Aggregation Strategies in Reachable Set Computation of Hybrid Systems: A Safe Satellite Rendezvous Case Study

ABSTRACT

Reachable set computation tools for hybrid systems approximate the reachable set over an interval of time with a symbolic representation. Most reachable set computation tools implement some form of aggregation for handling discrete transitions. Sometimes, the reachable set is insufficient for inferring the safety specification since the aggregation strategies lead to very conservative overapproximations. However, if such an aggregation is not performed, then one has to keep track of exponential number of symbolic representations and incur significant computation costs.

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 stragety is driven by counterexamples and hence has soundness and relative completeness guarantees. We apply our technique to infer safety properties of satellite rendezvous mission and demonstrate the computational benefits of these enhancements.

KEYWORDS

Hybrid Systems and Reachable Set and Linear Differential Equations and Aggregations for Reachable Set and Adaptive Deaggregation.

ACM Reference Format:

. 2019. Aggregation Strategies in Reachable Set Computation of Hybrid Systems: A Safe Satellite Rendezvous Case Study. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 with confidence that the systems functions in a safe manner. However, such extensive testing is not always possible. In the case of satellite maneuver protocols, it is impossible to create a controlled test bed on earth to test such systems. Moreover, once deployed, updates to the control algorithms on satellite maneuver protocols is very difficult, if not impossible. 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 [9, 14, 16, 18–22].

In this paper, we perform safety analysis of satellite rendezvous mission modeled as a hybrid automata. Our analysis relies on computing an artifact called *reachable set*. Given a set of initial configurations (often uncountable) Θ, the reachable set is the set of all possible configurations encountered by the system trajectories starting from Θ. Since the reachable set is also an uncountable set, we compute a symbolic representation of the reachable set. Often, symbolic representations of convex sets such as support functions [11], polytopes [10], zonotopes [12], etc. are used because operations such as linear transformation, intersection, and Minkowski sum can be easily performed over these representations.

However, such convex representations are at a significant disadvantage while performing mode switches. Consider the case of a satellite rendezvous mission that involves a mode switch from continue to abort. This mode switch can happen at any time in a given interval. Due to this non-determinism, the initial set of states in the abort mode is a convex overapproximation of the reachable set. Often, this overapproximation is too conservative and hence, such reachable set computation is not useful in determining whether all the behaviors of the systems are safe. While some approaches tried to avoid computing intersections, to compute the reachable set [2], one has to perform overapproximation, which often becomes very conservative.

This paper exclusively focuses on improving the accuracy of the reachable set by presenting template based and convex hull based aggregation strategies for discrete transitions in linear hybrid automata. We also present a data structure called Aggregated Directed Acyclic Graph (AGGDAG) and explain how aggdag helps us in implementing various deaggregation strategies. We also provide soundness and relative completeness guarantees of our reachable set computation algorithm. We implement our strategies in a tool called HyLAA [3] and analyze a satellite rendezvous mission. Analyzing this model is particularly difficult because of the nondeterminism in switching behavior. In the literature only restricted models of such switching behavior were analyzed owing to the overapproximation created by the state of the art tools.

2 PRELIMINARIES

States and vectors are 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.

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

Conference'17, July 2017, Washington, DC, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

Loc is a finite set of locations (also called modes).

$X \subseteq \mathbb{R}^n$ is the state space of the behaviors.

$Flow : Loc \rightarrow AffineDeq(X)$ assigns an affine differential equation $\dot{x} = A_l x + B_l$ for location l of the hybrid automaton.

$Inv : Loc \rightarrow 2^{\mathbb{R}^n}$ assigns an invariant set for each location of the hybrid automaton.

$Trans \subseteq Loc \times Loc$ is the set of discrete transitions.

$Guard : Trans \rightarrow 2^{\mathbb{R}^n}$ defines the set of states where a discrete transition is enabled.

For a linear hybrid automaton, the invariants and guards are given as a conjunction of linear constraints.

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 [4]. 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 $x_i = x_{i+1}$, $x_i \in Guard(a)$ 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.

2.1 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*. Reachable set computation using generalized star representation leverages the superposition property of the linear dynamical systems. We include the basic details of this representation and the reachable set computation technique in this paper for completeness. For additional details, the readers can refer to [4, 8].

Definition 6. A *generalized star* (or simply star) Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is called the *center*, $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vectors in $\mathbb{R}^n$ called the *basis vectors*, and $P : \mathbb{R}^n \rightarrow \{\top, \bot\}$ is a predicate. A generalized star Θ defines a subset of $\mathbb{R}^n$ as follows.

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

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.

```
input  : Initial Set: Θ = ⟨c, V, P⟩, time step: h, time bound: k · h
output : Reach(Θ) = Reach_0(Θ), ..., Reach_k(Θ)
for each i from 0 to k do
    c_i ← ρ(c, h, k)[i];
    for each v_j ∈ V do
        v'_j ← ρ(c + v_j, h, k)[i] − c_i;
    end
    V_i ← {v'_1, ..., v'_m};
    Reach_i(Θ) ← ⟨c_i, V_i, P⟩;
    Append Reach_i(Θ) to Reach(Θ);
end
return Reach(Θ);
```

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 $\Theta \triangleq \langle c, V, P\rangle$ with $V = \{v_1, v_2, \ldots, v_n\}$, we compute the reachable set for a linear dynamical system $\dot{x} = Ax + B$ using simulations. We generate simulations starting from c (denoted as $\rho(c, h, k)$), and $c + v_j$ $\forall 1 \leq j \leq n$ (denoted as $\rho(c + v_j, h, k)$). For a given time instance $i \cdot h$, the reachable set denoted as $Reach_i(\Theta)$ is defined as $\langle c_i, V_i, P\rangle$ where $c_i = \rho(c, h, k)[i]$ and $V_i = \langle v'_1, v'_2, \ldots, v'_n\rangle$ where $\forall 1 \leq j \leq n$, $v'_j = \rho(c + v_j, h, k)[i] - \rho(c, h, k)[i]$. Notice that the predicate does not change for the reachable set, but only the center and the basis vectors are changed.

Notice that Algorithm 1 can compute the reachable set of linear dynamical systems using $n + 1$ simulations. The reachable set computation technique for hybrid automata has two additional subroutines. First, it computes the overlap of the reachable set in each location with the location invariant. Second, it also computes the overlap of the reachable sets with the guards of discrete transitions that cause a change in location. When a discrete transition is performed, the reachable set in the new location 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 automaton H, Time bound $k \cdot h$.
output: *ReachSet* as the set of reachable states.
1 *queueStars* $\leftarrow \emptyset$; append Θ to *queueStars*; *ReachSet* $\leftarrow \emptyset$;
2 **while** *queueStars is not empty* **do**
3 $S \leftarrow$ dequeue(*queueStars*);
4 $R \leftarrow$ ReachableSetDynamicalSystem($S, S.loc$);
5 $R' \leftarrow$ InvariantOverlap($R, R.loc$);
6 *ReachSet* $\leftarrow$ *ReachSet* $\cup R'$;
7 *nextRegions* $\leftarrow$ discreteTrans($R', H.Trans$);
8 append *nextRegions* to *queueStars*;
9 **end**
10 **return** *ReachSet*;

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

3 AGGREGATION AND DEAGGREGATION STRATEGIES

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

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

There are two main drawbacks of this aggregation mechanism. First, the collection of sets $S_1, S_2, \ldots, S_m$ is often a non-convex set. Whereas the representation used for computing reachable set is for convex sets. Therefore, this overapproximation of a non-convex set by a convex set is very conservative. More worryingly, the reachable set of S' 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 above mentioned challenges, we develop new aggregation and deaggregation techniques. Our technique works the following way. 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 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.

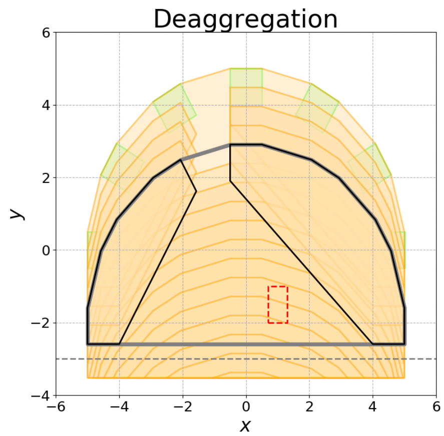

Figure 1: The deaggregation process is shown for a two-mode system. Upon reaching an error mode (red dotted region), the fully aggregated set of states (gray large region), is split in half (two black regions), which no longer contain error states. A video of the complete computation is available online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

One of the advantages of generalized stars is that it allows for easy aggregation and deaggregation. Additionally, leveraging the data structure of generalized stars, we avoid computing the entire reachable set, but only compute the specific sections of the reachable set that are useful for safety verification. To keep track of the computations, we maintain a data structure called aggregated directed acyclic graph (AGGDAG). The algorithm corresponding to the aggregation and deaggregation is given in Algorithm 3. A working example of the aggregation and deaggregation is provided in Figure 1.

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 11 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 2- 17 terminates after finite number of times. This is easy to infer as there are only finitely many reachable sets that we compute. Hence, we perform

input : Initial set Θ, Hybrid automaton H, Time bound $k \cdot h$, Unsafe set U.
output : If there is a trajectory starting from Θ and visiting U.
1 *queueStars* $\leftarrow \emptyset$; append Θ to *queueStars*; *ReachSet* $\leftarrow \emptyset$;
2 **while** *queueStars is not empty* **do**
3 $\quad S_{agg} \leftarrow$ aggregation(all aggregatable stars going to same mode in *queueStars*);
4 $\quad S_{agg}.tag \leftarrow$ aggregate; dequeue stars going to $S_{agg}.loc$ from *queueStars*;
5 $\quad R \leftarrow$ ReachableSetDynamicalSystem($S_{agg}, S_{agg}.loc$);
6 $\quad R' \leftarrow$ InvariantOverlap($R, R.loc$);
7 $\quad$ **if** $R' \cap U \neq \emptyset$ *and* $R'.tag =$ aggregate **then**
8 $\quad\quad$ Deaggregate at least one star in the path from S_{agg} to root;
9 $\quad\quad$ Enqueue the *queueStars* with the results of deaggregation;
10 $\quad$ **end**
11 $\quad$ **if** $R' \cap U \neq \emptyset$ *and* $R'.tag \neq$ aggregate **then**
12 $\quad\quad$ **return** There exists a trajectory from Θ visiting U;
13 $\quad$ **end**
14 $\quad$ *ReachSet* $\leftarrow$ *ReachSet* $\cup R'$;
15 $\quad$ *nextRegions* $\leftarrow$ discreteTrans($R', H.Trans$);
16 $\quad$ append *nextRegions* to *queueStars*;
17 **end**
18 **return** All trajectories are safe;

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

only finitely many aggregations. Since we strictly do not aggregate the stars that were deaggregated before, the condition in line 7 will only be encountered finite number of times. Hence the loop terminates and the algorithm either returns safe or unsafe. □

3.1 Aggregation and Deaggregation Using Generalized Stars

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

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.

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

Proof. Trivially follows from the definition of generalized stars. □

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 $(P_1 \vee P_2 \vee P_k) \Rightarrow P'$ 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$P_2(\alpha) = \top$. Similarly we 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**
$\quad$ **for** *each star* S_i **do**
$\quad\quad d_j^i \leftarrow \max c_j^T\alpha$ given $P_i(\alpha) = \top$;
$\quad$ **end**
$\quad d_j \leftarrow \max\{d_j^1, \ldots, d_j^m\}$;
end
return $P' \triangleq (c_1^T\alpha \leq d_1) \wedge (c_2^T\alpha \leq d_2) \wedge \ldots \wedge (c_l^T\alpha \leq d_l)$;

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

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

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. Without such cost reduction, template based overapproximation becomes very expensive. While the presentation here has restricted itself to only stars with same center and basis vectors (for the sake of simplicity), it is easy to observe that the template based aggregation can also be extended to stars with different centers and different basis vectors.

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 our experience, 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.

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) [17] 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 [13]. We include the basic details of this convex hull operation for the sake of completeness.

Definition 7. A symbolic orthogonal projection 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

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

If a polytope is represented as a generalized star, there is 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 [13]).

Definition 8. Given two symbolic orthogonal projections $_1 \triangleq (A_1, L_1, \mathbf{a_1})$ and $_2 \triangleq (A_2, L_2, \mathbf{a_2})$, the convex hull of $_1$ and $_2$ 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 $_1$ and $_2$ had $n + k$ variables and m constraints, then the number of constraints in $_3$ 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

When the overapproximation obtained from reachable set is too convervative and overlaps with the unsafe set, we perform deaggregation. In typical reachable set computation tools, one has to resume the computation of the reachable set from the newly deaggregated sets. However, we leverage the properties of generalized stars in reachable set computation and decrease the computations that need to be performed.

Remark 1 Consider an aggregated star $S_a \triangleq \langle c, V, P_a \rangle$ and the bounded time reachable set be $S_{a_1}, \ldots, S_{a_k}$ where $S_{a_i} \triangleq \langle c_i, V_i, P_a \rangle$. After performing deaggregation, S_a results in stars S_b and S_c where $S_b \triangleq \langle c, V, P_b \rangle$ and $S_c \triangleq \langle c, V, P_c \rangle$, then the reachable set starting

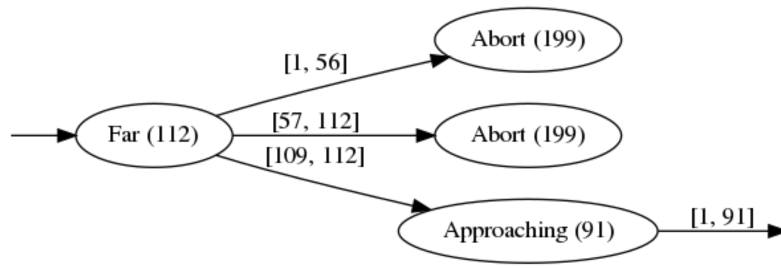

Figure 2: Example of an aggdag.

from S_b is given as $S_{b_1}, \ldots, S_{b_k}$ where $S_{b_i} \triangleq \langle c_i, V_i, P_b \rangle$. Similar relation holds for S_c as initial set. Therefore, one need not recompute the center and basis vectors for computing the reachable set with new initial set. Merely changing the predicate in the generalized stars suffices.

While recomputing the new center and basis vectors can be avoided, we also reduce our effort by checking intersection with guards and overlap with unsafe set after deaggregation. That is, when an aggregation is performed and the reachable set is computed, we only keep track of the stars that either overlap with the unsafe set or encounter a discrete transition. We keep track of the relationship between the reachable sets called as Aggregated Directed Acyclic Graph (AGGDAG).

Definition 9. An AGGDAG is a directed graph (G, H) where, the set of nodes G corresponds to the generalized stars obtained 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 generalized stars.

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 $[57, 112]$ and stays in the Abort mode without encountering any unsafe set. The successors of the stars that encounter a discrete transition to the Approaching mode in the time interval $[109, 112]$, encounter the unsafe set in future. In the Figure 2, the star representing the node Approaching is the collection of the stars that encounter the discrete transition in the time interval $[109, 112]$. Simiarly, the node Abort [Abort (90) to be more precise] corresponds to the collection of the stars that take the discrete transition from Approaching mode to the Abort mode. Out of the stars in the Abort(90) node, the violation of safety propery happens between the time intervals $[16, 86]$.

To check if the safety property is indeed violated in the reachable set computation, we first inspect the aggdag in Figure 2. Since the safety is violated in trajectories in Abort mode, we need to inspect whether there is overapproximation induced in the aggregation of the reachable set in the path from the root node to the Abort node. This overapproximation can be at two instances, first, the aggregation of stars in the Approaching modes 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

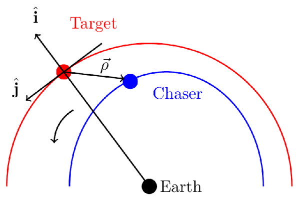

Figure 3: Collisions are checked between spacecraft in orbit in relative coordinates (image from [5]).

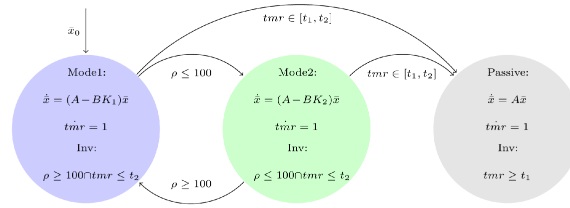

Figure 4: Hybrid automaton model of our case study (image from [5], which also include the dynamics matrices). We check for passive safety over the full time bound ($t_1 = 0$ and $t_2 = 200$).

violation. Hence, refining only the aggregation associated with the reachable sets in the specific interval of time suffices.

The aggdag is useful only in bookkeeping the states that encounter 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 STUDY: SPACECRAFT RENDEZVOUS PASSIVE SAFETY

We evaluate our method on a spacecraft rendezvous passive safety case study. The 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) [7].

Our model is based on a published benchmark for this system [5, 15]. 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 [6], which is often used in close proximity operations. 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 five-dimensional linear system, with nondeterministic transitions to the passive mode. In our analysis, we check for passive safety over the full analysis time bound, $t_1 = 0$ and $t_2 = 200$. The initial set of states, dynamics, and controller in each mode are described in the original benchmark proposal [5]. We focus on the collision-free safety property, that the spacecraft must remain 0.2 meters apart at all times.

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 not be known in advance.

The analysis done in the original work [5] is slightly better, in that it checks for passive safety for a known 5 minute failure interval during the 200 minute maneuver. The reason stated in the paper for this is 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 run several experiments. Presumably, a manual guess-and-check approach should be used to create these subintervals.

The full passive safety problem can be solved using the proposed aggdag method. Our aggdag method performs full state aggregation (an overapproximation), and then recursively desegregates if the overapproximation reaches an error mode. The advantage of this is that (1) the method is fully automatic, (2) steps where the overapproximation is safe can be skipped by the refined sets, which is more efficient than using multiple independent experiments, and (3) if an error exists, it will be detected after full deaggregation is performed, which allows the generation of a concrete counter example trace. Other verification tools for linear hybrid systems do not typically generate counter-examples when safety cannot be proven.

Our experiments are performed on a system with an Intel i5-5300U CPU running at 2.30GHz with 16GB RAM and using Ubuntu Linux 18.04. We first analyze the runtime of the method, shown in Table 1 on the left. We look at the number of seconds and the number of reachability steps needed to prove safety for the system as we vary the step size. A reachability step in this case is a single continuous post operation (safety check at a single multiple of the time step), or a refinement step when performing deaggregation. As the step size for this system is reduced, the number of combinations of steps that reach each guard in the hybrid automaton increases. However, from the table, we observe that the runtime and number of steps remains inversely proportional to the step

Table 1: Verification time for the safe case (left) and unsafe case (right).

Step Size	Runtime (s)	Num Steps	Step Size	Runtime (s)	Num Steps
1.0	5.1	726	1.0	9.2	1232
0.5	11.0	1508	0.5	34.8	3736
0.2	34.7	4657	0.2	94.7	10958
0.1	73.2	9557	0.1	243	25091

size. This means that the analysis is successfully using state set aggregation to eliminate combinatorial explosion, with sufficient precision to guarantee the system avoids collisions.

In our experiments, we observed that template based aggregation works much faster than convex hull based aggregation. In this case study, due to the high nondeterminism in the switching conditions, the convex hull representation becomes prohibitively large and checking safety properties with that convex hull aggregation becomes expensive. We conjecture that in instances where switching does not occur often or the nondeterminism involved with the switching is less, convex hull approximation could work better. Our deaggregation strategy is from leaves to root, i.e., we first deaggregate the states in the discrete transition closest to the occurrence of the overlap with the unsafe set.

A plot of the reachable state is shown in shown in Figure 5. The initial states are in the lower left corner in the `far` mode, $x \in [-925, -875]$, $y \in [-425, -375]$. Upon entering the set denoted by the dotted triangle, the system enters a different `approaching` mode. The unsafe set is shown as the red box near the origin, which is not reachable after multiple deaggregation steps are performed. A video of the computation and refinement process is available at https://youtu.be/iXJlJnsxeN0.

The analysis is exact, in that if the system were to have a collision, the deaggregation approach would eventually find it. In the next experiment, we increase the collision distance from 0.2 to 1.0 meters. In this case, a collision is possible, and our approach generates the corresponding counter-example trace (initial state and switching times) for every step size analyzed. The results are shown in Table 1 on the right. The runtime increases compared with the safe version of the benchmark, as more deaggregation is necessary in this case since a real error trace is present (the deaggregation continues until single time instants are considered, at which point a concrete trace can be generated).

Overall, our main evaluation result is that analysis of this system is possible by maintaining the aggdag data structure and performing deaggregation upon reaching an error mode. Prior to this, all analysis on this model checked for switching to the passive mode at a single time instant or small time window, since otherwise the methods would have too much error to prove the system is safe. For this reason, we could not perform a tool runtime comparison; analysis is not possible on this model with existing tools. Furthermore, we were able to generate counterexamples in the cases where the safety property was violated.

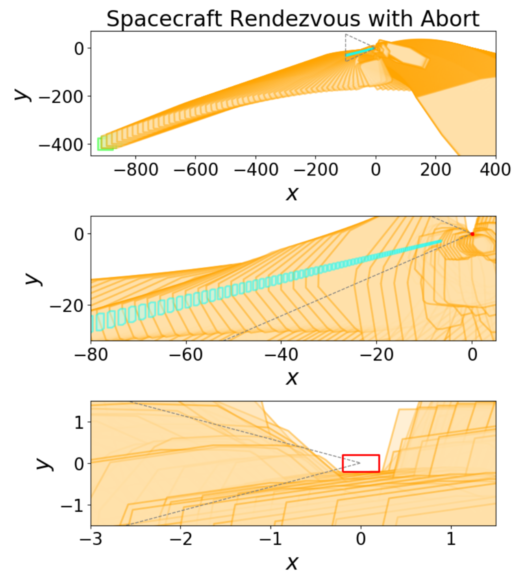

Figure 5: The reachable set for the spacecraft rendezvous system at three different zoom levels is shown. Reachable states near the unsafe set (red square near origin) are deaggregated using the proposed approach until no unsafe states are reachable. A video of the computation is online at https://youtu.be/iXJlJnsxeN0.

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 a challenging case study of satellite rendezvous mission and prove the passive safety property.

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.

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 (EPiC Series in Computing)*, Goran Frehse (Ed.), Vol. 54. EasyChair, 23–52. https://doi.org/10.29007/73mb

[2] Matthias Althoff and Bruce H Krogh. 2012. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*. ACM, 45–54.

[3] 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.

[4] 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.

[5] Nicole Chan and Sayan Mitra. 2017. Verifying safety of an autonomous spacecraft rendezvous mission. *arXiv preprint arXiv:1703.06930* (2017).

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

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

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

[9] 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*. Springer, 215–229.

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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*. Springer, 252–266.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.