Utilizing the Structure of Safety Properties to Aid in the Verification of Hybrid Controllers

Natasha Neogi
University of Illinois, Urbana-Champaign
Rm 317, Talbot Laboratory
104 South Wright Street
1-217-333-4741
Neogi@uiuc.edu

ABSTRACT

Identifying hazardous states in a system and ensuring they are not reachable is an ongoing problem in any large system safety analysis. An exhaustive search of most complex process-control systems will encounter the problem of state explosion. The use of predicate abstraction has been studied and employed to great effect. However, in this paper, we outline a novel approach which utilizes the structure of the safety property to be verified to mitigate state explosion. The state explosion problem is minimized by creating an abstraction that partitions the state space into equivalence classes based on the predicates of the safety property. An ordering on these predicates, based on the number of continuous variables inherent to each term, is used to develop the set of equivalence classes that minimizes the number of continuous variables that must be computed in order to determine if a transition between classes occurs. This acts to minimize the computational complexity of calculating reachability sets by limiting the number of continuous variables that need to be explicitly calculated. The work is then put in the context of an Air Traffic Conflict Detection example problem, and conclusions regarding scalability are drawn.

Categories and Subject Descriptors

Abstraction, Verification, Algorithms and Analysis Techniques, Reachability Computation

Keywords

Abstraction, Reachability, Software Safety, Formal Methods,

1. INTRODUCTION

In today's modern world, complex systems dominate the landscape. These systems, such as the flight management system (FMS) of an airplane, possess enormous state spaces, which are nearly impossible to test and explore in search of unsafe or *hazardous* states. A hazardous state is a state of a system that, together with other environmental conditions, leads to an accident. The identification of these hazardous states in software design, and their eventual elimination from the design by ensuring they are not reachable from the initial states, is a recurring problem in the design of safe software systems [14].

Hazard analysis is at the heart of any effective safety program. Simply knowing that a hazard exists may provide sufficient information to eliminate or control it, even without in-depth analyses of its causes. For many hazards and systems, analysis may consist of comparing the design with various standards and codes that have been developed over time to deal with known hazards. However, as new technology is developed and system size increases dramatically, such as in aerospace systems that may have millions of states, new hazards arise and the possibility of introducing hazards increases. The ability to trace a hazard back to its initiating state in order to provide options for elimination or control becomes extremely computationally intensive, if not infeasible.

A reasonable method to evaluate whether a hazard could be eliminated might entail starting with the hazardous state and working backward in a model of the system to see if the initial state is reached. If the initial state is reached, then the hazardous state is reachable and must be eliminated or controlled in order for the system to operate safely. However, the number of backwards paths is enormous for most real systems, even if those ending in only hazardous states is considered. A method to circumvent the effects of the *state explosion* problem becomes essential in order to pursue any form of effective hazard analysis in any complex, real system.

One approach to mitigate the state explosion problem is the method of predicate abstraction. An infinite state space can be partitioned based on Boolean predicates, resulting in a finite representation of an infinite space. The state-space is partitioned into finitely many equivalence classes so that states in the equivalence classes exhibit similar behavior with respect to the predicate used to create the abstraction. Reachability analysis can then be applied to the equivalence class itself, rather than upon each individual state in the class [1]. However, the choice of the Boolean predicates used to create the abstraction greatly influences the efficiency of the verification technique. The calculation of the minimal set of continuous variables necessary in order to determine the satisfaction of the predicates is essential in order to enable a speedup of the verification process.

In the next section, a brief overview of the different techniques used to mitigate the state explosion problem is presented, and the reasoning behind utilizing abstraction is highlighted. In Section 3, the theoretical basis for creating a finite abstraction of an infinite state space is outlined, and an approach for developing the most advantageous abstraction, in order to minimize computational complexity is proposed. In Section 4, a resume of different techniques for performing

reachability analysis in hybrid systems is presented. Refinements are made to the equational logic of the abstraction that illustrate how the inherent structure of the problem and constraints can be used to help simplify the computation of the reachability set of the system. Section 5 explains the Aircraft conflict detection example, and how the particular abstraction chosen enables the elimination of a specific hazard. Conclusions are drawn in the final section as to the scalability and practicality of this approach.

2. STATE EXPLOSION AND REACHABILITY

The problem of verification reduces to that of reachability. Determining whether or not a system is verifiably safe with respect to a given safety constraint becomes the task of proving that, for all states of the system, the safety constraint holds. This task amounts to proving that, from all initial states of the system, all hazards corresponding to the violation of the safety constraint are absent from the reachability graph of the system. The verification of systems that possess a large number of components that interact in a complex fashion with one another is problematic. The asynchronous interaction of components and the use of data structures that can assume many different values leads to an enormous global state space. If the system being modeled contains both discrete and continuous components, the state space is infinite. It then becomes impossible to check whether the system possesses any hazardous states by simple enumeration alone. Several techniques, such as partial order reduction, compositional reasoning, symmetry, induction and abstraction can be used to mitigate the state explosion problem, and help to efficiently generate the reachable space of the system to be verified.

One of the most successful techniques for dealing with the state explosion problem is based on partial order reduction [10,23]. This technique exploits the independence of concurrently executed events. Two events are independent of each other when executing them in either order results in the same global state. The most common model for representing concurrent software is the interleaving model, in which all of the events in a single execution are arranged in a linear order called an interleaving sequence [22]. However, the initial model only considered a restricted model of concurrency that did not include looping and nondeterministic choice. The proof system of Katz and Peled [12] suggests using an equivalence relation between interleaving sequences that correspond to the same partially ordered execution. Their system included proof rules for reasoning about a selection of interleaved sequences rather than all of them. When a specification cannot distinguish between two interleaving sequences that differ only by the order in which concurrently executed events are taken, it is sufficient to analyze only one of them. As a result, the number of states that are needed for model checking is reduced [27].

Compositional reasoning exploits the modular structure of complex protocols [5]. Many finite state systems are composed of multiple processes running in parallel. The specifications for such systems can be decomposed into properties that describe the behaviour of small parts of the system. An obvious strategy is to check each of the local properties, using only the part of the system that the property describes. If it is possible to show that the system satisfies each local property, and if the conjunction of the local properties implies the overall specification, then the complete system must satisfy this specification as well. If there are interdependencies in the components, a form of assume-guarantee reasoning can be employed. When proving a property about one component, assumptions are made about the behaviour of all the other components. The assumptions must then be proved when the correctness of the other components is established [11].

Symmetry can also be used to reduce the state explosion problem [4]. Finite state concurrent systems frequently contain replicated components or structures. Having symmetry in a system implies the existence of a non-trivial permutation group that preserves the state transition graph. Such a group can be used to define an equivalence relation on the state space of the system and to reduce the state space. The reduced model can be used to simplify the verification of the properties of the original model express by a temporal logic formula.

Induction involves reasoning automatically about entire families of finite-state systems [6]. Such families can arise in the design of reactive systems in software, as well as hardware. A process control system can be parameterized, defining an infinite family of systems. The goal is to prove that every system in a given family satisfies some temporal logic property. In general the problem is undecidable, but it is possible to provide a form of invariant process that represents the behavior of an arbitrary member of the family. Using the invariant, the property can be checked for all members of the family at once. An inductive argument is then used to verify that the invariant is an appropriate representative.

Finally, the technique employed to the greatest advantage is called abstraction [5]. This technique appears to be essential for reasoning about reactive systems that involve data paths. The use of abstraction is based on the observation that the specifications of systems that include data paths usually involve fairly simple relationships among the data values in the system. The abstraction is usually specified by giving a mapping between the actual data values in a system and a small set of abstract data values. By extending the mapping to states and transitions, it is possible to produce an abstract version of the system under consideration. The abstract system is often much smaller than the actual system, and as a result it is usually much simpler to verify properties at the abstract level. Extending this method to systems which possess continuous dynamics, and thus infinite state spaces, allows for a finite abstraction of the infinite state space, and thus reduces the number of reachable states which need to be generated, thereby rendering the verification problem more tractable.

Thus, it seems plausible, that by a combination of clever modeling techniques, and assiduously chosen abstracted state variables, it is possible to generate an algorithm that would be able to check a given design, be it software or hardware, for the presence of identifiable hazards.

3. FINITE ABSTRACTIONS OF INFINITE STATE SYSTEMS

Discrete and hybrid systems, which possess a large (possibly infinite) number of states are difficult to verify. Consider a system that may possess infinitely many states (such as a hybrid automaton), and a set of *k* safety constraints (which may be temporal in nature) whose verity must be ascertained for each state of the labeled transition system. Obviously, employing the technique of enumerating each state of the system, then evaluating each of the constraints for that state will not enable you to successfully verify that the system is safe with respect to the constraints. However, if the system can be expressed as a labeled transition system, and the safety constraints can be expressed as a finite number of logical predicates (or temporal logic formulae) then a finite abstraction of the infinite state space can be derived [1].

More formally, given the labeled transition system T, where:

$$T = (Q, I, A, \rightarrow) \tag{1}$$

and Q is the set of states (possibly infinite) of the system, I is the set of initial states of the system, A is the set of actions which label transitions between states, and $\rightarrow$ is the set of labeled transitions of the form:

$$q \xrightarrow{a} q', \quad q, q' \in Q, \ a \in A \tag{2}$$

we consider an equivalence relation $\cong$ on Q, which results in a finite partitioning of Q. The quotient ($T/\cong$) is a labeled transition system whereby:

$$\left(T/\cong\right) = \left(P, H, A, \rightarrow'\right) \tag{3}$$

and states $p \in P$ are equivalence classes of T, a state p is initial so that $p \in H$ if p contains a state in I, the set of actions A which label transitions between states in P, and $\rightarrow'$ is the set of labeled transitions of the form:

$$p \xrightarrow{a} p' \Leftrightarrow q \xrightarrow{a} q' \quad \text{for some } q \in p, q' \in p' \tag{4}$$

It is immediately obvious from this definition that if a property is true for all states P in the labeled transition system ($T/\cong$), then the property is true for all states Q in the labeled transition system T [1].

It only remains to define the nature of the equivalence relation ($\cong$). While any relation that preserves the above properties will reduce the size of the (possibly infinite) state space, certain equivalence relations shall prove more advantageous than others.

3.1 Defining the Equivalence Relation

The equivalence relation is a method by which the state space Q of the labeled transition system T is partitioned into finitely many equivalence classes [1]. More formally, consider the state space Q of the labeled transition system T being partitioned into finitely many equivalence classes using k Boolean predicates $\varphi_1, \varphi_2, \ldots \varphi_k$. If the labeled transition system T possesses a state space Q of dimension $L \times R^n$, then the quotient labeled transition system ($T/\cong$) will possess a state space of dimension $L \times \{0,1\}^k$ (see Fig. 1).

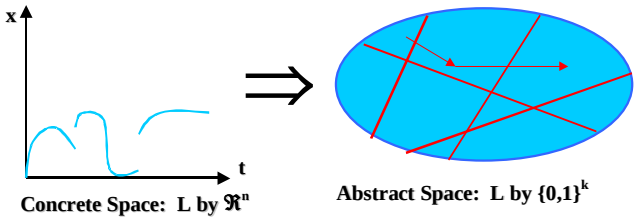

Figure 1:Transformation of State Space under Abstraction

Two fundamental questions arise concerning the nature of the predicates chosen to create the abstraction of the original state space:

1. How does one reduce the number of equivalence classes necessary to represent the state space of the system, while still keeping track of all pertinent behaviour?
2. What is the best way of selecting the abstraction in order to aid in computational efficiency of generating the reachability set of the abstracted system in order to verify the given safety constraints?

We propose that the minimum number of equivalence classes necessary to allow for the verification of a set of safety constraints, which can be represented by a finite number of Boolean predicates, can be derived using some minimal combination of those Boolean predicates (and possibly some additional completing predicates) used to express the safety constraint, in order to create the abstraction.

Consider the case where the Boolean predicates $\varphi_1, \varphi_2, \ldots \varphi_m$, (each of which is a function of quantified continuous variables) along with logical operators and inference rules can be used to construct the safety constraints of the system to be verified. Furthermore, we can augment these predicates in order to form a complete and confluent set Φ (all the behaviour is explicitly specified with no obvious contradiction arising from the conjunction of the predicates) whereby:

$$\Phi = \{\varphi_1, \varphi_2, \ldots \varphi_m, \varphi_{m+1}, \ldots, \varphi_n\} \tag{5}$$

Hence, if we consider the system of minimal logical predicates to be a set wherein each predicate cannot be simplified further or eliminated through logical deduction, inference rules or equational rewriting, we can achieve a minimal set of predicates $\Phi_{\min}$ such that:

$$\Phi \mapsto \Phi_{\min} \tag{6}$$

that is, the original set can be reduced to the minimal set by zero or more rewriting steps. The truth of each predicate $\varphi_i \in \Phi_{\min}$ cannot be determined through logical operation on the set $\{\Phi_{\min} - \varphi_i\}$. This set of minimal predicates, $\Phi_{\min}$ is the set we wish to employ to create the equivalence classes of the original transition system T.

4. REACHABILITY ANALYSIS AND PREDICATE ABSTRACTION

Given our set of predicates $\Phi_{\min}$ derived from the safety properties of the system, we can begin to construct the abstraction of the state space of the original system. However, the overall goal is to be able to verify that all of the safety constraints are satisfied over the entire state space of the system. We must still generate the reachable space of the abstracted system, in order to check that all of the constraints are satisfied. There are many techniques for generating the reachable space of a system. For instance, in very large discrete systems, ordered binary decision diagrams have been employed to great effect, as they are a compact representation of the state space which allows for efficient search techniques to be employed [3]. For systems with both continuous and discrete dynamics (hybrid systems) where the state space is infinite, there are several methods used, which are outlined below.

For time invariant, state independent dynamics specified by a convex polytope constraining the rates, the dynamics of the system can be abstracted by using differential inclusions that bound the rates at which the system can evolve [1]. However, this acts to greatly restrict the types of systems that can be analyzed considerably.

Another method for reachability computation, called face-lifting for differential equations has been employed with some success [8]. Unlike other approaches that attempt to give exact answers, this approach is based on numerical approximation and a combination of techniques taken from discrete verification, computational geometry and optimization. This method could theoretically work with arbitrary continuous dynamics ($dx/dt=f(x)$ where f is Lipschitz) but its performance is rather limited, in that systems of no greater than 4 dimensions can successfully be solved. This approach can be specialized to work with systems with linear differential equations ($dx/dt=Ax$) and generalized to include the analysis of systems with uncontrolled inputs ($dx/dt=Ax+Bu$) and to the problem of synthesizing switching controllers. A tool with a reasonable performance has been implemented and is currently under testing with examples taken from traffic control, engine control, robotics and chemical process control. For the purpose of representing the set of reachable states, a new representation scheme for orthogonal polyhedra has been invented. It should be noted that this representation is canonical (unique) for all (convex and non-convex) polyhedra in any dimension.

Another formulation for computing the reachability of a system with both continuous and discrete dynamics requires the solution of a Hamilton-Jacobi partial differential equation, and a grid-based numerical solution approach based on level set methods can be used for this purpose [26]. The continuous-time nonlinear dynamics of the system need to be considered carefully in assessing reachability. Simulation examples exploiting this technique have been presented for three flight management applications: two-aircraft collision avoidance, the related problem of conflict resolution, and ensuring safety during final landing approach. The exact reachability computation falls prey to the curse of dimensionality: its computational complexity is exponential with respect to the continuous dimension. Tomlin et al. [26] also present an alternative approach, which is based on over-approximating the reachable set of states with a polyhedron. This is also computationally intractable since the propagation of the system's dynamics will result in a potentially unlimited number of constraints (faces of the polyhedron), but a novel technique for identifying and pruning redundant and irrelevant constraints has been developed in conjunction with this work to mitigate the number of constraints. This technique appears to show promise for higher dimensional problems. The basis of the approach is the computation of the maximum volume ellipsoid contained in a polyhedron, a computation that can be formulated as a convex optimization problem.

The method of calculating the reachability of a system with both continuous and discrete dynamics by using ellipsoidal over-approximations to reach tubes, developed by Kurzhanski and Varaiya, [13] is promising for system dynamics which have a hard bound on controls and are essentially linearizable under small disturbances. The approach of ellipsoidal over-approximation uses the intersection of a given family of hyper-ellipsiods to approximate the reachable space of the dynamic system. The reachability set of the continuous equations can be regarded as being the tube consisting of all possible system trajectories. The evolution of the boundary of the reachability set can be approximated by the solution of the "integral funnel" of a differential inclusion [13,21], and the family of hyperplanes tangential to the boundary can be parameterized. A series of hyper-ellipsoids can be generated by picking two supporting hyperplanes, and generating a hyperellipsoid which circumscribes the reachability set and is tangential to its boundary at the hyperplanes. The advantage in this method comes from being able to create the parameterization of hyperplanes in terms of a system trajectory which runs along the boundary of the reachability set. This ensures a tight external approximation, thereby reducing the number of hyper-ellipsoids necessary to approximate the reachability tube. This method is employed for the aircraft conflict detection example, as it is a relatively simple process to get a system trajectory tangential to the reachability tube boundary, given the hard bounds on controls.

4.1 Using the Predicate Abstraction to Reduce the Computational Complexity of Reachability Calculation

Even for a system with several continuous variables, it becomes computationally challenging to generate the reachable space of the system. Calculating the value of all of the continuous variables for the reachable space creates an enormous burden, and any technique to reduce the number of variables to be calculated comes at a great computational savings. Now, if we consider the previous section, whereby we created the quotient ($T/\cong$) labeled transition system with the state space P of equivalence classes of the system T, based on the logical predicates arising from the safety constraints which needed to be verified, another fundamental question arises:

1. What is the minimum number of continuous variables which must be computed in order to ascertain whether or not any of the safety constraints have been violated?

This question can also be answered by looking to the predicates used to create the abstraction. If we consider all of the outgoing transitions from a given equivalence class p_i, the set of successor equivalence classes S_p is defined as:

$$S_p = \{p' \,|\, p \xrightarrow{a} p', \text{for some } a \in A, p, p' \in P\} \quad (7)$$

Recall that the equivalence classes P were formed by partitioning the state space Q of the original system using the Boolean predicates in $\Phi_{\min}$. That means, for a given class p_i, each Boolean predicate in $\Phi_{\min}$ evaluates to either true or false. Similarly, for each successor predicate p', each Boolean predicate assumes a truth value. We wish to consider *only* the predicates φ_j that change their value as an outgoing transition is taken. The validity of these predicates depends on the values of some subset of the continuous variables of the system. If we take the conjunction of all of the continuous variables employed in determining the truth value of each φ_j, we arrive at a minimal set of continuous variables V_{min} which must be evaluated in order to determine whether or not any outgoing transitions can be taken. This minimal set changes based on the set S_p, and must be re-evaluated every time a discrete action is taken. A great computational savings can be gained if it is possible to adjust the minimal set of predicates used to create the abstraction such that the successor classes to a given equivalence class depends on the change in truth value of a small number of predicates that depend on the smallest possible number of continuous variables.

The minimal set of predicates is isomorphic under all rules of logical implication, and can be restructured to create a basis minimal set of predicates Φ_{basis}. Each predicate φ_i in $\Phi_{\min}$ can be expressed as a function of some continuous variables. Through equational substitution and algebraic simplification, we can find $V = \{v_1, v_2, \ldots, v_m\}$, the minimum number of continuous variables necessary to express the all of the predicates in $\Phi_{\min}$. Let us introduce the notation $n(V)$ where:

$$n(V) = \text{number of elements in the set } V \quad (8)$$

We then select a predicate $\varphi_i \in \Phi_{\min}$ that is the function of continuous variables $V_i = \{v_k, v_l, \ldots\},\ k, l \in \text{Integers}$ such that:

$$\neg\exists \varphi_j \,|\, n(V_j) < n(V_i), \varphi_j \in \Phi_{\min} \quad (9)$$

that is, there is no $\varphi_j \in \Phi_{\min}$ which depends on fewer continuous variables. This φ_i becomes the first predicate φ'_1 in Φ_{basis}, with $V'_1 = V_i$. The next predicate φ'_2 is constructed by selecting a predicate φ_j such that:

$$\forall \varphi_k \in \Phi_{\min} \,|\, n(V_k \cap V'_1) < n(V_j \cap V'_1), k \neq i, j \quad (10)$$

that is, no predicate in $\Phi_{\min}$ has more continuous variables in common with φ'_1, and then creating:

$$\varphi'_2 = \varphi_2 \vee \varphi'_1 \quad (11)$$

In a similar manner, the p+1th predicate that forms Φ_{basis} can be created by selecting a $\varphi_k \in \Phi_{\min}$ based on:

$$\forall \varphi_l \in \Phi_{\min} \,|\, n(V_l \cap V'_p) < n(V_l \cap V'_p), l \neq i, \ldots k \quad (12)$$

and using the recursive formula:

$$\varphi'_{p+1} = \varphi_k \vee \varphi'_p \quad (13)$$

Thus, the satisfaction of any φ'_I implies the validity of all previous φ'_j for $i>j$. Thereby each predicate φ'_I possesses an explicit portion which is independent of all other $\varphi'_j \in \Phi_{basis}$. This creates a partial ordering on the predicates in Φ_{basis} such that:

$$\varphi'_1 < \varphi'_2 < \ldots \varphi'_{p+1} \quad (14)$$

If the equivalence classes P of are created using the predicates in Φ_{basis}, then each transition between equivalence classes explicitly states which continuous variables must change their value in order for the predicate to change its truth value. This allows for the explicit enumeration of the minimum number of continuous variables that define the transitions between equivalence classes.

5. AIRCRAFT CONFLICT DETECTION EXAMPLE

In order to demonstrate the utility of this method of abstraction, a scaled down version of a Medium Term Conflict Detection (MTCD) algorithm is examined, and the hazard of a missed detection due to failure to detect a trajectory overlap as a consequence of incomplete predicate specification is eliminated.

MTCD is a conflict detection algorithm under development to support Air Traffic Controllers (ATCOs) in their task of monitoring and separating aircraft. Therefore, MTCD must provide controllers with enough time to assess, and, if necessary, resolve the conflict by deliberate action [21].

MTCD supports conflict detection for all flights for which a system trajectory is available. MTCD begins conflict detection for a flight when it is a pre-defined time from entering the area of operation, and continues conflict detection until the flight leaves the area entirely. We shall consider a scaled down version of MTCD that detects loss of separation between probable positions of two aircraft, based on system trajectories and uncertainty areas, the latter introduced to take minor deviations into account

MTCD is a planning tool with a typical detection horizon of zero to twenty minutes for aircraft conflicts, twenty to sixty minutes for nominal route overlaps, and zero to sixty minutes for special use airspace penetrations and descents below lowest usable flight level. MTCD is not a conflict alert tool. Conflict alert, with a typical horizon of zero to two minutes, is covered a by separate function, called Safety Nets. MTCD calculations are based on system trajectories of flights, flight plan data and aircraft data. This data is provided by the Real-Time Flight Data Processing and Distribution function. Trajectories can be either system trajectories or tentative trajectories. To be able to end existing conflicts, Real-Time Flight Data Processing and Distribution must tell MTCD when a flight leaves the area of operation, or when a tentative trajectory has been deleted. In addition to

trajectory data, MTCD requires environment data. The data required by MTCD is provided by the Environment Data Processing and Distribution function [21].

In principle, MTCD is quite simple. The traffic and its evolution are specified by a set of trajectories. All that needs to be done is to examine the trajectories in pairs and report whenever trajectories come too close to each other. Complications occur because of model uncertainties in aircraft behavior and slow response to trajectory updates. By postulating the existence of elliptical uncertainty buffers between aircraft trajectories that include the separation standard, it is merely necessary to check for the overlap of uncertainty buffers in order to determine whether or not a conflict is imminent. Let us consider the matter of vertical separation. The vertical separation standard is 1000 feet below an altitude of 2900 ft, and 2000 ft above that altitude. If we consider the differential position of two planes to be $\Delta z = z_2 - z_1$, where z_i is the altitude of the plane, then the planes are in conflict if $\Delta z \le d$, where d is the separation standard for the flight level. The following predicates are used in the formal document to capture the explicit semantics of the safety constraint [21]:

$$\begin{aligned} \varphi_1 &: \Delta z \le 2000 \wedge (z_1 > 2900 \wedge z_2 > 2900) \\ \varphi_2 &: \Delta z \le 1000 \wedge (z_1 < 2900 \wedge z_2 < 2900) \end{aligned} \quad (15)$$

Note that these predicates form do not form a complete and consistent set Φ_{min}. In order for this to occur, we must add an additional two predicates:

$$\begin{aligned} \varphi_3 &: \Delta z \le 2000 \wedge (z_1 < 2900 \wedge z_2 > 2900) \\ \varphi_4 &: \Delta z \le 2000 \wedge (z_1 > 2900 \wedge z_2 < 2900) \end{aligned} \quad (16)$$

These predicates form a complete and consistent set. If any of these predicates are true, then the system is in conflict. Using equational rewriting, we can simplify these four predicates into the minimal set Φ_{min} (which in this case is trivially equal to the basis set):

$$\begin{aligned} \varphi'_1 &: \Delta z \le 1000 \\ \varphi'_2 &: \Delta z \le 2000 \wedge (z_1 > 2900 \vee z_2 > 2900) \end{aligned} \quad (17)$$

The basis set can be found according to equations (12-13), and is equal to the minimal set in this case. Thus, when we are calculating the reachability sets, at each enabled transition for the equivalence classes, we must explicitly enumerate the values of z_1 and z_2. Implicitly, the values of pitch, pitch-rate, horizontal (x) velocity and acceleration, as well as vertical (z) velocity and acceleration may need to be calculated, but no logical comparisions need to be carried out with their values. Since any equivalence class can only transition to three other possible equivalence classes, the computational complexity of calculating the reachability set for the longitudinal system becomes much more tractable.

These boundary conditions, for a bounded control input, lends itself to the selection of a system trajectory, which can be used to generate the hyperplane of support in order to create the ellipsoidal approximation to the continuous variables (such as pitch, pitch-rate etc.) in the reachability tube. Note that these variables do not have to be explicitly enumerated, only approximated, as it is their influence on the altitude variable z which is of interest. Note that the longitudinal equations of motion for aircraft are linear under small disturbance theory, which enables for a great deal of simplification in the method of ellipsoidal over-approximation, thus rendering the problem of generating the reachability set of six continuous variables (x and z velocity, pitch, pitch rate, elevator angle and thrust) tractable, as only the z-velocity variable must be explicitly integrated in order to perform logical comparisons. The leapfrog integration scheme was used to perform time integration, and a Recursive-Subdivision method to generate a fast, adaptive mesh scheme in order to calculate the spatial integrations [21,25]. Using this approach, the reachable space of the simplified model was generated, until the following violation was found in the safety constraint.

Consider the situation where two aircraft are flying below 2900 feet (one at an altitude of 2800 ft and another at an altitude of 1300 ft) and have 1500 ft of vertical separation. These aircraft are obviously not in conflict. Now, consider the topmost aircraft ascends to 3000 feet. The two aircraft are now in conflict. However, this would yield a missed detection by the algorithm with the safety constraints as initially specified. Violation of the constraint φ'_2 occurs, but not of φ_2. Therefore, the hazard of a missed detection at ascent above 2900 ft is detected by employing the outlined method of predicate abstraction, in conjunction with ellipsoidal over-approximation of the reachability sets of the equivalence classes.

6. CONCLUSIONS

Verification and validation of large complex systems is very difficult. The state spaces of such systems may be very large (infinite) in nature, and thus are impossible to enumerate. Generating the entire reachable space of the system in order to check that each state satisfies a given constraint (safety, liveness etc.) which needs to be validated becomes intractable. Instead, a method of predicate abstraction is proposed, in order to create a finite abstraction of an infinite state space. If the abstraction is created based on a minimal set of predicates which completely and consistently quantify the constraints to be verified, then the problem of generating approximations to the reachable space of the continuous variables in the abstraction can be greatly simplified. This is illustrated using a simplified example taken from Air Traffic Conflict Detection. A simplified vertical separation problem was addressed, and the hazard of a missed detection due to incomplete specification was identified. Further computational savings can possibly be achieved by converting the minimal set of predicates into a basis set as outlined. This method shows great promise for situations in which the underlying dynamics of the problem, and symmetries in the constraints can be exploited in order to create a small number of equivalence classes, each of which have a small number of successor classes, and depend explicitly upon only a limited number of continuous variables.

7. ACKNOWLEDGEMENTS

This research was partially supported by the Design for Safety program at NASA Ames.

8. REFERENCES

[1] Alur, A., Dang, T., and F. Ivanfii. Reachability analysis of hy- brid systems via predicate abstraction. In C. Tomlin and M.R. Greenstreet, editors, Hybrid Systems: Computation and Control, Fifth International Workshop, LNCS 2289, pages 35-48. Springer-Verlag, March 2002.

[2] Bensalem, S., Bouajjani, A., Loiseaux, C. and Sifakis, J. Property Preserving Simulations. Workshop on Computer Aided Verification. Fourth International Workshop. CAV'92. Proceedings LNCS 663. Springer 1992. pp. 260-273.

[3] Browne, M.C., Clarke, E.M. and Dill, D.L. Automatic Circuit Verification using Temporal Logic: Two New Examples. Formal Aspects of VLSI Design. Elsevier 1986.

[4] Clarke, E.M., Filkorn, T. and Jha, S. Exploiting Symmetry in Temporal Logic Model Checking. Proceedings of the 5th Workshop on Computer Aided Verification. June/July 1993. pp. 450-462.

[5] Clarke, E.M., Long, D.E. and McMillan, K.L. A Language for Compositional Specification and Verification of Finite State Hardware Controllers. Proceedings of the 9th International Symposium of Computer Hardware Description Languages and Their Applications. North Holland 1989. pp. 281-295.

[6] Clarke, E. M., Grumberg, O. and Peled, D. Model Checking. MIT Press 2001.

[7] Clarke, E.M., Long, D.E. and McMillan, K.L. A Language for Compositional Specification and Verification of Finite State Hardware Controllers. Proceedings of the 9th International Symposium of Computer Hardware Description Languages and Their Applications. North Holland 1989. pp. 281-295.

[8] Dang T., and Maler, O., Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, Hybrid Systems: Computation and Control, volume 1386 of LNCS, pages 96--109. Springer, April 1998.

[9] Dahleh, M. Introduction to Linear State Space Control Theory. Course Notes. Massachusetts Institute of Technolgy. Fall 1999.

[10] Goefroid, P. and Pirottin, D. Refining Dependencies Improves Partial Order Verification Methods. In Proceedings of the 5th Conference on Computer Aided Verification. LNCS 531. Springer 1990. pp. 438-449.

[11] Grumberg, O. and Long, D.E., Model Checking and Modular Verification. ACM Transactions on Programming Languages and Systems 16:843-872.

[12] Katz, S, and Peled, D., An efficient Verification Method for Parallel and Distributed Programs. In Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, pp. 489-507. Springer 1998.

[13] Kurzhanski, A.B. and Varaiya, P. Ellipsoidal techniques for reachability analysis. In N. Lynch and B. Krogh, editors, Hybrid Systems: Computation and Control (HSCC'00), LNCS 1790, pages 203--213. Springer-Verlag, 2000.

[14] Leveson, N.G. Safeware: System Safety and Computers. Addison-Wesley 1995.

[15] Leveson, N.G., Heimdahl, M.P.E. and Reese, J.D. Designing Specifications Languages for Process Control Systems. Foundations of Software Engineering. Toulouse. Sept. 1999.

[16] Leveson, N.G. and Stolzy, J.L. Safety Analysis Using Petri Nets. IEEE Transactions on Software Engineering. SE-13(3):386-397. March 1987.

[17] Manna, Z. and Pnueli, A. Temporal Verifications of Reactive Systems—Safety. Springer 1995.

[18] Meseguer. J., Research Directions in Rewriting Logic, Computational Logic, NATO Advanced Study Institute Series F, Vol. 165, pp. 345-398, Springer-Verlag, 1999.

[19] Miller, S. Modelling Software Requirements for Embedded Systems. Altitude Switch Specification. Rockwell Collins.

[20] Modugno, F., Leveson, N. G., Reese, J. D., Partridge, K. and Sandys, S.D. Integrated Safety Analysis of Requirements Specifications. IEEE 1997. pp. 148-159.

[21] Neogi, N. Hazard Elimination Using Backwards Reachability and Hybrid Modelling Techniques. Ph.D Thesis. Massachusetts Institute of Technology. 2001.

[22] Overman, W.T., Verification of Concurrent Systems: Function and Timing. PhD Thesis, University of California at Los Angeles, 1981.

[23] Peled, D., Combining Partial Order Reductions with On-the-Fly Model Checking, Proceedings of the 1994 Workshop on Computer-Aided Verification Methods for Finite State Systems, LNCS 818. Springer 1994, pp. 377-390.

[24] Rajan, S., Shankar, N., and Srivas, M.K. An Integration of Model Checking with Automated Proof Checking. Proceedings of the 1995 Workshop on Computer Aided Verification. LNCS 939. Springer 1995. pp. 84-97.

[25] Ribbens, Calvin J., A Fast Adaptive Grid Scheme for Elliptic Partial Differential Equations, ACM Transactions on Mathematical Software (TOMS), Volume 15 Issue 3, September 1989.

[26] Tomlin, C., Mitchell, I., and Ghosh. R., Safety Verification of Conflict Resolution Maneuvers, IEEE Transactions on Intelligent Transportation Systems, Volume 2, Number 2, June 2001.

[27] Valmari, A, A Stubborn Attack on State Explosion. In Proceedings of the 16th International Colloquium on Automata, Languages and Programming, LNCS 372. pp. 761-772. Springer, 1989.