Aggregation Strategies in Reachable Set Computation of Hybrid Systems

PARASARA SRIDHAR DUGGIRALA, University of North Carolina at Chapel Hill
STANLEY BAK, Safe Sky Analytics LLC

Computing the set of reachable states is a widely used technique for proving that a hybrid system satisfies its safety specification. Flow-pipe construction methods interleave phases of computing continuous successors and phases of computing discrete successors. Directly doing this leads to a combinatorial explosion problem, though, as with each discrete successor there may be an interval of time where the transition can occur, so that the number of paths becomes exponential in the number of discrete transitions. For this reason, most reachable set computation tools implement some form of set aggregation for discrete transitions, such as, performing a template-based overapproximation or convex hull aggregation. These aggregation methods, however, in theory can lead to unbounded error, and in practice are often the root cause of why a safety specification cannot be proven.

This paper proposes techniques for improving the accuracy of the aggregation operations performed for reachable set computation. First, we present two aggregation strategies over generalized stars, namely convex hull aggregation and template based aggregation. Second, we perform adaptive deaggregation using a data structure called Aggregated Directed Acyclic Graph (AGGDAG). Our deaggregation strategy is driven by counterexamples and hence has soundness and relative completeness guarantees. We demonstrate the computational benefits of our approach through two case studies involving satellite rendezvous and gearbox meshing.

Additional Key Words and Phrases: Hybrid Systems, Reachable Set, Linear Differential Equations, Aggregations for Reachable Set, Adaptive Deaggregation.

1 INTRODUCTION

Aeronautical systems such as air-traffic control protocols, auto-pilot software, and satellite maneuver protocols are safety critical in nature. Design errors in such systems, such as floating point bugs in Ariane 5 spacecraft, might lead to unsafe behaviors causing loss of property and in some cases, life. Testing such systems extensively under various scenarios might give confidence to the system designer that the systems functions in a safe manner. However, such extensive testing is not always possible. One of the widely used method for ensuring that the system does not encounter any unsafe scenarios in such cases is model-based analysis. In this method, a high-fidelity model of the system is created and extensive testing and verification is performed on the model. Hybrid automata is a well suited framework for modeling such safety critical systems and formal verification approaches for proving safety properties of several aeronautical systems modeled as a hybrid automata are widely available in the literature [13, 19, 21, 23–25, 28, 29].

In this paper, we perform safety analysis of two case studies, satellite rendezvous mission and gearbox meshing system modeled as a hybrid automata. We adopt a widely used technique for establishing safety properties of hybrid systems: *reachable set computation*. Given a set of initial configurations Θ (often uncountable), the reachable set is the set of all possible configurations encountered by the system trajectories starting from Θ. Since the reachable set is also uncountable,

This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.

Authors' addresses: Parasara Sridhar Duggirala, psd@cs.unc.edu, University of North Carolina at Chapel Hill; Stanley Bak, stanleybak@gmail.com, Safe Sky Analytics LLC.

we compute its symbolic representation. If the reachable set does not contain any unsafe configurations, then one can prove that the safety specification is satisfied. Often, symbolic representations of convex sets such as polytopes [14], zonotopes [17], support functions [16], etc. are used because operations such as linear transformation, intersection, convex hull, and Minkowski sum can be easily performed over these representations.

Such representations, however, are at a disadvantage while performing mode switches. Due to the non-determinism of mode switching behavior, one has to compute an overapproximation of all the states that can perform the mode switch in the chosen representation. This operation is called aggregation. Often, this overapproximation is very conservative and as a consequence, the reachable set computed might overlap with the unsafe set. Algorithmically selecting the sets for performing aggregation to improve accuracy involves combinatorial search and is a challenging task. However, if one does not perform aggregation, then one has to keep track of exponential number of symbolic representation after a finite number of mode switches.

This paper exclusively focuses on aggregation and de-aggregation strategies in reachable set computation. We present two aggregation strategies, first, template based, and second, convex hull based. For efficient de-aggregation, we introduce a data structure called Aggregated Directed Acyclic Graph (AGGDAG), and explain our de-aggregation strategy. We demonstrate that goal driven aggregation mechanisms along with template based aggregation and deaggregation can handle challenging case studies of satellite rendezvous and gerbox meshing involving challenging discrete transitions.

In a rendezvous mission, the satellite can operate in one of the two modes: approach or abort. In the approach mode, the trajectory of the satellite proceeds towards the rendezvous point. In the abort mode, the decision to rendezvous is aborted and hence the satellite should maintain a safe distance from the rendezvous point. The nondeterminism in the mode switch from approach to abort makes the safety analysis very challenging. Gear meshing system models the behavior of a sleeve switching gears. Depending on the angular position at which the sleeve arrives at the next gear, the impact forces cause the sleeve to bounce off the gear and delay the meshing process. The number of bounces the sleeve encounters increases the order of overapproximation due to aggregation. The aggregation and de-aggregation techniques proposed in this paper are crucial for combating the computational cost while improving the accuracy for performing safety analysis.

2 PRELIMINARIES

States and vectors, elements in $\mathbb{R}^n$, are denoted as x and v. In this work, we use the following mathematical notation of a linear hybrid automata.

Definition 1. A *linear hybrid automaton* is defined to be a tuple $\langle Loc, X, Flow, Inv, Trans, Guard, Reset\rangle$ where:

- Loc is a finite set of locations (also called modes).
- $X \subseteq \mathbb{R}^n$ is the state space of the behaviors.
- $Flow : Loc \to AffineDeq(X)$ assigns an affine differential equation $\dot{x} = A_l x + B_l$ for location l of the hybrid automaton.
- $Inv : Loc \to 2^X$ assigns an invariant set for each location of the hybrid automaton.
- $Trans \subseteq Loc \times Loc$ is the set of discrete transitions.
- $Guard : Trans \to 2^X$ defines the set of states where a discrete transition is enabled.
- $Reset : Trans \times X \to X$ defines the reset function that determines the next state after the discrete transition.

For a linear hybrid automaton, the invariants and guards are given as a conjunction of linear constraints and the reset function is an affine function.

The *set of initial states* $\Theta \triangleq (loc_0, S_0)$ where $loc_0 \in Loc$ is called the initial location and S_0 is given as a conjunction of linear constraints. An *initial state* q_0 is a pair (Loc_0, x_0), such that $x_0 \in X$, and $(Loc_0, x_0) \in \Theta$. Unsafe states U is also given as a conjunction of linear constraints.

Definition 2. Given a hybrid automaton and an initial set of states Θ, an *execution* of the hybrid automaton is a sequence of trajectories and actions $\tau_0 a_1 \tau_1 a_2 \ldots$ such that (i) the first state of τ_0 denoted as q_0 is in the initial set, i.e., $q_0 = (Loc_0, x_0) \in \Theta$, (ii) each τ_i is the solution of the differential equation of the corresponding location Loc_i, (iii) all the states in the trajectory τ_i respect the invariant of the location Loc_i, and (iv) the state of the trajectory before each action a_i satisfies $Guard(a_i)$.

The set of states encountered by all executions that conform to the above semantics is called the *reachable set*. For linear systems, the closed form expression for the trajectories is given as $\tau_i(t) = e^{A_l t}\tau(0) + \int_0^t e^{A_l(t-\mu)}B_l d\mu$ where A_l and B_l define the affine dynamics of the mode. Instead of computing the reachable set of states, we compute the set of states which can be reached by a fixed simulation algorithm. We call this reachable set as *simulation equivalent reachable set*, defined in [6]. We provide the details here for completeness.

Definition 3. A sequence $\rho_H(q_0, h) = q_0, q_1, q_2, \ldots$, where each $q_i = (Loc_i, x_i)$, is a (q_0, h)-simulation of the hybrid automaton H with initial set Θ if and only if $q_0 \in \Theta$ and each pair (q_i, q_{i+1}) corresponds to either: (i) a continuous trajectory in location Loc_i with $Loc_i = Loc_{i+1}$ such that a trajectory starting from x_i would reach x_{i+1} after exactly h time units with $x_i \in Inv(Loc_i)$, or (ii) a discrete transition from Loc_i to Loc_{i+1} (with $Loc_{i-1} = Loc_i$) where $\exists a \in Trans$ such that $x_i = Reset(a, x_{i+1})$, $x_i \in Guard(a)$ and $x_{i+1} \in Inv(Loc_{i+1})$. Bounded-time variants of these simulations, with time bound $k \times h$, are called (q_0, h, k)-simulations.

Definition 4 (Simulation-Equivalent Reachable Set)**.** Given a hybrid automaton H, initial set Θ, bounded time T, and simulation step size h, the simulation equivalent reachable set RS is the set of all states y such that there exists a simulation $\rho_H(q_0, h, k)$ with $q_0 \in \Theta$ that visits y.

Definition 5 (Simulation-Equivalent Safety)**.** A hybrid automaton H with initial set Θ, time bound T, step size h, and unsafe set U is said to be Simulation-equivalent safe, if all the simulations $\rho_H(q_0, h, k)$ with q_0 from Θ do not visit the unsafe set U.

Remark 1 Note that simulation-equivalent safety does not ensure that all executions (Definition 2) of the hybrid automaton are safe. The execution might encounter an unsafe state in between the time instances for performing safety verification. To the authors' knowledge, none of the tools available can provide relative completeness guarantees for safety property of executions. We stick to the notion of simulation equivalent safety because of two reasons. First, we can provide soundness and relative completeness guarantees and second, for every unsafe system, our approach can generate a counterexample simulation. These guarantees come at a cost: the user should select the appropriate step size for analyzing the system under. In the case studies analyzed in this paper, we highlight the different choices of this step size and present the results.

2.1 Symbolic Representation: Generalized Stars

We include the basic details of the symbolic representation called generalized stars [6, 12] with some syntactic modifications. Operations such as linear transformation, intersection, and Minkowski sum can be performed very efficiently over generalized stars, making them very suitable for reachable set computation.

Definition 6. A *generalized star* (or simply star) Θ is a tuple $\langle a, G, P\rangle$ where $a \in \mathbb{R}^n$ is called the *anchor*, $G = \{g_1, g_2, \ldots, g_n\}$ is a set of vectors in $\mathbb{R}^n$ called the *generators*, and $P : \mathbb{R}^n \to \{\top, \bot\}$ is a predicate. A generalized star Θ defines a subset of $\mathbb{R}^n$ as follows.

$$[\![\Theta]\!] = \{x \mid \exists \bar{\alpha} = [\alpha_1, \ldots, \alpha_n]^T \text{ such that } x = a + \Sigma_{i=1}^{n} \alpha_i g_i \text{ and } P(\bar{\alpha}) = \top\}$$

Sometimes we will refer to both Θ and $[\![\Theta]\!]$ as Θ.

The generalized stars that we encounter in our analysis have predicate P defined as a conjunction of linear constraints.

Example 2.1. A unit square S in the cartesian plane with corners at $(0, 0)$ and $(1, 1)$ can be represented as a star $\langle a, G, P\rangle$ where $a = (0, 0)$, $G = \{i, j\}$ where i and j are unit vectors along the x and y axes respectively, and $P \triangleq 0 \leq \alpha_1 \leq 1 \wedge 0 \leq \alpha_2 \leq 1$.

A set can be represeted in multiple ways in generalized star representation. Changing the anchor to $a' = (1, 1)$ results in the new predicate $P' \triangleq -1 \leq \alpha_1 \leq 0 \wedge -1 \leq \alpha_2 \leq 0$. Changing the generators to $G' = \{i + j, i - j\}$ would result in the predicate $P'' \triangleq -\sqrt{2} \leq \alpha_1 + \alpha_2 \leq 0 \wedge -\sqrt{2} \leq \alpha_1 - \alpha_2 \leq 0$. Observe that changing the anchor would correspond to translation and changing the generators corresponds to a linear transformation over the predicate P.

Given two generalized stars $\Theta_1 \triangleq \langle a, G, P_1\rangle$ and $\Theta_2 \triangleq \langle a, G, P_2\rangle$, $\Theta_1 \cap \Theta_2 \triangleq \langle a, G, P_1 \wedge P_2\rangle$. Here, the predicates P_1 and P_2 are defined over the same set of variables. If two stars do not share the same anchor and generators, one has to perform translation and linear transformation to have the same anchor and generators.

2.2 Reachable Set Computation of Linear Dynamical Systems Using Generalized Stars

In this section we will outline the reachable set computation of linear dynamical systems that uses a symbolic representation called *Generalized Stars*. Generalized star representation leverages the superposition property of the linear dynamical systems [12].

Input : Initial Set: $\Theta = \langle a, G, P\rangle$, Dynamics: (A, B), Step: h, Bound: k
Output: $Reach(\Theta) = Reach_0(\Theta), \ldots, Reach_k(\Theta)$

for *each* i *from* 0 *to* k **do**
  $a_i \leftarrow \rho(a, h, k)[i]$;
  for *each* $g_j \in V$ **do**
    $g'_j \leftarrow \rho(a + g_j, h, k)[i] - a_i$;
  $G_i \leftarrow \{g'_1, \ldots, g'_m\}$;
  $Reach_i(\Theta) \leftarrow \langle a_i, G_i, P\rangle$;
  Append $Reach_i(\Theta)$ to $Reach(\Theta)$;
return $Reach(\Theta)$;

Algorithm 1: Algorithm that computes the reachable set for a linear dynamical system at time instances $i \cdot h$ from $n + 1$ simulations.

Given an initial set $\Theta \triangleq \langle a, G, P\rangle$ with $G = \{g_1, g_2, \ldots, g_n\}$, we compute the reachable set for a linear dynamical system $\dot{x} = Ax + B$ using simulations. We generate simulations starting from a (denoted as $\rho(a, h, k)$), and $a + g_j$ for all $1 \leq j \leq n$ (denoted as $\rho(a + g_j, h, k)$, respectively). For a given time instance $i \cdot h$, the reachable set denoted as $Reach_i(\Theta)$ is defined as $\langle a_i, G_i, P\rangle$ where $a_i = \rho(a, h, k)[i]$ and $G_i = \langle g'_1, g'_2, \ldots, g'_n\rangle$ where $\forall 1 \leq j \leq n, g'_j = \rho(a + g_j, h, k)[i] - \rho(a, h, k)[i]$.

Readers can refer to [6] for the correctness of this algorithm. All the trajectories starting from Θ satisfy the safety specification if $Reach(\Theta)$ does not overlap with the unsafe set U.

Remark 2 Notice that the predicate in the reachable set remains same as that of the initial set Θ. Hence, if one changes the initial set by changing the predicate, one needs not generate additional simulations for computing the reachable set. One can just change the predicate in line 6 and compute the new reachable set. For checking the safety specification with the new initial set, we still have to check perform the overlap of the new reachable set with the unsafe set U.

2.3 Reachable Set Computation of Linear Hybrid Systems Using Generalized Stars

The reachable set computation technique for hybrid automata has three subroutines. First, it computes the reachable set of the dynamical system in the current mode of operation (line 6). Second, it computes the overlap of the reachable set in the current mode with the mode invariant (line 7). Third, it computes the overlap of the reachable sets with the guards of discrete transitions that cause a change in mode (line 9). When a discrete transition is performed, the reachable set in the new mode is computed by invoking the Algorithm 1 subroutine. Algorithm 2 is a pseudocode description of the algorithm. This reachable set computed is simulation equivalent reachable set, i.e., a state is in the reachable set if and only if there exists at least one simulation that visits the state.

Input : Initial set Θ, Hybrid system H, Step: h, Bound: k.
Output : *ReachSet* as the set of reachable states.

```
1  queueStars ← ∅;
2  append (Θ, loc_0, 0) to queueStars;
3  ReachSet ← ∅;
4  while queueStars is not empty do
5      S ← dequeue(queueStars);
6      R ← ReachSetDynSys(S, S.loc, h, k − S.time);
7      R′ ← InvariantOverlap(R, R.Inv);
8      ReachSet ← ReachSet ∪ R′;
9      nextRegions ← discreteTrans(R′, H.Trans);
10     append nextRegions to queueStars;
11 return ReachSet;
```

Algorithm 2: Algorithm that computes bounded time simulation equivalent reachable set.

We want to highlight that the reachable set is only computed at discrete instances of time. However, the advantage of our approach is that we can provide a counterexample when the safety specification is violated. Algorithm 2 terminates because of two reasons. First, the simulations considered in this paper spend at least one step in its current mode of operation. Second, we compute the reachable set for only a bounded number of steps k. Hence, the set of states in *queueStars* is finite.

One of the primary drawbacks of Algorithm 2 is in handling the discrete transitions. Suppose that in a given location, the number of stars that overlap with the guard of a discrete transition (line 9 in Algorithm 2) is m. As a result, the number of stars in the *queueStars* will become $O(m^2)$ after 2 discrete transitions. After η number of discrete transitions, the number of states in *queueStars*

grows to $O(m^n)$. To avoid the exponential blow up of the number of sets in *queueStars*, reachable set computation tools often use aggregation.

Remark 3 While Algorithm 2 might compute exponential number of stars, we do not generate exponential number of simulations for each mode in the hybrid automata. For each mode, we decide on an anchor a_{mode} and generators G_{mode} and pre-compute the $n+1$ simulations required for reachable set computation for that mode. For computing the reachable set of a star in a new mode (line 6), we change the anchor and the generators of the star to a_{mode} and G_{mode} respectively and use the observation in Remark 2.

While this helps us cut down the number of simulations, we still have to check for overlap of unsafe set with exponential number of stars. Aggregation is crucial step to decrease the number of such checks.

3 AGGREGATION AND DEAGGREGATION

In aggregation, the set of all stars in *queueStars* that are making a discrete transition to the same mode are collected together. Say, these are $S_1, S_2, \ldots, S_m$. Then, an overapproximation of these S' is computed such that $S_1 \cup S_2 \cup S_3 \ldots \cup S_m \subseteq S_{over}$. Instead of computing the reachable set for each of $S_1, S_2, \ldots, S_m$, the reachable set of S_{over} is computed in the future modes.

There are two main drawbacks of this aggregation mechanism. First, the union of sets $S_1, S_2, \ldots, S_m$ is often a non-convex set. Whereas the representation used for computing reachable set is used for representing convex sets. Therefore, this overapproximation of a non-convex set by a convex set is very conservative. More worryingly, the reachable set of S_{over} will trigger additional discrete transitions that would not happen while computing the reachable sets using $S_1, S_2, \ldots, S_m$. Such discrete transitions are artifacts of the conservative overapproximation during the aggregation process.

To overcome the two main challenges, we make the following modifications to the reachable set computation algorithm. First, while handling discrete transitions, we perform aggregation for all the sets in the *queueStars* that go to the same mode. The resultant star is tagged as an aggregate and the reachable set computation continues where the sets are tagged as aggregate. This way of computing the reachable set will result in a conservative overapproximation. If one of the sets in the computation overlaps with the unsafe set U, we check if the set is tagged as aggregate. If so, then we go to the initial set in the location and perform deaggregation and recompute the reachable set. Hence, we perform counterexample guided deaggregation. Our algorithm (Algorithm 3) terminates after we find either a counterexample for safety specification or prove that the overapproximation of the reachable set does not overlap with unsafe set.

A working example of the aggregation and deaggregation is provided in Figure 1. In Algorithm 3, the main loop (from lines 4- 24) continues until *queueStars* is empty. We first dequeue a star from

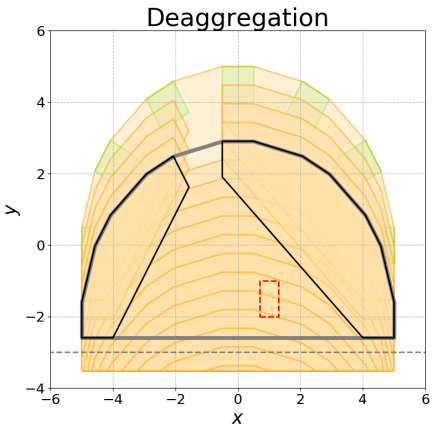


Fig. 1. The deaggregation process is shown for a two-mode system. Upon reaching an error mode (red dotted region), the fully aggregated set of states (gray large region), is split in half (two black regions), which no longer contain error states. A video of the complete computation is available online at https://www.youtube.com/watch?v=SDzGKDBq5tM.

Input : Initial set Θ, Hybrid automaton H, Step h, Bound k, Unsafe set U.
Output : If there exists a trajectory from Θ visits U.

```
queueStars ← ∅;
append (Θ, loc_0, 0) to queueStars;
ReachSet ← ∅;
while queueStars is not empty do
    S_1 ← deque(queueStars);
    if S_1.tag = unaggregated then
        S_nxt ← S_1;
    else
        listAgg ← stars in queueStars with mode S_1.loc;
        S_nxt ← Aggregate(listAgg);
        S_nxt.tag ← aggregated;
        dequeue listAgg from queueStars;
    R ← ReachSetDynSys(S_nxt, S_nxt.loc, h, k − S_nxt.time);
    R' ← InvariantOverlap(R, R.loc);
    if R' ∩ U ≠ ∅ and R'.tag = aggregated then
        S_agg ← first aggregated star in path from S_nxt to Θ;
        newStars ← deaggregate(S_agg);
        newStars.tag ← unaggregated;
        Enqueue newStars into queueStars;
    if R' ∩ U ≠ ∅ and R'.tag ≠ aggregate then
        return There exists a trajectory from Θ visiting U;
    ReachSet ← ReachSet ∪ R';
    nextRegions ← discreteTrans(R', H.Trans);
    append nextRegions to queueStars;
return All trajectories are safe;
```

Algorithm 3: Algorithm that performs aggregation and deaggregation for checking safety of the trajectories originating from Θ.

this queue and based on the tag, perform aggregation operation with all the stars for the same mode.
We then compute the reachable set of the aggregated star in this mode. If one of the stars in the reachable set overlaps with the unsafe set, we check if the reachable set is tagged as *aggregated*. If so, we pick the first aggregated star in the path from the current star to the root Θ and perform deaggregation of the selected star.

These deaggregated stars are added to *queueStars*, tagged as *unaggregated*, and the reachable set computation continues. If the reachable set of an *unaggregated* star overlaps with the unsafe set, then the algorithm returns that the safety specification is violated.

Lemma 3.1. *Algorithm 3 will return safe if and only if all the simulations starting from Θ for bounded time k are safe.*

Proof. This proof is a consequence of simulation equivalent reachability of Algorithm 2. We first prove that the Algorithm 3 is sound, that is, if the algorithm returns safe, then all simulations

are indeed safe and if the algorithm returns unsafe, then it is indeed unsafe. If the condition in line 20 of Algorithm 3 is satisfied, then the reachable set R' is equivalent to the one computed without any aggregation and hence is simulation equivalent. Therefore the system is indeed unsafe. Hence, whenever the algorithm returns unsafe, the system is indeed unsafe. If the algorithm returns safe, then the reachable set computed using aggregation, which is clearly an overapproximation of the reachable set, does not overlap with unsafe set. Hence, the system is safe.

It now remains to prove that the loop in lines 4- 24 terminates. This is easy to infer as there are only finitely many reachable sets that we compute. Hence, we perform only finitely many aggregations. Since we strictly do not aggregate the stars that were deaggregated before, the condition in line 15 will only be encountered finite number of times. Hence the loop terminates and the algorithm either returns safe or unsafe. □

Remark 4 In Algorithm 3, by default, we perform aggregation of all the stars making a discrete transition to the same mode. Also, upon visiting unsafe set, we deaggregate the first aggregated star in the path from unsafe set to the root and do not perform any future aggregations in reachable set computation.

The algorithm we implement is different from Algorithm 3 in two ways. First, we take into account the sequence of modes visited by the star for performing aggregation. That is, if two stars have encountered a different sequence of mode switches, they might be aggregated into different stars. Second, the deaggregation need not be performed at the first aggregated star in the path from leaf to root. We perform the deaggregation of one of the aggregated stars in the path. However, we ensure that we do not re-aggregate the children of deaggregated star. As a result, the line 21 would be invoked only if all the stars in the path from unsafe set to the initial set were unaggregated.

One of the advantages of generalized stars is that it allows for easy and efficient aggregation and deaggregation. Additionally, we avoid computing the entire reachable set, but only compute the specific sections of the reachable set that are important for safety verification. More specifically, we recompute the reachable set *only at times where discrete transitions are enabled or the unsafe set is reached.* This targeted recomputation is the main difference between this work and [26], where HyPro [27] is invoked with increasingly accurate configuration parameters to reduce the error in reachable set computation on a per-mode basis in the search tree. To keep track of the aggregation, deaggregation, and time instants for selective recomputation, we maintain a data structure called Aggregated Directed Acyclic Graph (AGGDAG).

3.1 Aggregation of Generalized Stars

In this section, we present two techniques for performing aggregation of generalized stars. The first is template based aggregation and deaggregation and the second is aggregation using convex hulls.

3.1.1 Template Based Aggregation: In this paper, since all the stars we encounter have predicates that are conjunctions of linear constraints, our overapproximation is also a predicate which is a conjunction of linear constraints. In contrast to the template-based counterexample refinement work in [7], this paper can handle hybrid automata with affine differential equations.

Lemma 3.2. *Consider stars* $S_1 \triangleq \langle a, G, P_1 \rangle$, $S_2 \triangleq \langle a, G, P_2 \rangle$, . . ., $S_m \triangleq \langle a, G, P_m \rangle$ *where the anchor and generators for all the stars is the same. A star* $S' \triangleq \langle a, G, P' \rangle$ *is an overapproximation of the union, i.e.,* $S_1 \cup S_2 \cup \ldots \cup S_m \subseteq S'$*, if and only if* $(P_1 \vee P_2 \vee \ldots \vee P_k) \Rightarrow P'$.

For computing the predicate P', we use a template based method. For each location, a set of template directions $c_1^T, c_2^T, \ldots, c_l^T$ are provided by the user and the predicate P' is determined by

selecting the appropriate values of $d_1, d_2, \ldots, d_l$ such that the condition $(P_1 \vee P_2 \vee \ldots \vee P_k) \Rightarrow P'$ is satisfied where $P' \triangleq (c_1^T\alpha \leq d_1) \wedge (c_2^T\alpha \leq d_2) \wedge \ldots \wedge (c_l^T\alpha \leq d_l)$.

For computing d_j, $1 \leq j \leq l$, we solve m linear programming problems. d_j^i is the maximum value of $c_j^T\alpha$ in P_i. That is, $d_j^1 = \max c_j^T\alpha$ given $P_1(\alpha) = \top$. Similarly, $d_j^2 = \max c_j^T\alpha$ given $P_2(\alpha) = \top$. We also compute $d_j^3, \ldots, d_j^l$. The value of $d_j = \max\{d_j^1, d_j^2, \ldots, d_j^l\}$.

Input : Predicates $P_1, P_2, \ldots, P_m$, Template directions $c_1^T, c_2^T, \ldots, c_l^T$.
Output: Predicate P' such that $(P_1 \vee \ldots \vee P_m) \Rightarrow P'$.

```
for each template direction c_j^T do
    for each star S_i do
        d_j^i ← max c_j^T α given P_i(α) = ⊤;
    d_j ← max {d_j^1, ..., d_j^m};
return P' ≜ (c_1^T α ≤ d_1) ∧ (c_2^T α ≤ d_2) ∧ ... ∧ (c_l^T α ≤ d_l);
```

Algorithm 4: Algorithm that performs template based aggregate of stars.

Lemma 3.3. *The predicate P' returned by Algorithm 4 is such that $(P_1 \vee \ldots \vee P_l) \Rightarrow P'$.*

Observe that we only consider aggregation of stars with the same anchor and generators. This is because of two reasons. First, the stars that we desire to aggregate correspond to the same mode. Second, as suggested in Remark 3, in order to decrease the number of simulations, we change the anchor and generators of the star to the anchor and generators of the corresponding mode for computing the reachable set. Hence, we perform this aggregation after the transformation of the predicate.

It is also inexpensive to perform deaggregation of the stars aggregated using template directions. Suppose that the aggregation of the stars $S_1, S_2, \ldots, S_l$ results in too conservative overapproximation. It is then desirable to perform two separate aggregations, the first aggregation is of the first half of the stars $S_1, \ldots, S_{l/2}$ and the the second aggregation corresponding to remaining half of the stars $S_{l/2+1}, \ldots, S_l$. For this deaggregation, one can reuse the results of the linear programs computed in Algorithm 4.

One might worry that template based aggregation might require solving a lot of linear programs. However, by using warm start optimization, the cost of solving several linear programs on the same polytopes becomes amortized. In warm start optimization, the seed for the next iteration of simplex is obtained from the solution of a previous linear program. Hence, the simplex algorithm skips the step of finding the feasible solution and computation is only used for finding the optimal solution for the new cost function. Without such cost reduction, template based overapproximation becomes very expensive.

One of the disadvantages associated with the template based overapproximation is that the order of overapproximation is dependent on the template directions that are selected. In addition to the axis directions, we pick the template directions dependent upon the dynamics of the location. The most appropriate template directions for improving the accuracy of overapproximation is a future area of investigation.

3.1.2 Convex Hull Aggregation: Given stars $S_1, S_2, \ldots, S_m$, one way to perform aggregation is to compute convex hull. A widely implemented technique in Multi Parametric Toolbox (MPT) [22] for computing convex hulls of polytopes requires transforming the representation from face representation to vertex representation and vice versa. This conversion among representations can

possibly takes exponential time. We avoid these exponential time operations by using the symbolic orthogonal projections [18]. We include the basic details of this convex hull operation for the sake of completeness.

Definition 7. A symbolic orthogonal projection O is given as a pair of matrices $A \in \mathbb{R}^{m \times n}$ and $L \in \mathbb{R}^{m \times k}$ and $\mathbf{a}$ is a column vector in $\mathbb{R}^m$, represented as $(A, L, \mathbf{a})$ prepresents the set

$$O = \{ x \in \mathbb{R}^n \mid \exists z \in \mathbb{R}^k, Ax + Lz \leq \mathbf{a} \}$$

If a polytope is represented as a generalized star, there are no existentially quantified free variables in it. Hence, generalized stars that represent polytopes are special cases of symbolic orthogonal projections. The convex hull of two symbolic orthogonal projections, which can be computed by merely transforming the structural representations is presented below (taken from [18]).

Definition 8. Given two symbolic orthogonal projections $O_1 \triangleq (A_1, L_1, \mathbf{a_1})$ and $O_2 \triangleq (A_2, L_2, \mathbf{a_2})$, the convex hull of O_1 and O_2 is given as a symbolic orthogonal projection $O_3 \triangleq (A_3, L_3, \mathbf{a_3})$ where

$$A_3 = \begin{bmatrix} A_1 \\ \mathbf{0} \end{bmatrix}, L_3 = \begin{bmatrix} A_1 & L_1 & \mathbf{0} & \mathbf{a_1} \\ -A_2 & \mathbf{0} & L_2 & -\mathbf{a_2} \end{bmatrix}, \mathbf{a_3} = \begin{bmatrix} \mathbf{a_1} \\ \mathbf{0} \end{bmatrix}$$

Where $\mathbf{0}$ represents the zero matrix of the appropriate dimension.

The advantage of symbolic orthogonal projection over other representations is that convex hull can be computed purely syntactically. However, observe that if O_1 and O_2 had $n + k$ variables and m constraints, then the number of constraints in O_3 is $2m$ and the number of variables is $2n + 2k + 1$. If one desires to perform convex hull of r symbolic orthogonal projections, then the number of constraints increases by r fold and the number of variables also increases r fold (it is not exponential). Hence, the number of constraints and variables required to specify the polytope exponentially increases with the number of discrete transitions. This increases the cost associated with checking the safety property of all the stars in the reachable set. Additionally, the deaggregation operation cannot reuse the computations performed during aggregation.

3.2 Aggregated Directed Acyclic Graph - AGGDAG

In Remark 3, we observed that we need not generate exponential number of simulations for each mode. Rather, we convert the anchor and generator of a star to a_{mode} and G_{mode}. We reuse the same simulations for computing reachable set of modes after deaggreation. To further reduce the computation after deaggregation, we use Aggregated Directed Acyclic Graph (AGGDAG).

Observe that after deaggregation, there are only two instances where recomputing the reachable set contributes to the accuracy for safety verification. First, one needs to check if the reachable set of deaggregated stars overlaps with the unsafe set. Second, recompute the overlap of reachable set and the guards for discrete transitions. For time instances that do not overlap with any guard or with the unsafe set, one need not recomputate the reachable set. AGGDAG is a bookkeeping mechanism for keeping track of time steps for discrete transitions and time steps for overlap with the unsafe set.

The nodes in AGGDAG correspond to the modes visited during the reachable set computation. Each mode is tagged with the maximum time a star remains in that mode. Each transition in AGGDAG corresponds to a discrete transition from one mode to another.

Fig. 2. Example of an AGGDAG.

The transition is also labelled with
the time steps when the reachable set
has overlap with the guard condition for the transition. Finally, if stars that experience different sequence of mode switches are aggregated, the corresponding node in AGGDAG has two incoming edges from different nodes. Each of these discrete transitions would be labelled by the time interval for the corresponding discrete transition.

Definition 9. An AGGDAG is a directed graph (G, E) where, the set of nodes G corresponds to the modes visited during the reachable set computation that encounter the discrete transitions or overlap with the unsafe set, and the set of edges E represents the successor relationship among these modes.

Example 3.4. Figure 2 is an example of AGGDAG where the hybrid automata has 3 modes of operation, namely, *Far*, *Approaching*, and *Abort*. The initial set starting from the Far mode takes the discrete transitions to Abort in the time duration $[1, 56]$ and $[57, 112]$, it also takes a discrete transition to Approaching mode in $[109, 112]$ steps. Notice that Far mode is tagged with 112, which means that after 112 steps, all of the stars leave the Far mode.

Notice that the two Abort(199) nodes do not have any outgoing edges. This implies that the reachable states in these abort modes does not take any further discrete transitions. On the other hand, the Approaching node has an outoing edge to Abort and the edge is labelled with time interval $[1, 91]$. Hence, the reachable set in Approaching mode take a discrete transition to Abort mode in time interval $[1, 91]$. Similarly, the Abort node (more specifically Abort(90)) mode has one outoing edge to the Error node and the edge is labelled with interval $[16, 86]$.

To check if the safety property is violated, we first inspect the AGGDAG in Figure 2 to see if any Error node is present in it. Since reachable set in Abort mode overlap with unsafe set, we need to inspect whether this overlap is caused due to overapproximation induced in the aggregation of the reachable set in the path from the root node (Far node) to the Abort node or if it corresponds to an real behavior for safety violation.

The overapproximation of reachable set can be at two instances, first, the aggregation of stars in the Approaching mode in the interval $[109, 112]$ or second, in the Abort mode in the interval $[1, 91]$. If both these reachable set do not have any aggregation, then we have proof that the overlap with the unsafe set is indeed a safety violation. If one of the discrete transitions has an aggregation, we perform deaggregation and compute the new reachable set. The AGGDAG in Figure 2 is a result of one such deaggregation. In the previous iteration, the AGGDAG had only one transition to Abort(199) mode. The resulting reachable set of the aggregation had overlap with the Error state. After deaggregation, we updated the AGGDAG to represent two Abort(199) nodes with different time intervals for discrete transition, corresponding to the different sets that were aggregated. State of the art reachable set computation tools either perform full aggregation or no aggregation at all. In the case studies we show that this goal driven deaggregation along with template based aggregation can handle challenging hybrid automata with multiple discrete transitions.

The AGGDAG is useful only in bookkeeping the modes encountered during reachable set and the time intervals for the discrete transitions. The strategy for deaggregation is still decided by the user. In our tool, we have implemented two deaggregation strategies, first, from the leaf to root and second, from root to leaf. In the case of leaf to root, we first deaggregate the reachable sets that are closest to the safety violation and continue the deaggregation to the top. In Figure 2, in the leaf to root strategy, we would first deaggregate the states taking the discrete transition from Approaching to Abort. In the root to leaf strategy, the deaggregation is performed at the node closest to the root node in the path leading to the unsafe overlap. In Figure 2, under the root to leaf strategy, one would deaggregate the stars in the discrete transition from Far to Approaching. The

best deaggregation strategy for proving safety or discovering the counterexample is still an area to be investigated and is a part of future research.

4 CASE STUDIES

We evaluate our proposed method on two case studies, a spacecraft rendezvous benchmark and a gearbox meshing benchmark.

4.1 Spacecraft Rendezvous Passive Safety

The spacecraft rendezvous system consists of a primary chaser spacecraft moving towards a secondary, free-flying object (such as a satellite) and performing close-proximity maneuvers. The maneuver is analyzed in relative coordinates, as shown in Figure 3. The verification goal is to ensure *passive safety*: at any time in the maneuver, a system failure may occur and the resulting propulsion-free trajectory must avoid colliding with the target satellite. This requirement comes from real-world failures. In 2005, NASA's DART spacecraft was intended to rendezvous with the MUBLCOM satellite, but due to depleted propellant instead collided with the target satellite (a loss of a $110 million project) [11].

Our model is based on a published spacecraft rendezvous benchmark [8, 20]. The system is modeled as a hybrid automaton with different discrete modes depending on the sensors being used for navigation, and an LQR controller is designed to meet physical and geometric safety constraints. The relative dynamics are linearized using the Clohessy-Wiltshire-Hill (CWH) equations [10], which is often used in close proximity operations, and generally considered valid when spacecraft are within a few kilometers. The hybrid automaton consists of three modes, two for different navigation strategies, and one for the passive dynamics, shown in Figure 4. This system is a six-dimensional linear system, with nondeterministic transitions to the passive mode. In our analysis, we check for passive safety from $t_1 = 0$ to $t_2 = 140$. The dynamics and controller in each mode are as described in the benchmark paper [8]. We also use the same initial set of states, the box where $x \in [-925, -875]$, $y \in [-425, -375]$ and the velocities are zero. We focus on the collision-free safety requirement, and strive to verify that the spacecraft remain separated by at least 5 meters in the infinity norm (the unsafe set is a 10×10 box centered at the origin).

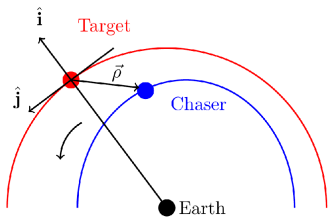


Fig. 3. Collisions are checked between spacecraft in orbit in relative coordinates (image from [8]).

Although several tools have successfully analyzed a version of this model in the ARCH hybrid systems tools competition in 2018 [1], a critical simplification was made: the competition model did not actually check the passive safety requirement. In particular, the competition model used a *fixed* time to transition to the passive mode. This is an unrealistic simplification, since the time of failure cannot be known in advance.

The analysis done in the original work [8] is slightly better, in that it checks for passive safety for a known 5 minute failure interval. In [8], the authors comment that, if larger time intervals are used, "the initial set of states under the Passive mode is large, making it very difficult to prove safety." The suggestion is then to create subintervals that cover the full time range of transitions to the passive mode, and then successively analyze each interval as a standalone verification problem. Presumably, a manual guess-and-check approach should be used to create these subintervals. This was done with CORA in the 2019 tools competition [2], where a larger passivity interval was analyzed through a manual process of selecting the aggregation subintervals.

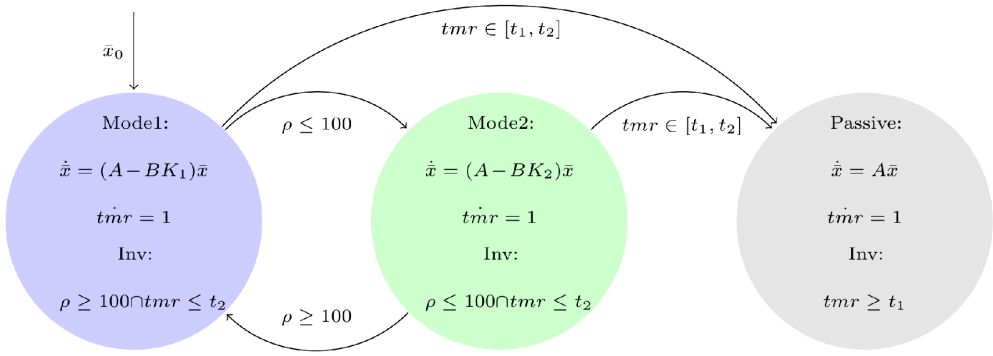


Fig. 4. Hybrid automaton model for our case study (image from [8], which also include the dynamics matrices). We check for passive safety from time $t_1 = 70 - r$ to $t_2 = 70 + r$, where r is a model parameter.

We consider a parameterized version of the problem, where a single parameter r controls the amount of nondeterminism in the switch to the passive mode. A simulation-based analysis revealed that entering the passive mode up to around time 140 will not violate the collision safety specification. We thus consider enable the switch to the passive mode from time $70 - r$ to time $70 + r$. In this way, $r = 0$ corresponds to the easy case where the switch occurs at exactly time 70, and $r = 70$ corresponds to the difficult case where the switch can occur at any time between 0 and 140. In the automaton in Figure 4, we set $t_1 = 70 - r$ and $t_2 = 70 + r$.

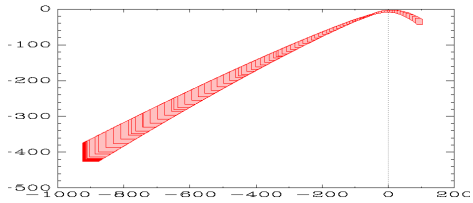
All the experiments are performed on a system with an Intel i5-5300U CPU running at 2.30GHz and 16GB RAM running Ubuntu Linux 18.04. Since we needed to script and measure many executions of the tools, we make use of the `hypy` [4] library distributed as part of the Hyst tool [3].

As a comparison, we also run the benchmark on the SpaceEx [16] verification tool. Our proposed aggregation and deaggregation enhancements are implemented as modifications to the publicly available Hylaa tool [5][1].

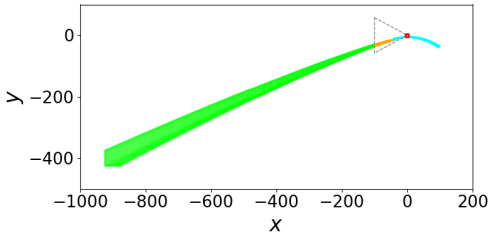
Note that SpaceEx analyzes the system using continuous time, whereas our approach builds off Hylaa, a discrete time (simulation equivalent) tool. Although SpaceEx needs extra operations to enable continuous-time analysis, this also allows it to use larger time steps where accuracy permits, as it will be known that the state will not jump through the unsafe region between steps (tunneling). Tunneling is possible with discrete-time analysis methods, so the choice of time step is an important parameter (we will analyze this with the Hylaa results). Another difference is that discrete-time methods permit the generation of counter-examples when specification violations occur, but continuous-time methods like SpaceEx do not generate these, as the detected violation may be due to overapproximation needed to handle continuous time. We expect the reach set to be qualitatively the same when comparing discrete-time analysis with small time steps and continuous-time analysis with sufficient accuracy. This is indeed the case, as shown in the plot of the reach set output from SpaceEx and our implementation in Figure 5.

4.1.1 SpaceEx. We use the latest space-time clustering (STC) reachability method implemented in SpaceEx [15] version `v0.9.8f`. Two aggregation methods are available, convex hull aggregation (`chull`) and error-guided partial aggregation (`none`[2]). We tune accuracy of the reachable set with the `flowpipe-tolerance` parameter as well as the number of support function directions to use,

[1]Our changes are in the public Hylaa repository at https://github.com/stanleybak/hylaa/releases/tag/EMSOFT2019-version.
[2]This was confirmed through discussions with the SpaceEx developers. The name is confusing.

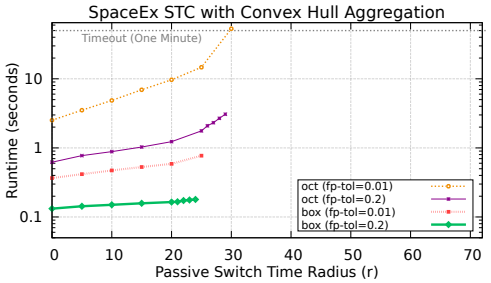


(a) SpaceEx (continuous time)

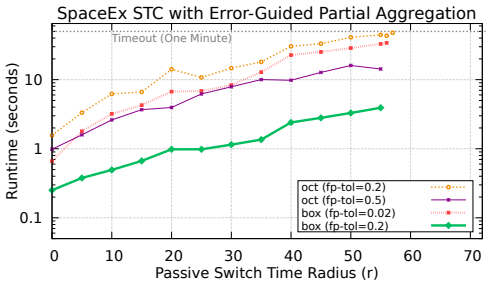


(b) Our implementation (discrete time)

Fig. 5. The reachability plots using SpaceEx (top) in continuous time and our implementation based off Hylaa (bottom) in discrete time are qualitatively similar. The initial states are in the lower left of the plots, the unsafe states are near the origin. In this plot, the switch to the passive mode occurs at exactly time 140.



(a) SpaceEx with convex hull aggregation can analyze the system up to around $r = 30$.



(b) SpaceEx with error-guided partial aggregation can analyze the system up to around $r = 55$.

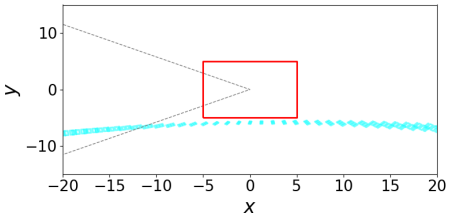
Fig. 6. Reachability analysis of satellite rendezvous using SpaceEx.

for which we consider both `box` and `oct`. By fixing the parameters, we can analyze the system as the passive-mode switch time parameter r is increased from 0 to 70. Since several options which trade off accuracy for computation time are available to the user, a fair comparison is difficult. We thus consider many permutations of these parameters, to try to find the best ones for each value of r. In the experiments we had a one minute timeout to run the verification task.

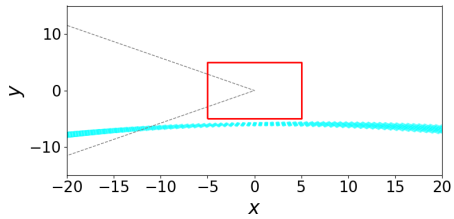
With convex hull aggregation (Figure 6a), the system can be analyzed successfully up to around $r = 25$. Some of the lines end before exceeding the timeout; these are the cases where increasing r by 1 would prevent successful analysis with those settings (safety could not be proven due to overapproximation). Different accuracy settings can slightly go beyond this, with $r = 30$ being possible in about one minute with `oct` directions and `flowpipe-tolerance=0.01`. This demonstrates the inherent overapproximation due to aggregation, where even modest uncertainty in the switch to the passive mode prevents verification.

With error-guided partial aggregation (Figure 6b), SpaceEx can analyze the system up to around $r = 55$. It can still not do full uncertainty (up to $r = 70$) since there is still some overapproximation from aggregation, the choice of flowpipe-tolerance and the choice of support function directions. We did not find parameter values where we could analyze for much larger values of r, although analysis should theoretically be possible by sufficiently increasing the accuracy settings and waiting long enough.

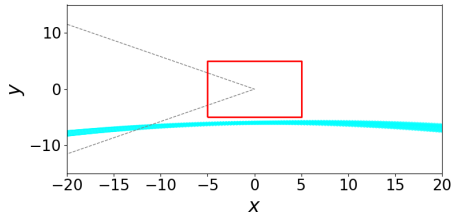
One final note is that parameter selection is difficult for SpaceEx, and it would be unreasonable to perform an exhaustive search for every model that needs analysis. Although we have presented



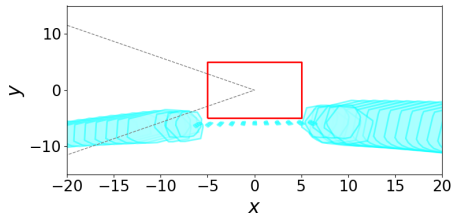
(a) Step Size 1.0



(b) Step Size 0.5

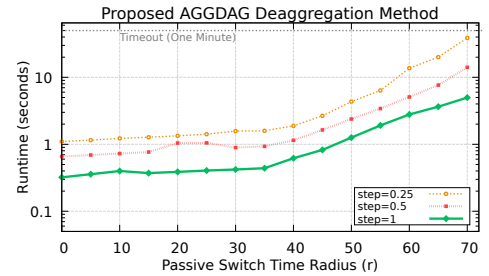


(c) Step Size 0.25

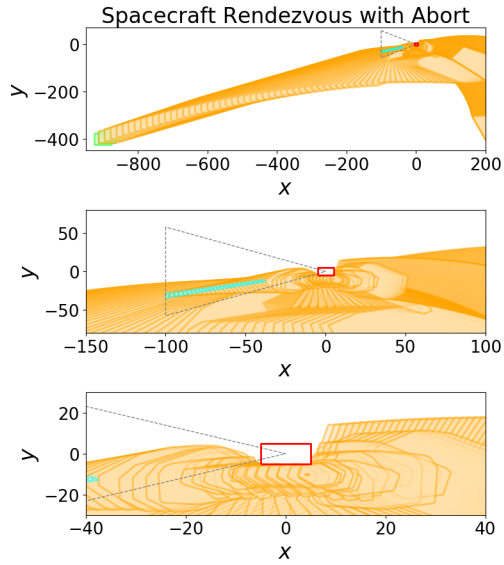


(d) Deaggregation with Step Size 1.0

Fig. 7. Plots of the discrete-time reach set near the unsafe state are shown with aggregation disabled (top three plots). While a step size of 1.0 is unlikely to tunnel through the unsafe states (red box), using 0.25 provides a continuous set of reachable states. Our aggdag approach produces the bottom plot (step size 1.0), showing improved accuracy near the unsafe states using deaggregation.



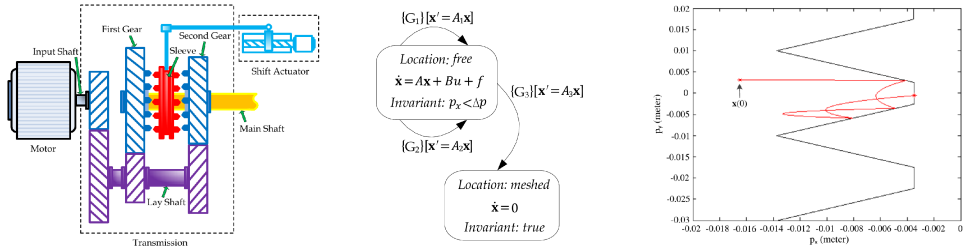
(a) The proposed AGGDAG approach can analyze up to the full $r = 70$, even for the smallest time step.



(b) The reachable set for the spacecraft rendezvous system at three different zoom levels is shown. Reachable states near the unsafe set (red square near origin) are deaggregated using the proposed approach until no unsafe states are reachable. A video of the computation is online at https://www.youtube.com/watch?v=iXJlJnsxeN0.

Fig. 8. Reachability analysis of satellite rendezvous using our method.

several reasonable options (and experimented with many others that were not presented), we have no guarantee that the combinations of options we tried were the fastest or most accurate. Further, there are other parameters we could have explored, such as the `lgg` reachability mode instead of `stc`, where a `clustering` parameter is available to more finely control the aggregation process. The SpaceEx help lists about 50 parameters, not all linked to method accuracy, that can affect the



(a) System Overview (b) Hybrid Automaton (c) Meshing Simulation

Fig. 9. Gearbox benchmark overview (images from [9]).

results of the computation, and it can be difficult to select the correct ones to explore, even for people familiar with the verification algorithms.

4.1.2 Proposed AGGDAG Approach. Since the proposed aggdag method was implemented on top of the discrete-time Hylaa tool, care must be taken to ensure that a proper time step is chosen. If the time step is too small, too many steps will be necessary and the analysis time can become excessive. On the other hand, if the time step is too large, the unsafe states may be missed if they are reached between time steps (this is called tunneling in collision detection methods). We examine the reach set with switching time 140, which goes close to the unsafe states.

The analysis is shown in Figure 7. This is zoomed-in plot of the system shown before from Figure 5b near the unsafe states. For the plot, it is clear that using a step size of 1.0 is unlikely to tunnel through the unsafe states, although the smaller step size 0.25 may be preferred as the reach set is a continuous set of state (the sets overlap between time steps).

We measure the performance of our method using all three step sizes. The result is shown in Figure 8a. Since all three step sizes will deaggregate if an unsafe set is reached, the analysis is exact with respect to the specification. This means that all the step sizes can successfully verify the system, even with maximum uncertainty in the switching time to the passive mode ($r = 70$). The only difference is the performance of the methods. In this case, all three step sizes verify the system before the one minute timeout. Furthermore, parameter tuning with our approach is a bit more straightforward, as the main parameter is just the step size, and initial analysis using simulations can be used to guide selecting a value for this.

A plot of the reachable state for the $r = 70$ case is shown in Figure 8b. Prior to our analysis, we were unaware of any tool which could successfully analyzed this model in the complete passivity settings ($r = 70$).

4.2 Gearbox Meshing

The gearbox meshing system [9] analyses the impact force and meshing time in a motor-transmission drive system, where a shift actuator is used to switch a sleeve between gears. Depending on the angular position with which the sleeve arrives at the next gear, impacts may occur which can delay the meshing process. The sleeve may bounce off the gears several times before meshing succeeds. The hybrid automaton model of this system consists of a `free` mode and a `meshed` mode, where the system can take circular discrete transitions in the `free` mode corresponding to the sleeve bouncing off the sides of the gear, as well as a final transition from the `free` mode to the `meshed` mode when meshing succeeds. Resets along discrete transitions are used compute the accumulated impact impulse, which changes instantaneously whenever a transition is taken. The system has five variables: two for the sleeve position and two for the sleeve velocity, and a one

Table 1. Gearbox Parameter Evaluation

Parameter	Runtime (s)	Deaggregation Steps
Deagg Leaves First	17.6	52
Deagg Root First	11.2	35
Deagg Most States	16.1	49
Box Agg	10.2	38
Convex Hull Agg	`timeout`	-
No Deaggregation	`timeout`	-
SpaceEx	`timeout`	-

further variable to track the accumulated impact impulse. The analysis goal is to find the worst-case accumulated impact impulse, as well as the longest meshing time. An overview of the system is shown in Figure 9, where Figure 9a shows the sleeve moving between two gears, Figure 9b shows the hybrid automaton model of the system, and Figure 9c shows a simulation where the sleeve bounces off the gear three times before meshing.

This model was also used as part of the ARCH hybrid systems verification tools competition [2]. In order to make analysis tractable, a critical simplification was made: a small set of initial states was analyzed. The initial set of states used for competition was so small, in fact, that all reachable states go through the same exact sequence of discrete transitions as the simulation shown in Figure 9c, with no branching possible in system executions. The reason for this is that aggregation error can quickly accumulate over multiple discrete jumps, so that existing aggregation strategies easily lead to more and more error. The additional error can then make it appear that the sleeve can bounce off the gear indefinitely, without meshing.

We analyze this system using the proposed AGGDAG and deaggregation approach with a significantly larger set of initial states. While the competition benchmark used the initial set of positions $x_0 \in [-0.0168, -0.0166]$, $y_0 \in [0.0029, 0.0031]$, we analyze the larger initial set $x_0 \in [-0.017, -0.016]$, $y_0 \in [-0.005, 0.005]$. We use a step size of 0.001 and analyze up to a time bound of 0.35, which is sufficient for all executions to reach the `meshed` mode, as we will demonstrate with reachability analysis. One hundred simulations from random initial states are shown in Figure 10, which show that many different sequences of discrete transitions are possible. Some executions immediately mesh, while others bounce several times before meshing.

Since we do not have an unsafe set of states in this system, we instead drive the deaggregation process based on splits in the discrete state transition graph. That is, if a single aggregated star can reach multiple guards when computing the reach set of the dynamical system in a single mode, then we will perform deaggregation. This is done by modifying line 15 in Algorithm 3. This use of deaggregation ensures that error from aggregation cannot introduce any spurious discrete transition sequences.

The reachable set computed by the approach is shown in Figure 11. The worst-case meshing time is computed to be around 0.31 seconds, with a worst-case impact impulse of about 35 Nm. The red line on the figure shows a witness simulation that bounces seven times before completing the meshing process. Notice this worst case was not observed during the random simulations performed in Figure 10.

We perform a preliminary exploration of the effects of various parameters available with our approach. The results are shown in Table 1.

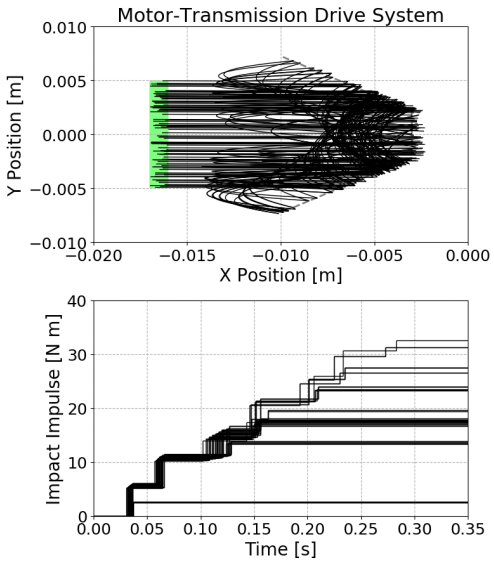


Fig. 10. One hundred random simulations are shown for the gearbox system on the x/y plane (top), as well as the total impact impulse over time (bottom). Notice the worst-case accumulated impact impulse, shown in Figure 11, was not observed in this simulation batch, demonstrating that simulations can miss important system behaviors. A video of the simulations is online at https://youtu.be/IBwKe2g4Rb0.

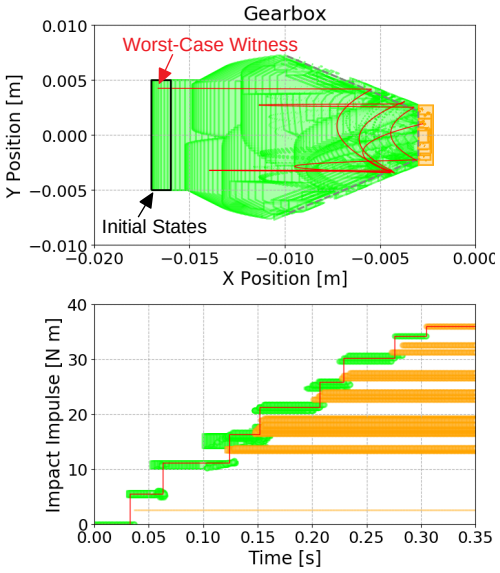


Fig. 11. The reachable set for the gearbox system is shown on the x/y plane (top), as well as the total impact impulse over time (bottom). Green states are in the `free` mode while orange states are `meshed`. The red line shows a worst-case witness which bounces seven times and reaches the maximum impact before meshing. A video of the reachability computation is online at https://youtu.be/trE0qjMTQmo, and a video of the worst-case witness simulation available at https://youtu.be/_C0YpEmCwKI.

For the deaggregation order exploration, we use template-based aggregation based on the mode dynamics. In this case, root-first deaggregation is more efficient, both in terms of runtime and the number of deaggregation operations needed. We suspect this is because overapproximation error early on in the computation leads to error later on, so even if deeaggregation is performed first on the leaves, eventually the states closer to the root will need to be deaggregated as well.

For evaluating the aggregation method, we used root-first deaggregation. There were two interesting observations. First, box aggregation takes slightly less time than the default template-based aggregation, even though there are more deaggregation steps needed. We attribute this speedup to the simpler star sets with the box method being easier to optimize over using an LP solver. Observing the output of the reachable set, we noticed that the box method did have more error during the continuous post operation (states to the left of the initial sets were reached when using box aggregation), although deaggregation ensured the sequences of discrete transitions was the same as with the more accurate template-based approach. Essentially, in this case the box method was faster, but had more error. Second, using the convex hull aggregation approach exceeded our one minute timeout. This was because although the method from Section 3.1.2 provide the exact convex hull, the number of variables and constraints in result of aggregating two stars is proportional to the sums of variables and constraints in the component stars. This makes the LPs

grow larger in size, thus slowing the method down. In this case, the result will be more accurate, but the method is slower.

Finally, we compared with existing approaches. Using no deaggregation (complete aggregation), the method reached the timeout. Observing the visualization we could see the suspected effect that aggregation error was growing across each jump, and the additional error caused the sleeve to appear to be able to bounce off the gear indefinitely, without meshing. With SpaceEx, we used the same settings as was used in the ARCH hybrid systems tools competition [2], changing only the initial set of states. By limiting the maximum number of discrete transitions and observing the output, we could observe the same effect; the reachable states was growing as more and more discrete transition were processed, which gave the appearance it was possible for the sleeve to bounce indefinitely without meshing. Aggregation error led to spurious discrete transitions being taken, which led to additional aggregation error and further spurious discrete transitions, and this cycle repeated indefinitely.

5 CONCLUSIONS

In this paper we have focused on computing accurate reachable set computation of hybrid automata where there is high nondeterminism in the discrete transitions. We presented two common techniques used for aggregation and highlighted the relative merits and demerits of each technique. We also presented AGGDAG data structure and outlined the deaggregation strategies that were implemented. Using the techniques we were able to handle the challenging case studies of satellite rendezvous mission and gearbox meshing.

Handling discrete transitions is still a major hurdle in scalable and accurate computation of reachable set for linear hybrid systems. As a part of future work, we intend to explore intelligent aggregation and deaggregation strategies that adapt based on the dynamics to provide an accurate reachable set.

ACKNOWLEDGEMENTS

The work done in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS 1739936, 1935724. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NSF. Effort sponsored in whole or in part by the Air Force Research Laboratory, USAF, under Memorandum of Understanding/Partnership Intermediary Agreement No. FA8650-18-3-9325 and prime contract FA8650-15-D-2516. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory.

REFERENCES

[1] Matthias Althoff, Stanley Bak, Xin Chen, Chuchu Fan, Marcelo Forets, Goran Frehse, Niklas Kochdumper, Yangge Li, Sayan Mitra, Rajarshi Ray, Christian Schilling, and Stefan Schupp. 2018. ARCH-COMP18 Category Report: Continuous and Hybrid Systems with Linear Continuous Dynamics. In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, Vol. 54. 23–52.

[2] Matthias Althoff, Stanley Bak, Marcelo Forets, Goran Frehse, Niklas Kochdumper, Rajarshi Ray, Christian Schilling, and Stefan Schupp. 2019. ARCH-COMP19 Category Report: Continuous and Hybrid Systems with Linear Continuous Dynamics. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*. 14–40.

[3] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. 2015. HyST: A Source Transformation and Translation Tool for Hybrid Automaton Models. In *18th International Conference on Hybrid Systems: Computation and Control*. ACM.

[4] Stanley Bak, Sergiy Bogomolov, and Christian Schilling. 2016. High-level Hybrid Systems Analysis with Hypy. In *ARCHâĂŹ16: Proc. of the 3rd Workshop on Applied Verification for Continuous and Hybrid Systems*.

[5] Stanley Bak and Parasara Sridhar Duggirala. 2017. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. ACM.

[6] Stanley Bak and Parasara Sridhar Duggirala. 2017. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

[7] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A Henzinger. 2017. Counterexample-guided refinement of template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 589–606.

[8] Nicole Chan and Sayan Mitra. 2017. Verifying safety of an autonomous spacecraft rendezvous mission. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*. EasyChair.

[9] Hongxu Chen, Sayan Mitra, and Guangyu Tian. 2014. Motor-Transmission Drive System: a Benchmark Example for Safety Verification.. In *ARCH@ CPSWeek*. 9–18.

[10] WH Clohessy. 1960. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences* 27, 9 (1960), 653–658.

[11] S Croomes. 2006. Overview of the DART mishap investigation results. *NASA Report* (2006), 1–10.

[12] Parasara Sridhar Duggirala and Mahesh Viswanathan. 2016. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*. Springer, 477–494.

[13] Parasara Sridhar Duggirala, Le Wang, Sayan Mitra, Mahesh Viswanathan, and César Muñoz. 2014. Temporal precedence checking for switched models and its application to a parallel landing protocol. In *International Symposium on Formal Methods*. 215–229.

[14] Goran Frehse. 2005. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC*. 258–273.

[15] Goran Frehse, Rajat Kateja, and Colas Le Guernic. 2013. Flowpipe approximation and clustering in space-time. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*. ACM, 203–212.

[16] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *International Conference on Computer Aided Verification*. Springer.

[17] Antoine Girard, Colas Le Guernic, and Oded Maler. 2006. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 257–271.

[18] Willem Hagemann. 2014. Reachability analysis of hybrid systems using symbolic orthogonal projections. In *International Conference on Computer Aided Verification*. Springer, 407–423.

[19] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. 2015. A formally verified hybrid system for the next-generation airborne collision avoidance system. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 21–36.

[20] Christopher Jewison and R Scott Erwin. 2016. A spacecraft benchmark problem for hybrid control and estimation. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*. 3300–3305.

[21] Taylor T Johnson, Jeremy Green, Sayan Mitra, Rachel Dudley, and Richard Scott Erwin. 2012. Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In *International Symposium on Formal Methods*. 252–266.

[22] Michal Kvasnica, Pascal Grieder, Mato Baotić, and Manfred Morari. 2004. Multi-parametric toolbox (MPT). In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 448–462.

[23] César Munoz, Anthony Narkawicz, and James Chamberlain. 2013. A TCAS-II resolution advisory detection algorithm. In *AIAA Guidance, Navigation, and Control (GNC) Conference*. 4622.

[24] Lucia Pallottino, Eric M Feron, and Antonio Bicchi. 2002. Conflict resolution problems for air traffic management systems solved with mixed integer programming. *IEEE transactions on intelligent transportation systems* 3, 1 (2002), 3–11.

[25] Pavithra Prabhakar, Vladimeros Vladimerou, Mahesh Viswanathan, and Geir E Dullerud. 2009. Verifying tolerant systems using polynomial approximations. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*. IEEE, 181–190.

[26] Stefan Schupp and Erika Ábrahám. 2018. Efficient dynamic error reduction for hybrid systems reachability analysis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 287–302.

[27] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. 2017. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*. Springer, 288–294.

[28] Claire Tomlin, George J Pappas, and Shankar Sastry. 1998. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on automatic control* 43, 4 (1998), 509–521.

[29] Yang Zhao and Kristin Yvonne Rozier. 2014. Formal specification and verification of a coordination protocol for an automated air traffic control system. *Science of Computer Programming* 96 (2014), 337–353.