The choice of domains is not critical to the methods being developed in this paper, so for simplicity, we considered rectangular domains.

A second choice when performing hybridization is the type of 'simpler' dynamics. Choices range from constant bounds [16, 29, 31, 32], linear and affine bounds [9, 21, 31], to polynomial bounds [1, 18]. In this paper, we target an unmodified implementation of the SpaceEx tool [24], and therefore simplify from nonlinear dynamics to affine dynamics.

3. PRELIMINARIES

We now define in more formal notation the syntax and semantics of hybrid automata. This will be used to precisely specific and justify the soundness of the model transformation steps used in the developed hybridization approach in the next section.

Definition 1. A *hybrid automaton* $\mathcal{H}$ is defined by a tuple $\mathcal{H} \triangleq (Modes, Var, Init, Flow, Trans, Inv)$, where: (a) $Modes$ is a finite set of modes. (b) $Var = \{x_1, \ldots, x_n\}$ is a set of real-valued variables. (c) $Init(m) \subseteq \mathbb{R}^n$ is the set of initial values for $x_1, \ldots, x_n$ for each mode $m \in Modes$. (d) For each $m \in Modes$, the flow relation $Flow(m)$ is a relation

over the variables in x and their derivatives $\dot{x} = f_m(x)$, where $x(t) \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$, i.e., differential inclusions are allowed. (e) *Trans* is a set of discrete transitions $t = (m, g, v, m')$, where m and m' are the source and the target modes, g is the guard of t, and v is the update of t. (f) $Inv(m) \subseteq \mathbb{R}^n$ is an invariant for each mode $m \in Modes$.

For a time interval T, we define a *trajectory* of $\mathcal{H}$ from state $s = (m, \mathbf{x})$ to state $s' = (m', \mathbf{x}')$ as a tuple $(L, \mathbf{X})$. In this tuple, the function $L : T \to Modes$ and $\mathbf{X} : T \to \mathbb{R}^n$ are functions that define for each time point in T the mode and values of the continuous variables, respectively.

A state s' is *reachable from a state* s if there exists a trajectory starting with s and ending with s'. A state s' is *reachable* if s' is reachable from a state s where s is an initial state. We denote the set of states reachable from the set X in mode m by $\text{Reach}_{\mathcal{H}}(m, X)$. $\text{Reach}(\mathcal{H})$ of $\mathcal{H}$ is defined as the set of states that are reachable from the set of initial states. We use $\text{Reach}^c_{\mathcal{H}}(m, X)$ and $\text{Reach}^c(\mathcal{H})$ to denote the versions of the these operators that return only the *continuous* part of the computed state space. We refer to $\text{Reach}^c(\mathcal{H})$ as the *continuous reachable state space* of $\mathcal{H}$. We denote the projection of the set $R \subseteq \mathbb{R}^n$ over variables Var to the subset $Var' \subseteq Var$ by $R \downarrow_{Var'}$. Throughout the paper, we always refer to time-bounded reachability, i.e., we consider trajectories which evolve up to the time horizon T_{max}. In order to simplify notations, we implicitly take this assumption for granted in our reasoning. Finally, given a mode m of the automaton $\mathcal{H}$, we refer to the set of *outgoing* transitions as $Trans_{\mathcal{H}}(m)$.

4. HYBRIDIZATION TRANSFORMATIONS

We are interested in methods to compute an overapproximation of the time-bounded set of reachable states, which produce tight overapproximations, yet are feasible from the computational point of view. The proposed approaches rely on several hybrid automaton transformations. A source-to-source *transformation* takes as input a hybrid automaton $\mathcal{H}$, a mode $m \in Modes$,[1] possibly some additional parameters, and returns as output another hybrid automaton $\theta(\mathcal{H})$. The four described transformations are (1) *time-triggered splitting*, (2) *space-triggered splitting*, (3) *domain contraction*, and (4) *dynamics abstraction*. In time-triggered splitting, a given mode of $\mathcal{H}$ is split into possibly multiple modes via a time-triggered splitting of the modes. Similarly, in space-triggered splitting, a mode is split by augmenting the mode invariant with a constraint induced by a *space trigger function*. Domain contraction adds auxiliary invariants called *contraction domains* to a mode by intersecting them with the existing invariants of the mode. Dynamics abstraction overapproximates the dynamics in a mode of the automaton, which in this paper, abstracts nonlinear differential equations by linear differential inclusions, in particular a linear differential equation with an additive set-valued (interval vector) input.

As hybridization of the *continuous dynamics* of hybrid automata is the most challenging part of the hybridization process, we focus on the continuous dynamics of hybrid systems in the rest of the paper and assume that an input hybrid automaton has only *one* mode. Our approach *over-approximates* the behavior of the original system by a hybrid automata consisting of *multiple* modes. Therefore, only reachable *continuous states* are relevant for the soundness of the transformations. This fact allows us to to conclude that the inclusion of the original *continuous* reachable state space into the transformed one is enough to show soundness of our transformations. Note, however, that although the input hybrid automaton for the whole hybridization approach is assumed to be a singleton, our transformations are defined in terms of general hybrid automata.

In this section, each of these four transformations is precisely defined. In the next section, the simple transformations will be combined in order to perform static time-triggered and mixed-triggered hybridization.

4.1 Time-Triggered Splitting

The time-triggered splitting transformation, informally, separates the handling of system behavior in the first τ time units, and the rest of the trajectory up to the time horizon. In order to achieve this goal, the transformation splits a given mode of a hybrid automaton into two and imposes constraints that guarantee that the system dwells in the first mode for τ time units and proceeds to the second one once the time threshold has been reached.

Definition 2. A *time-triggered splitting* is a transformation θ_{tt} of a hybrid automaton $\mathcal{H}$, that takes as input an automaton $\mathcal{H}$, a mode $m \in Modes$ that has no outgoing transitions[2], and a real positive time τ, a *time-trigger threshold*. The hybrid automaton $\mathcal{H}_{tt} \triangleq \theta_{tt}(\mathcal{H})$ is defined as: (a) $Modes_{\mathcal{H}_{tt}} \triangleq Modes_{\mathcal{H}} \cup \{m_{tt}\}$, where m_{tt} is a fresh (i.e., unique) mode name, (b) $Var_{\mathcal{H}_{tt}} \triangleq Var_{\mathcal{H}} \cup \{t\}$, where t is known as the *time-trigger* variable and is fresh, i.e., assume without loss of generality that t is a unique variable name,[3] (c) the initial states are copied; in addition, if $Init_{\mathcal{H}}(m)$ is not the empty set (i.e., m is an initial mode), then $Init_{\mathcal{H}_{tt}}(m) \triangleq Init_{\mathcal{H}}(m) \wedge t = \tau$, and otherwise $Init_{\mathcal{H}_{tt}}(m) \triangleq Init_{\mathcal{H}}(m)$; $Init_{\mathcal{H}_{tt}}(m_{tt}) \triangleq \emptyset$, (d) the flows are copied, and $Flow_{\mathcal{H}_{tt}}(m_{tt}) \triangleq Flow_{\mathcal{H}}(m)$, so mode m_{tt} copies the original dynamics of m, and in m, $\dot{t} = -1$, and in all modes other than m, $\dot{t} = 0$, (e) the transitions are copied; in addition, $Trans_{\mathcal{H}_{tt}}(m_{tt}) \triangleq Trans_{\mathcal{H}}(m)$, with an additional transition created from m to m_{tt} with the guard $t = 0$; moreover, every incoming transition to m has the reset $t := \tau$ added, (f) the invariants are copied; in addition $t \geq 0$ is added to $Inv_{\mathcal{H}_{tt}}(m)$ and $Inv_{\mathcal{H}_{tt}}(m_{tt}) \triangleq Inv_{\mathcal{H}}(m)$ (m_{tt} copied the original invariant of m).

Figure 1 illustrates the time-triggered splitting for a single mode. A *time-triggered transition* corresponds to any transition with guard $t = 0$ taken when the time-trigger variable

[1]For simplicity of presentation, each transformation is defined for a given mode of the hybrid automaton $\mathcal{H}$, and their application to multiple modes of $\mathcal{H}$ is straightforward by iterating over each element of *Modes*.

[2]In order to make the presentation of our transformation clearer, we consider a mode with no outgoing transitions. Our construction can be easily generalized to also accommodate this feature.

[3]If the time-triggered splitting transformation θ_{tt} is applied to an automaton multiple times, the time-trigger variable may be reused in each splitting, as it needs only to be fresh on the first application of the transformation and is what is done in our implementation. However, for simplicity of presentation, we presume the time-trigger variable is always fresh to allow θ_{tt} to be applied multiple times.

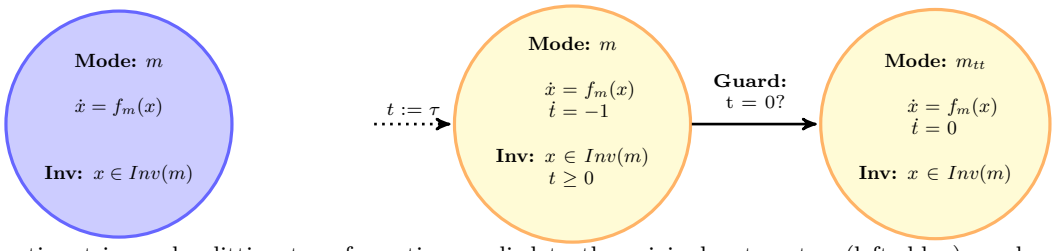


Figure 1: The time-triggered splitting transformation applied to the original automaton (left, blue) produces the output automaton (right, yellow). An additional time-trigger variable t is added that counts down to zero from an initial time τ.

$t = 0$. In contrast to general guards, the reachability along time-triggered transitions can be computed computationally efficient as many reachability algorithms automatically capture *time dependencies* as part of their workflow. For example, the STC scenario [23] of the hybrid model checker SpaceEx computes *time-dependent* piecewise-linear approximations of the support functions evolution.

The following lemma connects the time-triggered splitting transformation with the original hybrid automaton.

LEMMA 4.1. *Let $\mathcal{H}$ be a hybrid automaton with a set of continuous variables Var, $m \in Modes$ be a mode without outgoing transitions, and $\tau \in \mathbb{R}_{>0}$ be a time-trigger threshold. Then it holds that* $\text{Reach}^c(\mathcal{H}) \subseteq \text{Reach}^c(\theta_{tt}(\mathcal{H}))\restriction_{Var}$.

Here, we note that we need to project away the auxiliary variable t in order to ensure that the sets of reachable states of $\mathcal{H}$ and $\theta_{tt}(\mathcal{H})$ can be compared.

4.2 Space-Triggered Splitting

Space-triggered splitting is a further transformation, which works similarly to the time-triggered splitting by splitting a given mode into several ones. However, in contrast to the time-triggered transformation, it uses a *space-trigger function* to define criteria for mode splitting.

Definition 3. A *space-triggered splitting* is a transformation θ_{st} of a hybrid automaton $\mathcal{H}$, that takes as input an automaton $\mathcal{H}$, a mode $m \in Modes$ that has no outgoing transitions, and a function $\pi : \mathbb{R}^n \to \mathbb{R}$ called the *space-trigger* function. The function π must satisfy the condition that upon entering mode m, $\pi(x) \geq 0$, where x is the current state. This means that if m is an initial mode, for all states $x \in Init(m)$, $\pi(x) \geq 0$. The hybrid automaton $\mathcal{H}_{st} \triangleq \theta_{st}(\mathcal{H})$ defined as: (a) $Modes_{\mathcal{H}_{st}} \triangleq Modes_{\mathcal{H}} \cup \{m_{st}\}$, where m_{st} is a fresh (i.e., unique) mode name, (b) $Var_{\mathcal{H}_{st}} \triangleq Var_{\mathcal{H}}$, (c) the initial states are copied; $Init_{\mathcal{H}_{st}}(m_{st}) \triangleq \emptyset$, (d) the flows are copied; in addition, $Flow_{\mathcal{H}_{st}}(m_{st}) \triangleq Flow_{\mathcal{H}}(m)$, (e) the transitions are copied; in addition, $Trans_{\mathcal{H}_{st}}(m_{st}) \triangleq Trans_{\mathcal{H}}(m)$; moreover, an additional transition created from m to m_{st} with the guard $\pi(x) = 0$, and (f) the invariants are copied, with $\pi(x) \geq 0$ added to $Inv_{\mathcal{H}_{st}}(m)$ and $Inv_{\mathcal{H}_{st}}(m_{st}) \triangleq Inv_{\mathcal{H}}(m)$ (m_{st} copied the original invariant of m).

The space-triggered splitting transformation adapts the idea of pseudo-invariants [11] to the hybridization setting. In our setting, a space-trigger function π basically plays a role of a pseudo-invariant.

The resulting automaton overapproximates the continuous reachable state space of the original one which is formally stated in the following lemma.

LEMMA 4.2. *Let $\mathcal{H}$ be a hybrid automaton, $m \in Modes$ be a mode without outgoing transitions, and $\pi : \mathbb{R}^n \to \mathbb{R}$ be a function satisfying the assumptions in Definition 3. Then* $\text{Reach}^c(\mathcal{H}) \subseteq \text{Reach}^c(\theta_{st}(\mathcal{H}))$.

4.3 Domain Contraction

Domain contraction adds auxiliary invariants known as *contraction domains* that should contain the set of reachable states. Given a set D and a mode m of a hybrid automaton $\mathcal{H}$ where $\dot{x} = f_m(x)$, if $\text{Reach}_{\mathcal{H}}(m, X) \subseteq D$ for $X \subseteq Inv(m)$, i.e. the set of reachable states from mode m starting from a subset $X \subseteq Inv(m)$ is contained in D, then D may safely be added as an invariant of m. Of course, the set of reachable states is not available and is what is being computed or approximated, so error modes known as *domain contraction error modes* (DCEMs) are used to maintain soundness if the system leaves the states represented by these auxiliary invariants.

Definition 4. A *domain contraction* is a transformation θ_{dc} of a hybrid automaton $\mathcal{H}$, that takes as input an automaton $\mathcal{H}$, a mode $m \in Modes$, and a set $D \subseteq \mathbb{R}^n$ called the *contraction domain* auxiliary invariant.

The transformed hybrid automaton $\mathcal{H}_{dc} \triangleq \theta_{dc}(\mathcal{H})$ is defined as: (a) $Modes_{\mathcal{H}_{dc}} \triangleq Modes_{\mathcal{H}} \cup \{err\}$, the modes are the copied, with a new *domain contraction error mode* (DCEM) err added, (b) $Var_{\mathcal{H}_{dc}} \triangleq Var_{\mathcal{H}}$, (c) the initial states are copied; additionally, if m is an initial mode, and $Init(m)$ is not entirely contained in D, then add the err DCEM to the initial states; in this way, we capture a degenerate case if the initial set has states outside of the contraction domain. (d) the flows are copied; additionally, $Flow_{\mathcal{H}_{dc}}(err)$ of the form $\dot{x} = 0$ are added, (e) the transitions are copied, with additional transformations of the following form: given an incoming transition $d = (n, g, v, m)$ to mode m in $\mathcal{H}$, (1) augment the guard of the transition d with $x \in D$, and (2) add an additional transition $d' = (n, g \wedge x \in cl(\bar{D}), err)$ with an extra condition $x \in cl(\bar{D})$ on the guard and leading to the DCEM err, where $\bar{D}$ denotes the complement of D and $cl(\cdot)$ stands for topological closure and (3) add an additional transition $d'' = (m, x \in cl(\bar{D}), err)$, (f) the invariants are copied, except for the invariant $Inv_{\mathcal{H}_{dc}}(m) \triangleq Inv_{\mathcal{H}}(m) \cap x \in D$.

A visualization of the domain contraction transformation is given in Figure 2.

The conditions to enter a DCEM together ensure that regardless of the choice of the contraction domain, if the DCEM err is not reached, then the overapproximation of the reachable states is sound. Additionally, the condition that the dynamics are zero in the DCEM err ensures that during

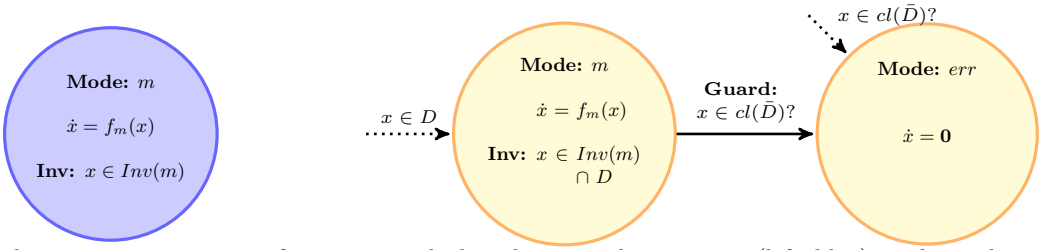


Figure 2: The domain contraction transformation applied to the original automaton (left, blue) produces the output automation (right, yellow). The contraction domain D is added to the invariant, with DCEM *err* inserted to detect if the reachable set of states leaves D.

a reachability computation, the exploration of the *err* will terminate and be a dead-end in the exploration of the state-space. Note that the notion of topological closure is required to ensure that the intersection of guard and invariant is non-empty.

LEMMA 4.3. *Let $\mathcal{H}$ be a hybrid automaton, $m \in Modes$ be a mode, and $D \subseteq \mathbb{R}^n$ be a contraction domain. Then, if no DCEM is reachable,* $\text{Reach}^c(\mathcal{H}) \subseteq \text{Reach}^c(\theta_{dc}(\mathcal{H}))$.

The contraction domain auxiliary invariants may be arbitrary and may be determined using any method, so they may not actually contain the set of reachable states. To maintain soundness, the DCEMs are added such that if the contraction domains *do not* contain the set of reachable states, transitions to the DCEMs *may* be taken.[4] If no DCEMs are reached, then the domain contraction transformation is sound, but otherwise, if a DCEM is reached, the resultant set of set of reachable states may not be subset of the original automaton's set of reachable states. If it is known that the set of reachable states will not leave the contraction domain by some other analysis, then the DCEMs are not necessary and the invariants may simply be augmented (conjuncted) with the contraction domain. In summary, if the contraction domains do not contain the set of reachable states for a given mode, then a state with a mode equal to the DCEM will be reached.

4.4 Dynamics Abstraction

Continuous dynamics are abstracted by transforming the flows of the original hybrid automaton into flows with increased nondeterminism. In this paper, nonlinear differential inclusions are overapproximated using linear differential inclusions, specifically linear ODEs with an additive set-valued input.

Definition 5. A *dynamics abstraction* is a transformation θ_{da} of a hybrid automaton $\mathcal{H}$, that takes as input an automaton $\mathcal{H}$, a mode $m \in Modes$, and a set-valued function $g : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ called the *abstract dynamics*, where, for the flow $\dot{x} = f_m(x)$ of mode m with invariant $Inv(m)$, $g_m(x)$ is such that $\forall x \in Inv(m)$: $f_m(x) \subseteq g_m(x)$. The hybrid automaton $\mathcal{H}_{da} \triangleq \theta_{da}(\mathcal{H})$ is defined as: (a) $Modes_{\mathcal{H}_{da}} \triangleq Modes_{\mathcal{H}}$, (b) $Var_{\mathcal{H}_{da}} \triangleq Var_{\mathcal{H}}$, (c) the initial states are copied, (d) the flows are copied, except for $Flow_{\mathcal{H}_{da}}(m)$ which is set to $\dot{x} = g(x)$, (e) the transitions are copied, (f) the invariants are copied.

[4]Since the semantics of hybrid automata defined do not support urgency or *must* transitions, we exploit the fact that the reachability computation explores all paths to ensure soundness.

Similarly to other transformations we have considered, we formulate a lemma relating the original and transformed systems.

LEMMA 4.4. *Let $\mathcal{H}$ be a hybrid automaton, $m \in Modes$ be a mode, $g : \mathbb{R}^n \to 2^{\mathbb{R}}$ be a set-valued function satisfying the assumptions in Definition 5. Then it holds that* $\text{Reach}^c(\mathcal{H}) \subseteq \text{Reach}^c(\theta_{da}(\mathcal{H}))$.

5. MIXED-TRIGGERED HYBRIDIZATION

Now we present the central result of the paper, a static mixed-triggered hybridization that combines the four transformations we have introduced.

Definition 6. A static *mixed-triggered hybridization* is a transformation θ_{mt} of a hybrid automaton $\mathcal{H}$ and has the following input:

- a single-mode automaton $\mathcal{H}$,
- a list of splitting elements $E_1 \ldots E_{n-1}$, where each element E_i is either a real number to be used for time-triggered splitting, or a π function to be used for space-triggered splitting (list 1),
- $D_1, \ldots, D_n$ are the contraction domains (sets) for each new location (list 2), and
- $g_1, \ldots, g_n$ are the dynamics abstraction functions for each location (list 3).

The mixed-triggered hybridization transformation consists of the following three steps:

- Apply either time-triggered splitting or space-triggered splitting based on the list $E_1 \ldots E_{n-1}$. We apply each transformation to the most-recently constructed mode, which has no outgoing transitions. The result of this step is a chain of modes.
- For each mode in the chain, apply N domain contractions based on the list $D_1, \ldots, D_n$.
- For each mode in the chain, apply N dynamics abstractions based on the list $g_1, \ldots, g_n$.

If the list of splitting elements (list 1) contains only time-triggered splitting elements (and no space-triggered splitting elements), then it is a *static time-triggered hybridization.*

The following theorem establishes the soundness of the mixed-triggered hybridization.

THEOREM 5.1. *Given a hybrid automaton $\mathcal{H}$, if no DCEM is reachable, then the continuous reachable state space of the mixed-triggered transformation $\theta_{mt}(\mathcal{H})$ overapproximates the continuous reachable state space of the original automaton:* $\text{Reach}^c(\mathcal{H}) \subseteq \text{Reach}^c(\theta_{mt}(\mathcal{H}))$.

PROOF. The proof follows by a straight-forward application of Lemmas 4.1, 4.2, 4.3, and 4.4. □

We observe that the mixed-triggered hybridization approach contains a number of parameters which must be carefully chosen in order to guarantee a sound abstraction, which is ensured when no error modes (DCEMs) are reachable. If the contraction domains are too small, then the set of reachable states will exit the domain and the DCEM will be reached. If the contraction domains are too large, then the dynamics abstraction will be a large overapproximation, and the set of reachable states will become both large and inaccurate. In modes copied during time-triggered splitting, whenever the time-triggered variable t reaches zero, the set of reachable states at each mode must be contained in the domains (invariants) of both the source and target locations. Space-triggered splitting requires as input the π functions which determine the splitting structure.

In the following, we describe an approach to generate the parameters for proposed hybridization approach in a way that will satisfy the above requirements. Again, the approach is described assuming a single-location hybrid automaton, where the initial set of states is a rectangle, although generalizations are not difficult.

5.1 Parameter Selection Algorithm

In order to construct the three lists to be used as hybridization parameters, an algorithm is proposed which uses numerical simulations. The proposed approach has its own user-provided parameters:

- T is the maximum time,
- $\mathcal{S}$ a simulation strategy, one of {POINT, STAR, STARCORNERS}
- δ_{tt} is the simulation time in a time-triggered transformation step,
- n_{pi} is the number of space-triggered transformation steps to use,
- δ_{pi} is the maximum simulation time in a space-triggered transformation step,
- ϵ is a bloating term to account for the difference between the simulated points the set of reachable states.

The algorithm first selects a finite set of simulation points sampled from the initial set of states. If $\mathcal{S}$ is POINT, only the center of the initial rectangle is used. If $\mathcal{S}$ is STAR, the center is used, as well as the center of every face of the rectangle, $1 + 2n$ points, where n is the number of variables. If $\mathcal{S}$ is STARCORNERS, the center is used, as well as the centers of every face, as well as the corners of the initial rectangle, $1 + 2n + 2^n$ points. Selecting more points may permit a smaller ϵ, but since the number of points is exponential, the STARCORNERS strategy may not always be practical. The collection of points are stored in a variable, `sims`.

The algorithm proceeds in iterations, at each iteration doing either a *space-triggered step*, or a *time-triggered step*. The three parameter lists (the output) are initially empty. A current time variable `ct`, initially zero, is maintained which tracks the amount of time elapsed during time-triggered steps (space-triggered steps do not add to `ct`). A second variable `next_st` tracks the time at which to insert the next space-triggered value. If $n_{pi} > 0$, `next_st` is initialized to 0, otherwise it is set to ∞.

At each iteration, if the current time `ct` variable is greater than or equal to next space-triggered time variable `next_st`, a space-triggered step is *attempted* and `next_st` is increased by $\frac{T}{n_{pi}}$. Otherwise, a time-triggered transition is performed and `ct` is increased by δ_{tt}. The process completes when `ct` exceeds the maximum time T.

A **time-triggered step** adds δ_{tt} to output list 1. Then, it computes the bounding box of `sims`, bloats it by ϵ, and stores it in `start`. Each point in `sims` is numerically simulated for δ_{tt} time. The bounding box of `sims` is computed again, bloated by ϵ, and stored in `end`. The bounding box of `start` and `end` is then computed, and put into output list 2 (contraction domains).

A visualization of two consecutive time-triggered steps is shown in Figure 3. Here, $\mathcal{S}$ = POINT, so `sims` is just a single point. Initially, `sims` is α. After δ_{tt} time, the point β is reached; after δ_{tt} further time, the simulation reaches γ. The modification of the output lists after these two steps would be the time-triggered value δ_{tt} twice inserted into list 1, the red rectangle set inserted into list 2, followed by the green rectangle set inserted into list 2.

A **space-triggered step** attempts to use numerical simulations to find a function π for space-triggered splitting, but may, in certain cases, be aborted without modifying the output lists. First, the bounding box of `sims` is computed, bloated by ϵ, and stored in `start`. The center point in `sims`, which we call `p`, is numerically simulated until either, (1) the plane induced by the point lies entirely on one side of `start`, or (2) the space-triggered time limit δ_{pi} is reached. If condition (2) occurs, the space-triggered step returns without modifying the output lists, and reverts the status of `sims`. For condition (1), the plane induced by a point `p` is a hyperplane that both contains `p` and is orthogonal to the gradient at `p`. The function π is created from the equation of the hyperplane, where π is zero along the plane and positive on the side of `start` (in the opposite direction of the gradient at `p`). Forcing transitions along hyperplanes orthogonal to the gradient was previously shown as effective in reducing the size of the currently-tracked set of reachable states in the context of pseudo-invariants [11, 12]. Each of the other points in `sims` are then numerically simulated until either (1) they reach a point along the constructed hyperplane where π evaluates to zero, or (2) they are simulated for the space-triggered time limit δ_{pi}. If for any point condition (2) occurs, again, the space-triggered step aborts without modifying the output lists, and reverts the status of `sims`. If condition (1) occurs for every point in `sims`, the bounding box of all the points in `sims` (which are all along the hyperplane) is taken, bloated by ϵ, and assigned to `end`. The bounding box of `start` and `end` is then computed, and put into output list 2 (contraction domains). The hyperplane function π is put into output list 1.

At the end of the iterative construction, output list 3 is created by performing linearization in each of the contraction domains in list 2, and then solving for the difference between the nonlinear dynamics function and its linearization. This is, in general, a global optimization problem, although guaranteed bounds can be computed using, for example, interval arithmetic. This is also an embarrassingly parallel problem, which can be exploited to speed up this computationally expensive step.

Finally, the last element of list 1 is removed, so that the last mode in the constructed chain will not be split. This process results in three lists, the first of size $N - 1$, and the

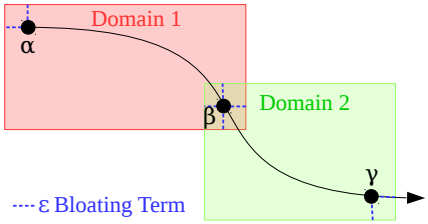


Figure 3: Two time-triggered steps use numerical simulations to create two contraction domains.

other two of size N, as is needed by the proposed mixed-triggered hybridization approach.

5.2 Generalizations

The proposed construction approach is simple in that only a small number of user-parameters are required. However, fine-tuning is possible which can create more precise abstractions, at the cost of requiring more input from the user.

First, the time step δ_{tt} could be changed for each domain. In Figure 3 this would correspond to the case where the difference in simulation times between points α and β is not the same as the difference between β and γ. Next, a per-domain bloating term ϵ is possible. Furthermore, each domain's bloating term could be further parameterized based on the face of the rectangular domain.

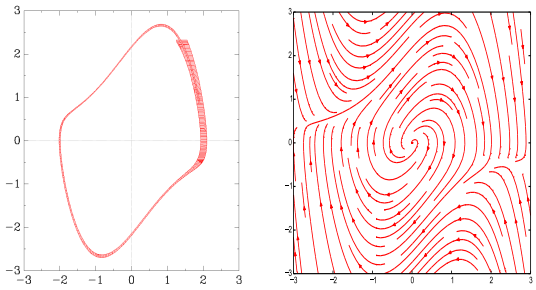
The domains need not be rectangles aligned to axes. Domains which are rotated rectangles, aligned with the direction of the flow, could reduce the error in the dynamics abstraction step. As with other hybridization work [21], domains which are triangles (simplices), or rotated variants could also be used. The complication with these approaches is that the global optimization step of domain abstraction, which is necessary for soundness, can become more complicated. For example, the simplex-based approach requires optimizing the Hessian matrix of the dynamics in a simplex domain, which may be difficult depending on the specific location's dynamics.

6. EVALUATION

As stated by Theorem 5.1, in order to soundly reason about the set of reachable states of the original automaton, the output automaton from the mixed-triggered hybridization process must not reach any DCEMs. The main purpose of the evaluation, therefore, is (1) to demonstrate that the hybridization parameters derived from simulations can result in models where DCEMs are not reached during reachability analysis of the output automaton. Additionally, we aim to (2) demonstrate the benefits of occasional space-triggered transitions compared with a pure time-triggered approach. Finally, we (3) demonstrate improved scalability by running our developed static approach on a higher dimensional model, at a granularity that would be impossible for existing static approaches. The evaluation was performed with these three goals in mind.

The proposed hybridization method was implemented in the HYST model translation and transformation tool [12][5]. The developed transformation pass implements the algorithm described in Section 5 leveraging the transformations

[5]SpaceEx model files for the examples evaluated, both before and after hybridization, are available at: http://verivital.com/hyst/pass-hybridization/.



(a) Computed reachability (b) Streamplot

Figure 4: The limit cycle for the Van der Pol system was computed with SpaceEx using our hybridization approach.

of Section 4. We target the latest version of the SpaceEx tool, which supports time-triggered transitions using the `map-zero-duration-jump-sets` flag. In order to derive the dynamics abstraction function, we use a global optimization routine from the `scipy.optimize` library. Other options are possible, for example interval arithmetic, interval arithmetic with grid-paving, SMT solvers, or combinations of these methods. Since the optimizations in each domain are run in parallel, more effort can be taken to derive tighter bounds without significant effects on overall runtime. The reported times were measured on a computer with an Intel Core 2 Quad CPU (Q9650) at 3.00 GHz with 4 GB RAM.

6.1 Van der Pol Oscillator

The first set of experiments consider a Van der Pol oscillator, which is a two-dimensional system with the following nonlinear dynamics:

$$\dot{x} = y$$
$$\dot{y} = (1 - x^2) * y - x$$

We use the same initial states as evaluated in other hybridization approaches [1], $(x, y) \in [1.25, 1.55] \times [2.28, 2.32]$. A maximum time of 5.5 was used, which is sufficient to complete one cycle of the oscillator, as in the earlier work.

We used numerical simulations based on the $\mathcal{S}$ = STAR strategy, a time-triggered step of $\delta_{tt} = 0.05$, a bloating term of $\epsilon = 0.05$, a number of space-triggered transformation steps of $n_{pi} = 31$, and a maximum simulation time in a space-triggered transformation step of $\delta_{pi} = 1$. Analyzing the generated model with SpaceEx resulted in no DCEMs being reached, which means that the set of reachable states overapproximates the set of reachable states in the the original automaton. This demonstrates goal (1) of the evaluation. The combined hybridization and computation process took 10.3 seconds. A visualization of the resultant set of reachable states produced by SpaceEx is given in Figure 4a, and can be compared to a streamplot of the dynamics given in Figure 4b.

It is insightful to examine the bounding box of the numerical simulations upon entering each mode, and compare it to the bounding box of the set of reachable states at the same times. In particular, by looking at the maximum width in any dimension of the bounding box of `sims` and comparing it with the maximum width of bounding box of the set of reachable states, we can estimate how close the set of reachable states was to the boundaries of the contraction domains where a DCEM would be reached. A plot of these widths upon entering each mode is shown in Figure 5.

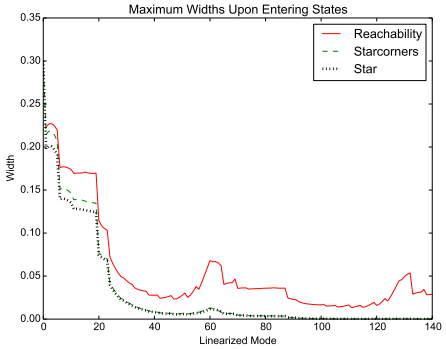


Figure 5: The maximum width of the bounding boxes of the reachable states and simulations upon entering each mode remains within $2 * \epsilon = 0.1$, which is necessary to avoid entering a DCEM.

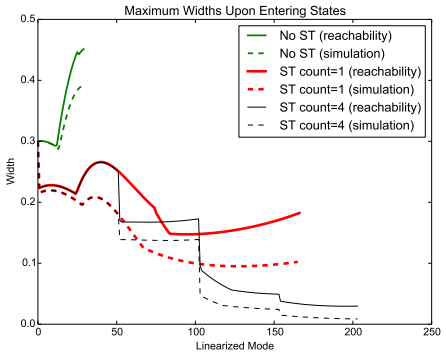


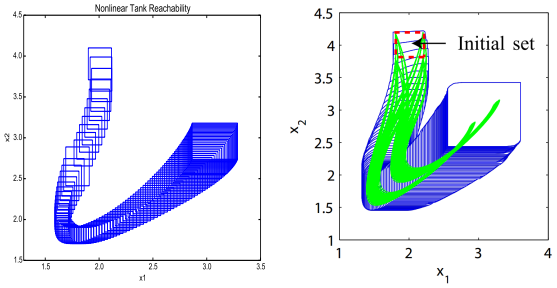
Figure 6: Space-triggered transitions serve to reduce the size of the tracked set of states.

Since we used a bloating term of $\epsilon = 0.05$, it is necessary that maximum width of the simulated states plus $2 * \epsilon = 0.1$ is greater than the maximum width of the set of reachable states at all times, otherwise, an error state will be reached. Additionally, from the plot we can see that the STARCORNERS strategy has slightly better tracking of widths of the set of reachable states near the start of the computation, although it makes less of a difference later on.

In order to show the effect of space-triggered transitions, we consider the same system using a shorter time bound of 2.0, a time step of $\delta_{tt} = 0.01$, and the same value of $\epsilon = 0.05$. We run the system with no space-triggered transitions, a single space-triggered transition at the start, and four space-triggered transitions. The widths of the tracked set of reachable states, and the bounding box of the simulated points is shown in Figure 6. Without space-triggered transitions, the width of the set of reachable states quickly gets larger than the simulated bounding box, and around time 0.29, a DCEM is reached. With a single space-triggered transition at the start, the tracked set of states is smaller, and a DCEM is not reached until around time 1.66. With four space-triggered transitions, the full 2.0 seconds is computed without reaching a DCEM. Furthermore, the decrease in the size of the tracked states is apparent at the space-triggered times 0.0 (mode #0), 0.5 (mode #51), 1.0 (mode #102), and 1.5 (mode #153). This demonstrates the effectiveness of space-triggered transitions in reducing the size of the currently-tracked set of states, goal (2) of the evaluation.

6.2 Nonlinear Water Tank

The second model we consider is a nonlinear tank model [5]. This model is parameterized on the number of tanks, n,



(a) Computed Reachability (b) Result from [5] (includes input disturbances)

Figure 7: A plot of a projection of the computed reachable states for x_1 and x_2 for the 6-D non-linear tank model.

where we use $n = 6$. Each tank i adds a single variable x_i to the model, which represents the height of the water in the tank. The input to the first tank is based on the level of the last tank, x_n. We analyze a deterministic version of the model, with no disturbance input and fixed tank parameters. The dynamics for x_1 and every other $x_{i>1}$ are:

$$\dot{x}_1 = 0.1 + 0.01(4 - x_n) + 0.015\sqrt{2gx_1}$$
$$\dot{x}_i = 0.015\sqrt{2gx_{i-1}} - 0.015\sqrt{2gx_i}$$

We used the same initial set of states as the earlier work, $x_1 \in [1.9, 2.1]$, $x_2 \in [3.9, 4.1]$, $x_3 \in [3.9, 4.1]$, $x_4 \in [1.9, 2.1]$, $x_5 \in [9.9, 10.1]$, and $x_6 \in [3.9, 4.1]$. Using the simulation strategy $\mathcal{S}$ = STARCORNERS, a maximum time of $T = 400$, a step size of $\delta_{tt} = 4$, a bloating term value of $\epsilon = 0.2$, a number of space-triggered transformation steps of $n_{pi} = 10$, and a maximum simulation time in a space-triggered transformation step of $\delta_{pi} = 10$, the hybridized model was created. SpaceEx was used to analyze this model, and indicated that no DCEMs were reached. The whole process took about 430 seconds. Figure 7 shows a projection of the set of reachable states onto x_1 and x_2, as well as a result from the earlier hybridization work.

This demonstrates goal (3) of the evaluation, that static-based hybridization approaches can scale to higher dimensions. Although only a six-dimensional model was considered, this is higher than we could find for any published static hybridization method.

7. CONCLUSION

In this paper, we developed the first static time-triggered and mixed-triggered hybridization approaches. The developed methods use simulations to guide the hybridization process and modify an input model for analysis with off-the-shelf verification tools, unlike dynamic hybridization methods that require tool modification. Additionally, we can perform the expensive dynamics abstraction (linearization) step for each mode in parallel, which can improve the speed of the method. We have shown the effectiveness of the method by hybridizing example nonlinear systems and computing the set of reachable states using SpaceEx, a tool that is only capable of reasoning with linear and affine systems.

Since this is the first paper investigating this category of hybridization techniques, we believe significant further optimization is possible. Extending the approach from single-mode input automata to multiple-mode systems would be a straightforward enhancement, and has been done in other

hybridization approaches [1]. Dynamic mixed-triggered approaches, have also yet to be investigated. Parameter selection for the approach can also be challenging and could be further automated, perhaps by using a CEGAR-like approach to detect when DCEMs (error modes) are reached, and performing additional simulations from violation regions. Finally, the simulation-based parameter construction algorithm does not track the set of reachable states well when nondeterminism or disturbances are present, and other approaches from hybrid automaton falsification may work better in these cases.

Acknowledgment

The material presented in this paper is based upon work supported by the Air Force Office of Scientific Research (AFOSR), in part under contract number FA9550-15-1-0258 and the Summer Faculty Fellowship Program (SFFP), by AFRL through contract number FA8750-15-1-0105, and by the National Science Foundation (NSF) under grant numbers CNS 1464311 and CCF 1527398. Furthermore, this research was supported in part by the European Research Council (ERC) under grant 267989 (QUAREM) and by the Austrian Science Fund (FWF) under grant numbers S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFRL, AFOSR, or NSF.

8. REFERENCES

[1] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, pages 173–182, New York, NY, USA, 2013. ACM.

[2] M. Althoff and J. Dolan. Set-based computation of vehicle behaviors for the online verification of autonomous vehicles. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, Oct 2011.

[3] M. Althoff and J. Dolan. Online verification of automated road vehicles using reachability analysis. *Robotics, IEEE Transactions on*, 30(4):903–918, Aug 2014.

[4] M. Althoff and B. H. Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Hybrid Systems: Computation and Control, HSCC'12, Beijing, China, April 17-19, 2012*, pages 45–54, 2012.

[5] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *47th IEEE Conference on Decision and Control (CDC)*, pages 4042–4048, Dec. 2008.

[6] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2), 2010.

[7] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[8] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In *Hybrid Systems: Computation and Control*, volume 2623 of *LNCS*, pages 20–35. Springer, 2003.

[9] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43:451–476, 2007.

[10] S.-i. Azuma, J.-i. Imura, and T. Sugie. Lebesgue piecewise affine approximation of nonlinear systems. *Nonlinear Analysis: Hybrid Systems*, 4(1):92–102, 2010.

[11] S. Bak. Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In *4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy 2014)*, pages 40–43, 2014.

[12] S. Bak, S. Bogomolov, and T. T. Johnson. HyST: A source transformation and translation tool for hybrid automaton models. In *Proc. of the 18th Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2015.

[13] S. Bogomolov, A. Donzé, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Abstraction-based guided search for hybrid systems. In E. Bartocci and C. R. Ramakrishnan, editors, *International SPIN Symposium on Model Checking of Software 2013*, LNCS. Springer, 2013.

[14] S. Bogomolov, A. Donze, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Guided search for hybrid systems based on coarse-grained space abstractions. *Software Tools for Technology Transfer (STTT)*, Aug. 2015.

[15] S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. S. Pasareanu, A. Podelski, and T. Strump. Assume-guarantee abstraction refinement meets hybrid systems. In *10th International Haifa Verification Conference (HVC 2014)*, volume 8855 of *LNCS*, pages 116–131. Springer, 2014.

[16] S. Bogomolov, C. Schilling, E. Bartocci, G. Batt, H. Kong, and R. Grosu. Abstraction-based parameter synthesis for multiaffine systems. In *11th International Haifa Verification Conference (HVC 2015)*, volume 9434 of *LNCS*, pages 19–35. Springer, 2015.

[17] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. *2013 IEEE 34th Real-Time Systems Symposium*, 0:183–192, 2012.

[18] T. Dang. Approximate reachability computation for polynomial systems. In J. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006.

[19] T. Dang, A. Donze, and O. Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In A. Hu and A. Martin, editors, *Formal Methods in Computer-Aided Design*, volume 3312 of *LNCS*, pages 21–36. Springer, 2004.

[20] T. Dang, C. Le Guernic, and O. Maler. Computing reachable states for nonlinear biological models. In *Computational Methods in Systems Biology*, pages 126–141. Springer, 2009.

[21] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In *Hybrid Systems: Computation and Control (HSCC)*, pages 11–20, New York, NY, 2010. ACM.

[22] P. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS, pages 68–82. Springer, 2015.

[23] G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *Hybrid Systems: Computation and Control (HSCC'13)*, pages 203–212. ACM, 2013.

[24] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[25] A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, LNCS. Springer, 2005.

[26] A. Girard and C. Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *LNCS*, pages 215–228. Springer, 2008.

[27] A. Girard, C. Le Guernic, et al. Efficient reachability analysis for linear systems using support functions. In *Proc. of the 17th IFAC World Congress*, pages 8966–8971, 2008.

[28] Z. Han and B. Krogh. Reachability analysis of nonlinear systems using trajectory piecewise linearized models. In *American Control Conference, 2006*, pages 6 pp.–, June 2006.

[29] T. Henzinger, P.-H. Ho, H. Wong-Toi, et al. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.

[30] L. Jaulin, M. Kieffer, and O. Didrit. *Applied interval analysis : with examples in parameter and state estimation, robust control and robotics*. Springer, London, 2001.

[31] T. T. Johnson, J. Green, S. Mitra, R. Dudley, and R. S. Erwin. Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In D. Giannakopoulou and D. Méry, editors, *Proceedings of the 18th International Conference on Formal Methods (FM 2012)*, pages 252–266. Springer, Paris, France, Aug. 2012.

[32] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In *Hybrid Systems II, volume 999 of LNCS*, pages 359–369. Springer, 1994.