Conversion and Verification Procedure for Goal-Based Control Programs

Julia M. B. Braman and Richard M. Murray [*]

August 15, 2007

Abstract

Fault tolerance and safety verification of control systems are essential for the success of autonomous robotic systems. A control architecture called Mission Data System, developed at the Jet Propulsion Laboratory, takes a goal-based control approach. In this paper, a method for converting goal network control programs into linear hybrid systems is developed. The linear hybrid system can then be verified for safety in the presence of failures using existing symbolic model checkers. An example task is developed and successfully verified using HyTech, a symbolic model checking software for linear hybrid systems.

1 Introduction

Autonomous robotic missions by nature have complex control systems. In general, the necessary fault detection, isolation and recovery software for these systems is cumbersome and added on as failure cases are encountered in simulation. There is a need for a systematic way to incorporate fault tolerance in autonomous robotic control systems. One way to accomplish this could be to create a flexible control system that can reconfigure itself in the presence of faults. However, if the control system cannot be verified for safety, the added complexity of the reconfigurability of a system could reduce the system's effective fault tolerance.

Mission Data System (MDS) is a software control architecture that was developed at the Jet Propulsion Laboratory [1]. It is based on a systems engineering concept called State Analysis [2]. Systems that use MDS are controlled by goals, which directly express intent as constraints on physical states over time. By encoding the intent of the robot's actions, MDS has naturally allowed more fault response options to be autonomously explored by the control system [3].

A great deal of work to date has focused on detecting and recovering from sensor failures in the control of autonomous systems [4]. Several fault tolerant control architectures for autonomous systems have been developed in which the control effort is layered to deal with faults on different levels, including low levels of hardware control and high levels of supervisory control [5], [6]. Fault diagnosis can be handled by modeling complex systems as stochastic hybrid systems with modes that account for failure states. The failures can then detected using multiple-model based hybrid estimation schemes [7] or by using variations of traditional particle filters to aid in the accurate estimation of low probability but high risk failure modes [8]. Although many fault tolerant control systems achieve reconfigurability, few actually change the commands given to the system. One system uses adaptive neural/fuzzy control to reconfigure the control system in the presence of detected faults [9], and another reconfigures both the control system design and the inputs to the control system [10], although neither adjusts the intent of the commands in response to failures.

Fault tolerant control systems are modeled in different ways, but one particularly useful method is to model them as hybrid systems. Much work has been done on the control of hybrid systems [11]. When the continuous dynamics of these systems are sufficiently simple, it is possible to verify that the execution of the hybrid control system will not fall into an unsafe regime [12]. There are several software packages available that can be used for this analysis, including HyTech [13], UPPAAL [14], and VERITI [15], all of which are symbolic model checkers. HyTech in particular is used for checking linear hybrid automata, where the dynamics of the continuous variables can be modeled by linear differential inequalities that take the general form of $\mathbf{A}\dot{\mathbf{x}} \leq \mathbf{b}$ [12]. Safety verification for fault tolerant hybrid control systems ensures that the occurrence of certain faults will not cause the system to reach an unsafe state.

[*]Tech. Rep. CaltechCDSTR:2007.001

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **15 AUG 2007**	2. REPORT TYPE	3. DATES COVERED **00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE **Conversion and Verification Procedure for Goal-Based Control Programs**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **California Institute of Technology,Materials and Process Simulation Center,Pasadena,CA,91125**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Fault tolerance and safety verification of control systems are essential for the success of autonomous robotic systems. A control architecture called Mission Data System, developed at the Jet Propulsion Laboratory, takes a goal-based control approach. In this paper, a method for converting goal network control programs into linear hybrid systems is developed. The linear hybrid system can then be verified for safety in the presence of failures using existing symbolic model checkers. An example task is developed and successfully verified using HyTech, a symbolic model checking software for linear hybrid systems.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**18**	

In this paper, MDS is used as a goal-based control architecture for a representative robotic task involving sensor failures and goal re-elaboration. The major contribution of this report is the continued design of a process to convert complex goal networks with several state variables and various fault tolerant goal elaborations into hybrid automata that can be verified for safety using existing symbolic model checking software. An example goal network is developed, converted to a hybrid automaton, and then verified for safety in the presence of sensor failures.

The structure of this paper is as follows. Section 2 summarizes important concepts of MDS which pertain to this work. Section 3 introduces the example task and goal network design. Section 4 describes the general process for converting goal networks into hybrid automata. Section 5 returns to the example, discussing the hybrid automata that were created and the results of the safety verification. Finally, Section 6 concludes the paper and discusses future directions of research.

2 Mission Data System Overview

2.1 State Analysis

State Analysis is a systems engineering methodology that focuses on a state-based approach to the design of a system [2]. In State Analysis, the control system and the system under control are considered separately. Models of state variable effects in the system under control are used for such things as the estimation of state variables, control of the system, planning, and goal scheduling. State variables are representations of states or properties of the system that are to be controlled or that affect a controlled state. Examples of state variables could include the position of a robot, the temperature of the environment, the health of a sensor, or the position of a switch.

Using State Analysis, the state variables of the system under control are identified. A model of the system under control is developed and controllers and estimators are designed using the models. Goals and goal elaborations are created, also based on the models. Goals are specific statements of intent used to control a system by constraining a state variable in time. Goals are elaborated from a parent goal based on the intent and type of goal, the state models, and several intuitive rules, as described in [2].

2.2 Mission Data System

A core concept of State Analysis is that the language used to design the control system should be nearly the same as the language used to implement the control system. Therefore, the software architecture, Mission Data System, is closely related to the systems engineering theory described in the previous section.

Data structures called software state variables are central to MDS [16]. A software state variable can contain estimates of much information; for example, a position state variable for a robot in the plane could contain the robot's (x, y) position, its velocity in component form, and uncertainty values for each piece of information. Each state variable has a unique estimator, and if necessary, a controller. Goals can be created that constrain some or all of a state variable's information. For example, a goal could constrain the velocity of the position state variable used in the previous example, but could leave the position or uncertainties unconstrained.

Goal networks replace command sequences as the control input to the system. Goal networks consist of a set of goals with their associated starting and ending time points and temporal constraints. A goal may cause other constraints to be elaborated on the same state variable and/or on other causally related state variables. These goals must have an associated elaboration class. The elaboration class instructs the elaborator in MDS to add certain goals to the goal network in support of the parent goal. The goals in the goal network and their elaborations are scheduled by the scheduler software component so that there are no conflicts in time, goal order or intent. The scheduled goals are then achieved by the estimator or controller of the state variable that is constrained.

Elaboration allows MDS to handle tasks more flexibly than control architectures based on command sequences. One example is fault tolerance. Re-elaboration of failed goals is an option if there are physical redundancies in the system, many ways to accomplish the same task, or degraded modes of operation that are acceptable for a task. The elaboration class for a goal can include several pre-defined tactics. These tactics are simply different ways to accomplish the intent of the goal, and tactics may be logically chosen by the elaborator based on programmer-defined conditions. This capability allows for many common types and combinations of faults to be accommodated automatically by the control system [3].

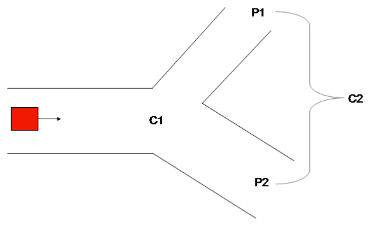


Figure 1: Simulated robotic task

3 Task and Goal Network Design

This section describes the design of an autonomous robotic task and two goal networks that will accomplish the task in the presence of sensor failures. This example illustrates some of the MDS principles outlined in the previous section.

3.1 Task Design

An autonomous robotic task is considered in which a simulated robot with several sensors follows a path within a given uncertainty bound. The task could be compared to a Mars scientific mission in which there are two points of interest (p_1 and p_2); the first is more desirable but needs a lower uncertainty in the robot's position to reach it. The mission is considered a success if the robot does not wander off the path (where it could be damaged or get stuck), and the mission is completed if the robot reaches either point of interest.

As shown in Figure 1, the planned route for the simulation consists of two checkpoints, c_1 and c_2; after the first checkpoint, c_1, there are two possibilities for the location of c_2, p_1 and p_2. The first of these possibilities, p_1, lies down a path that has a tighter error bound and requires a higher standard of sensor health. The other possibility, p_2, lies down a second path that allows for a larger error bound and a somewhat degraded sensor capability. An additional constraint is the maximum speed that the robot can have, which depends on the health of the sensors. As the collective sensor health degrades, the robot's maximum allowable speed decreases to reduce the potential impact of more sensor uncertainty on the robot's position.

The path is successfully navigated by the robot if the robot stays within the path boundaries, representing the error bounds allowed down each path. Completion of the task occurs when the robot navigates to and stops sufficiently near c_2 without breaching the boundary. The second checkpoint, c_2, is first assigned to be at location p_1, but can be changed to be p_2 upon the failure or degradation of critical sensors.

3.2 State Variables

The simulated robot used in this example is equipped with three sensors: a differential GPS, a LADAR unit, and odometry (the collection of position, orientation, and velocity information deduced from wheel encoders). These three sensors are used to estimate the robot's position, orientation, and velocity information. The LADAR scan matching algorithm developed by Lu and Milios [17], which outputs position and orientation, was adapted for use in this simulation.

Several state variables are needed to describe this system. First, the position and the orientation state variables track Cartesian and angular position and velocity, as well as the covariance matrices for the estimates. Three state variables discretely describe the health of the three sensors as GOOD, FAIR, POOR, and FAILED. Using the same labels, the health of the overall sensing system for this specific task is described by the system health derived state variable [16]. The state effects diagram is shown in Figure 2. The health of the sensors affect the knowledge of the robot's position, and so the system health indirectly affects the knowledge of the robot's position and orientation. Since this state effect exists, it is possible for goals on the position state variable to elaborate constraints on the system health state variable.

The robot's position and orientation are estimated using a multiple model-based method [18]. In order to make the estimation algorithm robust to changes in sensor availability and health, different Kalman filters were designed for each possible combination of sensors. This approach was chosen for its relative simplicity and ease of implementation. The three sensor health variables are estimated using a different process. In each sensor's health estimator, the output of the sensor is converted to a measured position and velocity value and is compared to the other sensor's outputs. Then,

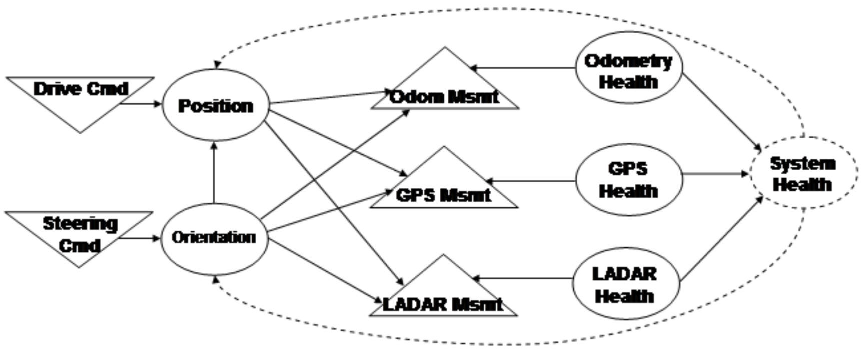


Figure 2: State effects diagram; solid ovals represent state variables and dashed ovals represent derived state variables. The effects of the sensor health state variables on position and orientation state variables are summarized by the system health state variable and are represented by the dashed arrows.

a voting scheme is employed to determine the health of the sensor. Once a sensor is failed, it is assumed to always be failed. The system health derived state variable is estimated using the three sensor health state variables. For this example, it is assumed that there is no uncertainty in the sensor health estimations.

3.3 Goal Design

One way to construct a goal network for the example that was introduced in Section 3.1 is shown in Figure 3. There is a goal called BeAt1or2Goal that elaborates into two goals on the robot's position, GetToC1Goal and GetToC2Goal. The first, GetToC1Goal, tells the robot to move to the first checkpoint, c_1. The second goal, GetToC2Goal, has two tactics it can elaborate; the first is GetToP1Goal and the second tactic is GetToP2Goal. These goals tell the robot to drive to the second checkpoint, which is either p_1 or p_2. GetToC1Goal elaborates to one tactic, which contains a goal constraining the system health to be FAIR or better and a goal constraining the orientation state variable to turn to the heading angle. GetToC2Goal elaborates two tactics, GetToP1Goal and GetToP2Goal, which both elaborate a similar constraint on the orientation state variable and a constraint on the system health for it to be GOOD and FAIR, respectively.

The constraints on the velocity are derived from a macro goal called SpeedLimitGoal. This goal has two tactics. The first constrains the velocity to have an upper bound of v_1 and constrains the system health to be GOOD. The second tactic constrains the velocity to have an upper bound of v_2 and constrains the system health to be FAIR. This macro goal and its tactics are bounded by an opening timepoint that is concurrent with the start of GetToC1Goal and by an ending timepoint that is concurrent with the start of BeAt1or2Goal.

4 Verification Procedure

Hybrid system analysis tools can be used to verify the safe behavior of linear hybrid systems; therefore, a procedure to convert goal networks into hybrid systems is an important tool for goal network verification. The procedure described in this section allows certain structures of goal networks to be converted into simple, linear hybrid automata in a general way. There are few restrictions on the goal networks; they can constrain several state variables, which may be linearly related to each other, and they can have goals with several tactics. The amount of time needed to complete a goal can be constrained or unconstrained, and transition and elaboration logic can be based on the state variable, affecting or affected state variables, order, and time. However, goal tactics that have constraints on controllable state variables must not introduce time points that occur during a goal on a controllable state variable that has transition conditions that depend on completion. This type of goal is called unsplittable. Also, goals in the network must have a unique ordering or scheduling, though several goals can be active concurrently as long as the goals on the same state variable are mergeable.

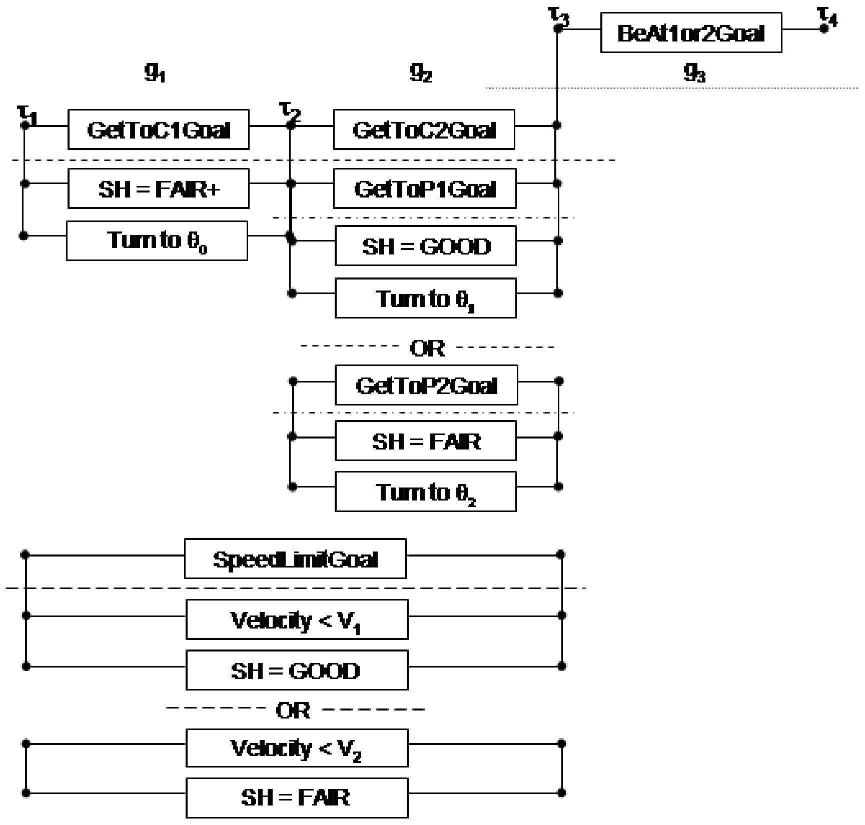


Figure 3: Goal network and elaborations for the example task; dashed lines under a goal indicate that goals beneath the line are elaborated from it with the “OR’s” delineating tactics

4.1 Conversion to Hybrid Automata

4.1.1 Controllable and Continuous Dependent State Variables

The first hybrid automaton that will be created from the goal network is based on all goals constraining the controllable and continuous dependent state variables. The hybrid automata created from the uncontrollable and discrete dependent state variables will be discussed in Section 4.1.2. The process to create the controllable and continuous dependent state variables' hybrid automaton is as follows.

1. Label each state variable in the goal network as controllable, uncontrollable, or dependent. Controllable state variables (CSVs) are directly controllable and have associated command classes. Uncontrollable state variables (USVs) are not controllable and do not have model dependencies on any controllable or dependent state variables (though CSVs and dependent state variables may have model dependencies on them). Dependent state variables (DSVs) do have model dependencies on one or more CSVs, and may also depend on one or more USVs. Dependent state variables do not have an associated command class.

2. Elaborate (keeping track of parent and tactic information) all tactics of all goals in the given scheduled goal network. Elaboration can be layered. Number each time point that is associated with a goal on a controllable state variable sequentially as $\tau_1, \tau_2, ..., \tau_{N+1}$, where $N+1$ is the number of sets of goals between time points.

3. Group goals between consecutive time points as $g_1, g_2, ..., g_N$. For the hybrid automaton, place "connectors," or small empty circles, between where the groups will be.

4. For each group $g_i, i = 1, 2, ..., N$, where N is the number of groups, find all the branch goals and uncontrolled tactics in the group. Branch goals are goals on controllable state variables that are not also ancestors of other goals on CSVs in the group. Uncontrolled tactics are tactics that do not have any constraints on controllable state variables either in the tactic or descending from goals in the tactic, but have at least one sibling tactic that contains at least one constraint on a CSV. Sibling tactics are tactics elaborated from the same goal. Make a location for each branch goal, and make a placeholder location for each uncontrolled tactic. If two or more branch goals in a group are in the same tactic, combine those branch goals into one location. Locations are the modes or the discrete states of the hybrid system.

   For each group, add to each location or placeholder location the parent goals that constrain a CSV of the branch goal or uncontrolled tactic. Once a goal on a CSV is added to a placeholder location, it becomes a location. Each time a parent goal is encountered (whether it is added or not), check for sibling goals of the parent goal. Sibling goals are goals on CSVs or goals that elaborate goals on CSVs that are concurrently elaborated from the same parent goal into the same tactic. Sibling goals share the same starting and ending time points. If there are one or more sibling goals, all locations containing each sibling goal are combined combinatorially so that each location that contains each sibling goal is combined with each location that contains each other sibling goal into a new location. The number of new locations from the sibling goal combinations will be the product of the numbers of locations in which each sibling goal was originally located. This process continues until the root goal of each location and siblings of the root goal are added. Root goals are goals on CSVs that have no ancestor goals on CSVs in the group. All root goals are sibling goals of all other root goals in the group, unless the root goals are elaborated into different tactics from the same parent goal. Branch goals in different tactics elaborated from the same root goal will never be present in the same location.

5. For each location in each group, label the location with the continuous or discrete dynamical equations that describe the evolution of each CSV that is constrained by a goal in that location and each continuous DSV that either is constrained by goal in one of the represented tactics, is elaborated from a goal in a represented tactic, or is a parent goal of one of the represented tactics. If there is a time constraint on any of the goals in the group, label the affected locations with the differential or difference equation updating the time variable.

6. Add Success and Safing locations to the hybrid automaton.

7. For each parent goal with tactics that constrain at least one controllable state variable, fill out the elaboration logic chart outlined in Table 1. For each tactic, list the logic that controls which tactic is initially elaborated in the "Starts in" column. These conditions often depend on the USV and DSV 'sibling' goals in the tactics. For goals with only one tactic, the entry will be "True." In the "Fail to" column, list where the goal network

Table 1: Outline of an elaboration logic table

Tactic	**Starts in**	**Fail to**	**Fail Conditions**
1			
2			
:			

Table 2: Outline of a transition, success, and failure logic table

From	**Unconstrained**	**Maintenance**	**Control**	**Combo 1**	**...**	**Success**	**Fail**
Unconstrained							
Maintenance							
Control							
Combo 1							X
:							X

execution goes upon the failure of any goal in the tactic. Possible entries for this column are other tactics, Safing, or 'Fail up,' which causes the failure of the parent goal. In the "Fail Conditions" column, list the conditions that would cause the failure of this tactic. Possible entries for this column include failure conditions for sibling goals on uncontrollable and dependent state variables, failure conditions for goals on USVs or DSVs elaborated from goals in the tactic if no goals on CSVs are also elaborated from those goals, or failures of the controllable state variables in the tactic (abbreviated 'CGF' for controllable or child goal failure; the actual failure conditions for these goals will be determined later).

More than one concurrently elaborated goals on uncontrollable and dependent state variables could create more than one constraint for the "Starts in" and "Fail Conditions" logic, and these constraint conditions should be combined with a logical 'AND' connector in the "Starts in" column and with a logical 'OR' connector in the "Fail Conditions" column. Multiple failure conditions in a tactic's "Fail Conditions" entry could cause several destinations in the "Fail to" column, which should all be listed with their specific failure condition(s). Goals on DSVs may have constraints on affecting controllable, uncontrollable, or dependent state variables due to the state effects models and these constraints can appear in the elaboration logic table.

8. For each controllable state variable, list out the different types of constraints placed on it during the goal network. Examples of the types of constraints that could be present include control, maintenance, knowledge, or unconstrained; there may be more than one constraint type for each of those general labels, also. Constraint types are designed for each state variable based on the different ways the state variable will be controlled or estimated. For each type of constraint and each possible combination of constraint types on each controllable state variable, fill out the table of transition logic between types of constraints, as outlined in Table 2. This transition logic is associated with the isReadyToTransition function in MDS.

9. For each constraint type for each CSV, fill out the columns of success and failure condition logic, as outlined in Table 2. Success logic is defined as the transition logic from a constraint type to either a specified success state or to an unconstrained state. Failure logic is defined as the conditions that would invalidate the constraint, and is part of what is referred to as 'CGF' in the "Fail Conditions" column of the elaboration logic tables. This failure logic is associated with the isStillSatisfiable function in MDS.

10. For each group $g_i, i = 2, 3, ..., N$, use the"Starts in" logic from the elaboration logic table for each parent goal whose tactics are present in g_i to make transition arrows from the group connector between groups g_{i-1} and g_i to the appropriate locations and label the arrows with the elaboration logic conditions. For group g_1, the arrow(s) will not originate from a group connector, but instead will indicate the starting point for the automaton. For locations representing branch goals from more than one tactic, combine the "Starts in" elaboration logic for each tactic using a logical 'AND' connector. Eliminate any connections that have transition conditions that logically reduce to "False." If g_i has only one location, the default "True" transition is to that location.

11. For each location in each group, use the “Fail to” elaboration logic for each tactic that is represented in the location to create exit transitions to the appropriate failure locations. If the “Fail to” location is to “Fail up,” the parent goal, which was elaborated from another goal, is failed. The location listed in the “Fail to” column for the ‘CGF’ condition in the tactic containing the parent goal in another elaboration logic table is used. If a goal in a location is a root goal that has an elaboration logic table, use the “Fail to” location listed for ‘CGF’ for any of its tactics as the destination of its failure transition arrow. If a goal in a location is an orphan goal, with no parents or descendants, draw a transition arrow to Safing for its failure transition. For each transition arrow, use the “Fail Conditions” elaboration logic for that tactic and destination and/or the failure logic for the constraint type(s) from the transition table combined with logical ‘OR’ connector as the transition condition. From any location, if there are two or more arrows that are pointing to the same failure location, then the arrows can be combined and the transition conditions from each are combined using a logical ‘OR’ connector. Eliminate any transition arrows whose transition conditions logically reduce to “False.”

12. For each group $g_i, i = 1, 2, ..., N-1$, add transition arrows from each location in g_i to the group connector between g_i and g_{i+1}. Label each transition with the transition logic from the constraint type (or merged constraint type) in that location to the constraint type (or merged constraint type) for that CSV found in g_{i+1}. If there is more than one constraint type in g_{i+1} that can be transitioned into from the group connector, add the logic for the other constraint types and combine the transition conditions with a logical ‘OR’ connector. Append the transition conditions to the appropriate “Starts in” transitions by using a logical ‘AND’ connector. If there is more than one CSV present in a location, combine transition logic with a logical ‘AND’ connector. Eliminate any transition arrows whose transition conditions logically reduce to “False.”

13. For g_N, add transition arrows from each location to the Success location. Using the success logic conditions in the transition logic table, label each transition with the success transition logic from the constraint type (or merged constraint type) of the CSV. If there is more than one CSV present in a location, combine the success logic conditions with a logical ‘AND’ connector. Eliminate any transition arrows whose transition conditions logically reduce to “False.”

14. For each group $g_i, i = 1, 2, ..., N$, if there are any goals present in any location that has time constraints, do the following. Add an action to each transition into the affected locations from the group connector (or from the initial transition if in g_1) to reset the time variable to zero. Add to the transition conditions out of the affected locations the constraint on the time variable by appending the time constraint to the other transition conditions with a logical ‘AND’ connector. If there are other transition conditions besides “True,” include a failure transition from the location to the location indicated by the ‘CGF’ condition of the elaboration logic table in which the goal with the time constraint belongs to a tactic or, if none, is the parent goal. If the time constrained goal is an orphan, the transition arrow goes to Safing. The failure conditions are the time constraint and the negative of the other transition conditions.

15. For each group $g_i, i = 1, 2, ..., N$, remove any location that is not entered by any transitions and remove all transitions that originate at that location. Remove any location whose only entry transition has conditions that, if true, immediately make an exit transition true. While keeping the overall connection between the matching entry and exit transitions intact, remove all other exit transitions from the deleted location. Remove any location that has an exit transition condition that is always “True” or if all entry transitions contain the conditions of an exit transition as a subset of their transition conditions. Keep the connection between each entry transition to the location and the destination of the aforementioned exit location, and remove all other exit conditions from the location.

4.1.2 Uncontrollable and Discrete Dependent State Variables

The process in the previous section results in a hybrid automaton for the controllable and continuous dependent state variables; the process for creating hybrid automata for the uncontrollable and discrete dependent state variables is described in this section. Since the transitions between discrete or continuous states for these state variables are not directly controllable, they generally happen stochastically, as modeled, or at a given rate. This information will be used to create the hybrid automata for these state variables and for setting up the verification problem.

The process outlined below details the creation of the other hybrid automata.

1. For each uncontrollable state variable, group the values that the uncontrollable state variable can take into a finite number of discrete sets.

2. Using only the discrete sets that relate to the controllable state variables' automaton, make corresponding locations in each uncontrollable and each discrete dependent state variables' hybrid automaton.

3. For each state variable's automaton, introduce the appropriate (or modeled) transitions between the locations and allow the dynamics and the transitions to be controlled by modeled effects, rates, or stochastic ranges of rates. Parameterize the transitions between the locations in each stochastic uncontrollable state variable's automaton.

4.2 Hybrid System Verification

Once all the hybrid automata are created, the system is ready for verification. The process now becomes dependent on which software will be used to verify the system as the following steps somewhat depend on syntax.

1. Convert the controllable and dependent state variables' hybrid automaton into a verifiable form. Specifically for the HyTech software, convert all locations in the automaton into locations in the code and list all exit transitions and transition conditions with each location. For exit transitions that end at group connectors, separately list the combination of that exit transition with each entry condition that begins at that group connector. Combine the exit transition conditions with the entry transition conditions using the logical 'AND' connector. Remove any transitions whose transition conditions logically reduce to "False." Add linearized dynamics that are appropriate to each location. If necessary, split locations into two or more to deal with the linearization of the dynamics.

2. Convert each of the USV and discrete DSV automata into code as described in the previous step, allowing for the parametrization of important transitions.

3. Synchronize the transitions between locations in each of the uncontrollable and discrete dependent state variables' automata to the transitions in the controllable state variable's hybrid automaton whose transition conditions depend on the value of the uncontrollable or discrete dependent state variables.

4. Find "incorrect" or "unsafe" sets and search over the parameters to ensure that the hybrid system does not enter into these sets.

The procedures for the creation of the hybrid automata could conceivably be automated, as could parts of the procedure for initiating the verification. This general procedure can be followed to complete the verification of general goal networks with several state variables.

5 Verification Results

Returning to the example task and goal networks described in Section 3, this section will describe the safety verification of the example. A simpler example verified using the same procedure can be found in [19] and a similar example verified using an earlier version of this procedure can be found in [20].

5.1 Conversion to Hybrid Automata

5.1.1 Controllable and Continuous Dependent State Variables

The goal network for this problem is shown in Figure 3. Starting with the process for controllable and continuous dependent state variables, the following steps are taken, which correspond with the numbered steps in Section 4.1.1.

1. The position and orientation state variables are controllable state variables, as they are directly associated with command classes. The system health state variable is an uncontrollable state variable as it does not depend on anything that is associated with a command class.

2. The elaboration of the scheduled goal network is shown in Figure 3. The time points have been labeled as shown in Figure 3.

3. There are three groups in this goal network, labeled in Figure 3.

Table 3: Elaboration logic table for BeAt1or2Goal

Tactic	**Starts in**	**Fail to**	**Fail Conditions**
1	True	Safing	CGF

Table 4: Elaboration logic table for GetToC1Goal

Tactic	**Starts in**	**Fail to**	**Fail Conditions**
1	True	Fail up	sh = POOR or CGF

4. In g_1, there are three branch goals; two are elaborated from the SpeedLimitGoal and one is elaborated from GetToC1Goal. The two branch goals elaborated from the SpeedLimitGoal are also root goals, as is GetToC1Goal. The branch goal on the orientation state variable combines with its parent goal, GetToC1Goal, since GetToC1Goal is a constraint on the position state variable. GetToC1Goal has two siblings, the two SpeedLimitGoal tactics, that represent different tactics elaborated from the same parent, so the GetToC1/Turn to θ_0 location combines with each other sibling goal separately, producing two locations in g_1, as shown in Figure 4.

   In g_2, a similar process unfolds. There are now four branch goals; the two that are elaborated from the SpeedLimitGoal are also root goals, and the other two are constraints on the orientation state variable, one of which is elaborated from GetToP1Goal and the other from GetToP2Goal. The two branch goals on the orientation state variable each combine with their respective parent goals, which are constraints on the position state variable. GetToP1Goal and GetToP2Goal are root goals representing tactics from the same parent goal; therefore, they do not combine, but do combine with each of the root goals from the SpeedLimitGoal separately, resulting in four locations in g_2.

   Finally, in g_3, there is only one branch goal, which is also a root goal; this results in one location. The final outcome of this step is illustrated in Figure 4.

5. The dynamical equations controlling the evolution of the position and orientation state variables are shown in each location in Figure 4. Note that the equation for position propagation is piecewise linear due to the speed limit goals in each location.

6. Success and Safing locations were added, as shown in Figure 4.

7. The elaboration logic table for BeAt1or2Goal is Table 3, for GetToC1Goal is Table 4, for GetToC2Goal is Table 5, for GetToP1Goal is Table 6, for GetToP2Goal is Table 7, and for the SpeedLimitGoal is Table 8.

8. The transition logic table for the position state variable is Table 9 and for the orientation state variable is Table 10. For the position state variable, there are three types of constraints, the control constraint (GetToC1Goal, GetToP1Goal, and GetToP2Goal), the velocity constraint (both speed limit goals), and the maintenance constraint (BeAt1or2Goal). There is one possible combination of constraints, the combination of the control and velocity constraints. For the orientation state variable, there are two types of constraints, the control constraint (Turn to θ goals) and the unconstrained constraint.

9. The success and failure logic is in Table 9 for the position state variable and in Table 10 for the orientation state variable. The success state is one in which the position is in the constrained location (p_1 or p_2) and velocity is

Table 5: Elaboration logic table for GetToC2Goal

Tactic	**Starts in**	**Fail to**	**Fail Conditions**
1 (*GetToP1Goal*)	if sh = GOOD	Fail up	CGF
2 (*GetToP2Goal*)	if sh = FAIR	Fail up	CGF

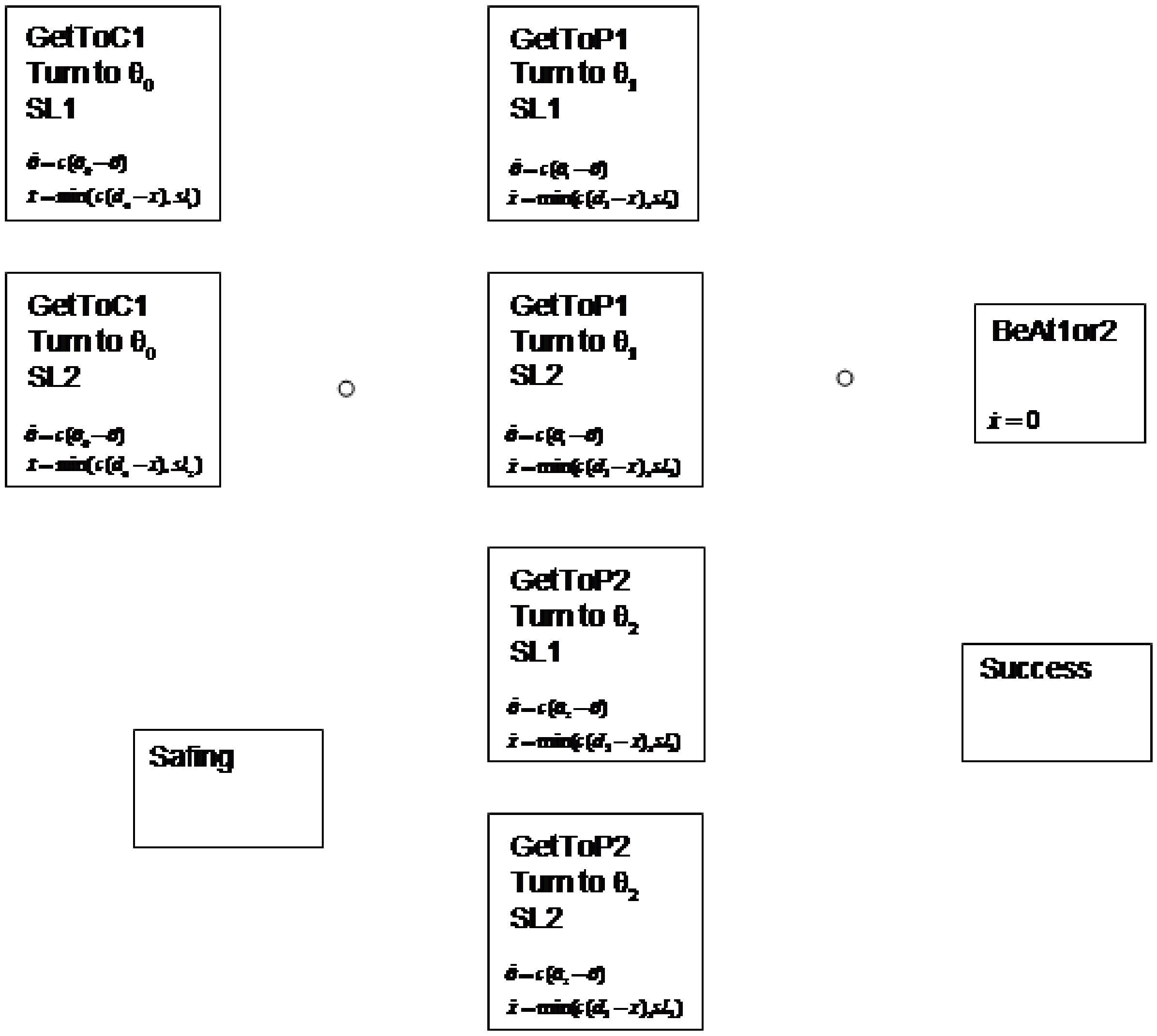


Figure 4: Locations of the controllable state variables' hybrid automaton. Dynamical equations for position and orientation labeled in each location.

Table 6: Elaboration logic table for GetToP1Goal

Tactic	Starts in	Fail to	Fail Conditions
1	True	Fail up	sh = FAIR or sh = POOR or CGF

Table 7: Elaboration logic table for GetToP2Goal

Tactic	Starts in	Fail to	Fail Conditions
1	True	Fail up	sh = POOR or CGF

Table 8: Elaboration logic table for SpeedLimitGoal

Tactic	**Starts in**	**Fail to**	**Fail Conditions**
1 (v_1)	if sh = GOOD	if sh = FAIR, Tactic 2; else Safing	sh = FAIR or sh = POOR or CGF
2 (v_2)	if sh = FAIR	Safing	sh = POOR or CGF

Table 9: Transition logic table for the position state variable (x = position, v = velocity, c= current constraint (position or velocity), nc = new constraint (velocity))

From	**Maintenance**	**Control**	**Velocity**	**Combo**	**Success**	**Fail**
Maintenance	True	True	True	True	True	$v \neq 0$
Control	$x = c$ & $v = 0$	$x = c$	$x = c$ & $v \leq nc$	$x = c$ & $v \leq nc$	$x = c$ & $v = 0$	$x \neq c$ & $v = 0$
Velocity	$v = 0$	True	$v \leq nc$	$v \leq nc$	$v = 0$	$v > c$
Combo	$x = c$ & $v = 0$	$x = c$	$x = c$ & $v \leq nc$	$x = c$ & $v \leq nc$	$x = c$ & $v = 0$	X

zero (a maintenance constraint) and the orientation is unconstrained.

10. For g_1, the "Starts In" elaboration logic is "True" for GetToC1Goal and depends on the system health state variable for the SpeedLimitGoal tactics; the combination results in only the conditions on the system health state variable. For g_2, the "Starts In" elaboration logic for both GetToC2Goal and SpeedLimitGoal depend on the system health state variable. Therefore, the conditions for each tactic in the locations are combined with a logical 'AND' connector, which in two cases results with a "False" transition. These transitions are removed. The only location in g_3 results in a "True" transition to it from the group connector. These transitions are illustrated in Figure 5.

11. Starting with the locations in g_1, from the "Fail to" and "Fail Conditions" columns in the elaboration logic tables of SpeedLimitGoal and GetToC1Goal (Tables 8 and 4), the appropriate failure transition arrows are drawn and transition conditions labeled. The failure conditions found in the position and orientation state variable's transition logic table (Tables 9 and 10) for the control constraint types are combined with the conditions on the transitions from each location to Safing using a logical 'OR' connector. This is due to the "Fail to" column in the BeAt1or2Goal's elaboration table (Table 3).

    For the locations in g_2, the "Fail to" and "Fail Conditions" columns of the elaboration logic tables of SpeedLimitGoal and GetToC2Goal are used (Tables 8 and 5). For g_3, the only failure transition is to Safing from Table 3 and the condition is from the position state variable's transition logic table, Table 9. These transitions are shown in Figure 5.

12. In g_1 and g_2, the transitions from each location are to the group connector. The transition conditions are, for g_1, the combination of the control to control entry of the orientation state variable's transition logic table (Table 10) and the Combo1 to Combo1 entry of the position state variable's transition logic table (Table 9), and for g_2, the control to unconstrained entry of the orientation state variable's transition logic table and the Combo1 to maintenance entry of the position state variable's transition logic table. These transition conditions are combined with a logical 'AND' connector. These transitions are illustrated in Figure 5.

Table 10: Transition logic table for the orientation state variable (θ = orientation, $\dot{\theta}$ = angular velocity, c= current constraint (orientation))

From	**Control**	**Unconstrained**	**Success**	**Fail**
Control	$\theta = c$	$\theta = c$	$\theta = c$	$\theta \neq c$ & $\dot{\theta} = 0$
Unconstrained	True	True	True	False

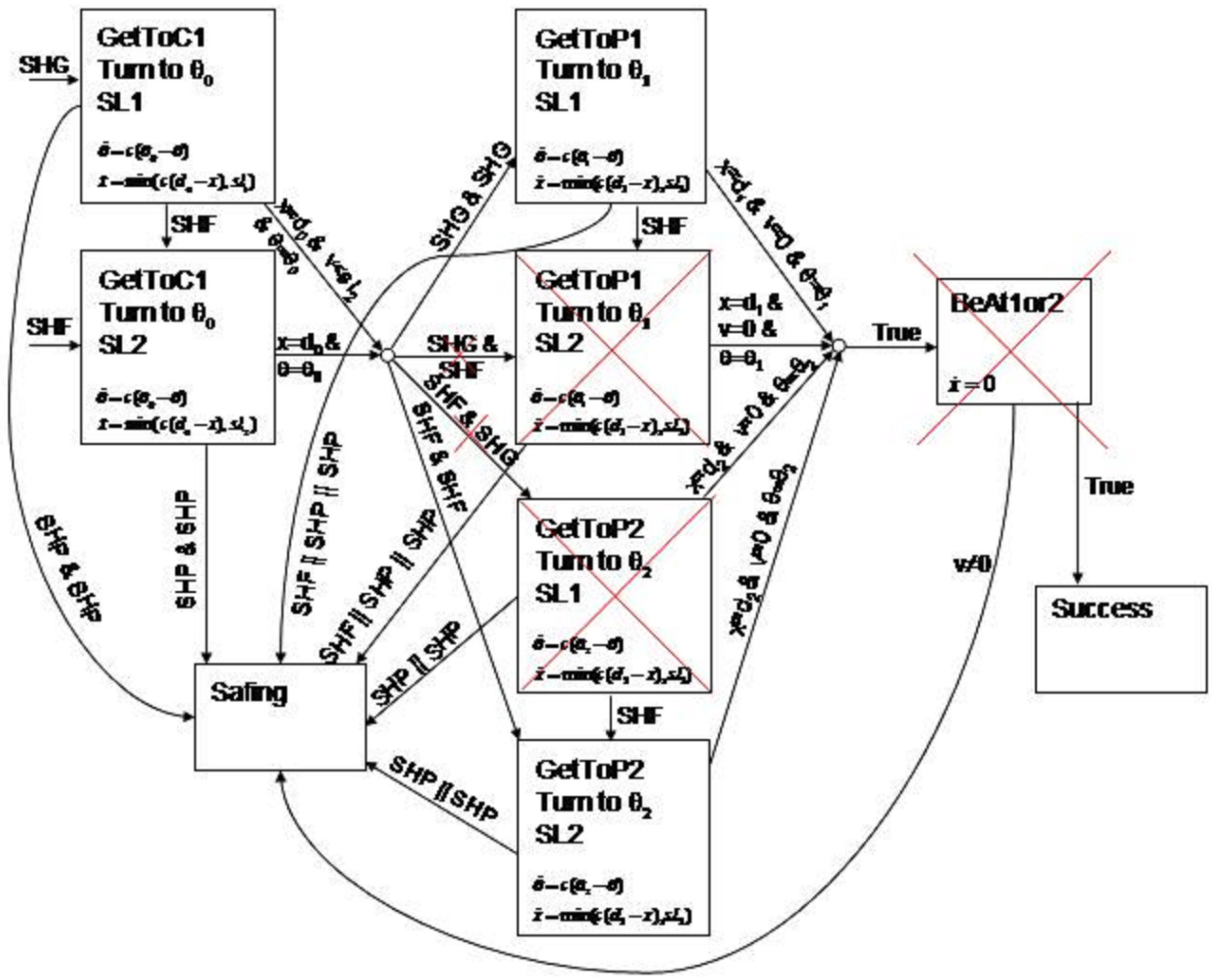


Figure 5: Unreduced version of the controllable state variables' hybrid automaton. SHG, SHF, SHP stand for system health is GOOD, FAIR, and POOR, respectively. Failure conditions due to position and orientation constraint failures are not labeled for clarity.

13. For the last group, g_3, the transition from the only location to the Success location is conditioned by the maintenance to Success entry of the position state variable's transition logic table, Table 9 and on the unconstrained to Success entry of the orientation state variable's transition logic table, Table 10). This transition is shown in Figure 5.

14. No goals have definite time constraints.

15. In g_2, the GetToP2/SL1 location has no remaining entry transitions and so is eliminated along with all exit transitions. Also in g_2, the GetToP1/SL2 embedded location has one entry transition whose condition, when "True", also automatically makes a failure transition "True," and so that location is eliminated, although the entry to exit transition is kept as one transition from the GetToP1/SL1 location to Safing. Finally, in g_3, all transition conditions into the location require, among other conditions, that the velocity be zero, which makes the failure exit transition false and the other transition (to the Success location) always true. Therefore, the location (and the group) is eliminated, although the transitions from the locations in g_2 through the group connector to the Success location are kept. The eliminated transitions and locations are marked in Figure 5 and the final controllable state variable hybrid automaton is shown in Figure 6.

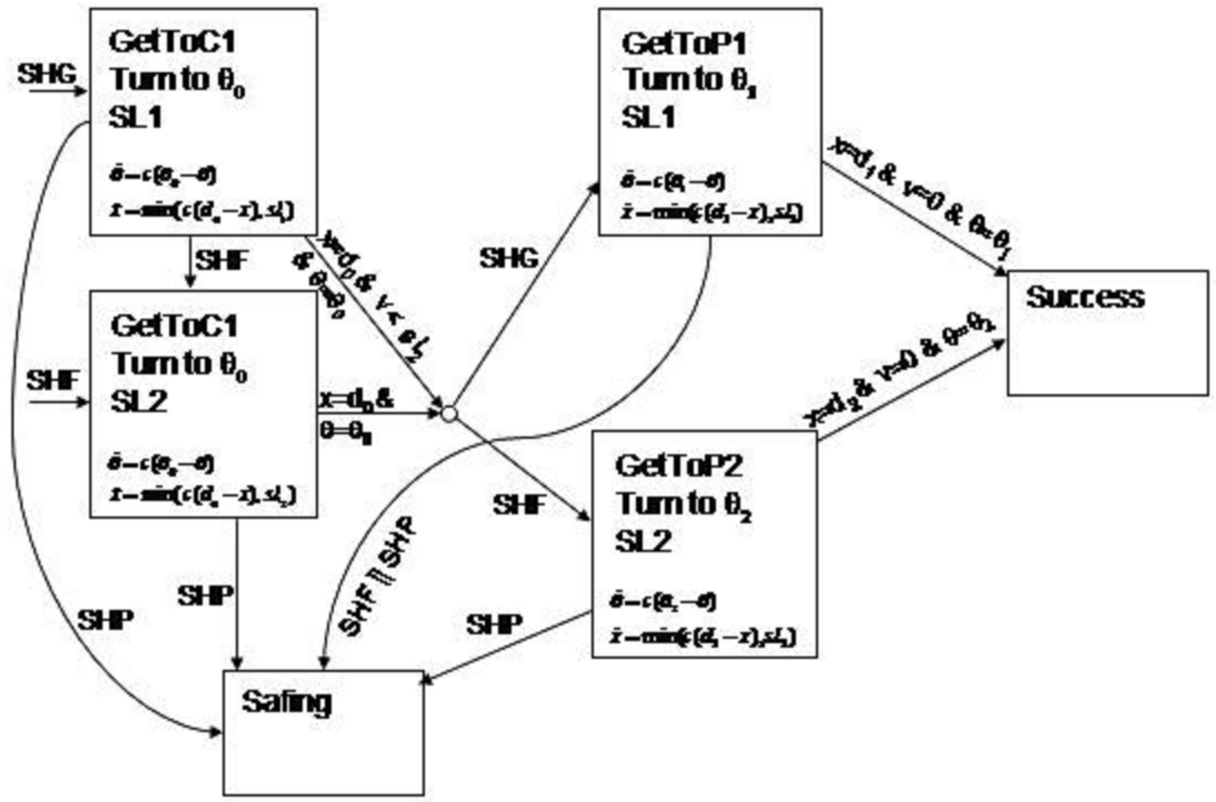


Figure 6: Final version of the controllable state variables' hybrid automaton. SHG, SHF, SHP stand for system health is GOOD, FAIR, and POOR, respectively. Failure conditions due to position and orientation constraint failures are not labeled for clarity.

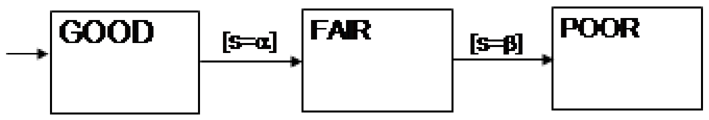


Figure 7: System health state variable's hybrid automaton

5.1.2 Uncontrollable and Discrete Dependent State Variables Automata

For this example, there is only one uncontrollable or discrete dependent state variable, the system health state variable, so there will be only one automaton created in this section. The numbered steps follow the numbered steps in the procedure from Section 4.1.2.

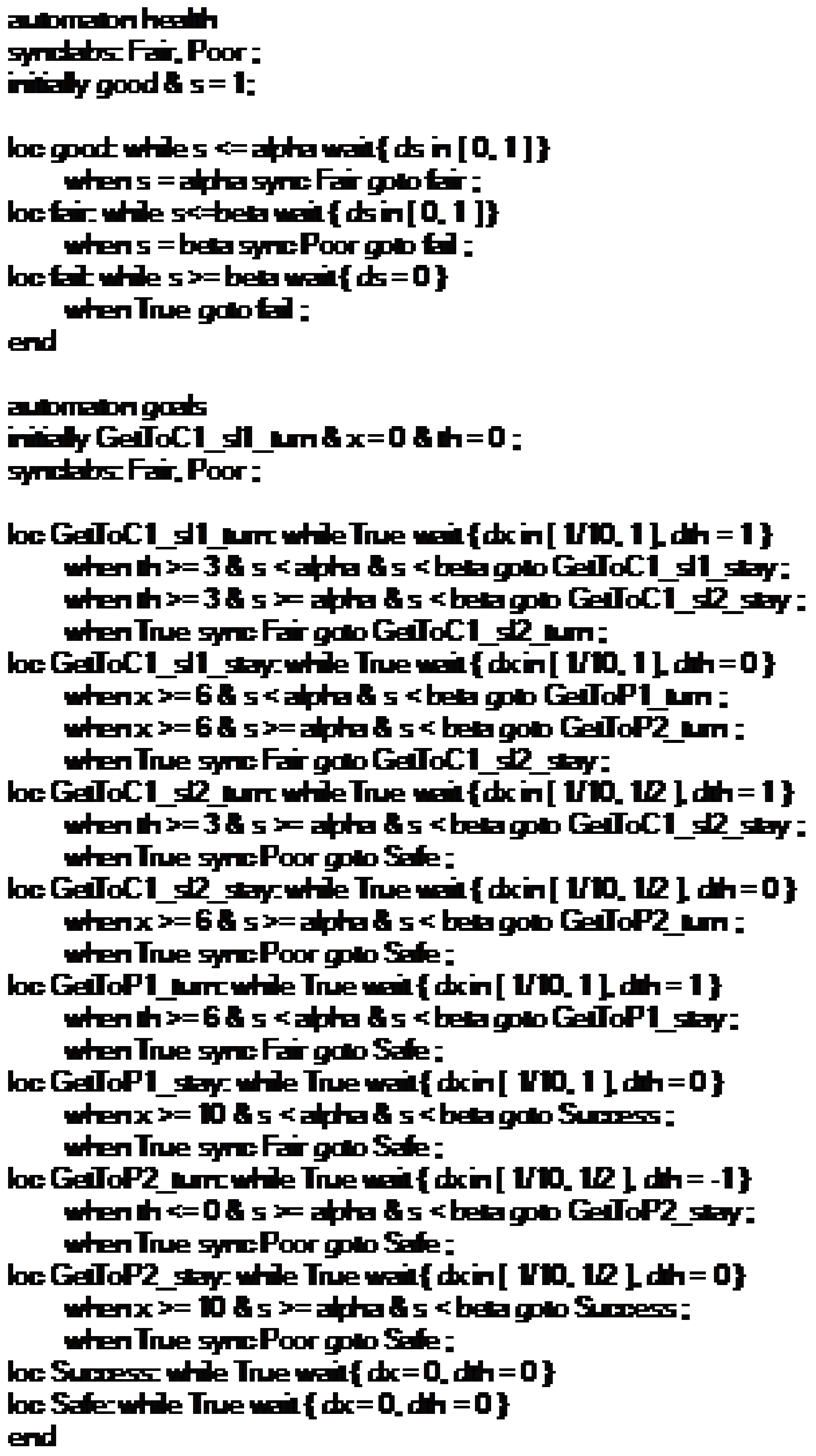
1. The system health state variable has four discrete states (GOOD, FAIR, POOR, and FAILED).
2. From the hybrid automaton for the position state variable shown in Figure 6, the states of the system health that are referenced are GOOD, FAIR, and POOR. These three states become locations of the system health state variable's automaton, as seen in Figure 7.
3. It is assumed that the system health can only degrade, and so transition arrows are drawn as shown in Figure 7. The transition conditions are stochastic rates that can range from zero to one. A variable, s, is used to represent the system health state variable, and the system health degrades as s increases. The parameters α and β (where $\alpha \leq \beta$) are chosen to represent the threshold values that s must reach for the system health to transition to FAIR and POOR, respectively.

5.2 Hybrid System Verification

The procedure from Section 4.2 is followed for the example and the steps are described below.

1. The hybrid automaton for the position and orientation state variables is converted into HyTech code. The two dynamical equations in each location must be linearized, which caused two locations to be created in the code for each location in the automaton. It was decided that the orientation constraint would be achieved much faster than the position constraint, so the locations were split accordingly with no loss of information for the verification.
2. The hybrid automaton for the system health state variable is converted into HyTech code.
3. The two automata are synchronized. The resulting HyTech code for this and the previous two steps is shown in Figure 8.
4. The union of "incorrect" sets is chosen to consist of the following subsets:
   (a) Position and orientation are Safe and $s < \alpha$
   (b) Position and orientation are GetToC1_sl1_turn and $s > \alpha$
   (c) Position and orientation are GetToC1_sl1_stay and $s > \alpha$
   (d) Position and orientation are GetToC1_sl2_turn and $s > \beta$
   (e) Position and orientation are GetToC1_sl2_stay and $s > \beta$
   (f) Position and orientation are GetToP1_turn and $s > \alpha$
   (g) Position and orientation are GetToP1_stay and $s > \alpha$
   (h) Position and orientation are GetToP2_turn and $s > \beta$
   (i) Position and orientation are GetToP2_stay and $s > \beta$

Both forward and backward analysis can be used for this verification [12]. The initial conditions start the position and orientation automaton in location GetToC1_sl1_turn and the system health automaton in GOOD. The analysis in both directions prove that there is no possible path from the initial conditions to the union of incorrect sets.



```
automaton health
synclabs: Fair, Poor ;
initially good & s = 1;

loc good: while s <= alpha wait{ ds in [ 0, 1 ]}
        when s = alpha sync Fair goto fair ;
loc fair: while s<=beta wait { ds in [ 0, 1 ]}
        when s = beta sync Poor goto fail ;
loc fail: while s >= beta wait{ ds = 0 }
        when True goto fail ;
end

automaton goals
initially GetToC1_sl1_turn & x = 0 & th = 0 ;
synclabs: Fair, Poor ;

loc GetToC1_sl1_turn: while True wait {dx in [ 1/10, 1 ], dth = 1 }
        when th >= 3 & s < alpha & s < beta goto GetToC1_sl1_stay;
        when th >= 3 & s >= alpha & s < beta goto GetToC1_sl2_stay ;
        when True sync Fair goto GetToC1_sl2_turn ;
loc GetToC1_sl1_stay: while True wait { dx in [ 1/10, 1 ], dth = 0 }
        when x >= 6 & s < alpha & s < beta goto GetToP1_turn ;
        when x >= 6 & s >= alpha & s < beta goto GetToP2_turn ;
        when True sync Fair goto GetToC1_sl2_stay;
loc GetToC1_sl2_turn: while True wait {dx in [ 1/10, 1/2 ], dth = 1 }
        when th >= 3 & s >= alpha & s < beta goto GetToC1_sl2_stay ;
        when True sync Poor goto Safe ;
loc GetToC1_sl2_stay: while True wait { dx in [ 1/10, 1/2 ], dth = 0 }
        when x >= 6 & s >= alpha & s < beta goto GetToP2_turn ;
        when True sync Poor goto Safe ;
loc GetToP1_turn: while True wait { dx in [ 1/10, 1 ], dth = 1 }
        when th >= 6 & s < alpha & s < beta goto GetToP1_stay;
        when True sync Fair goto Safe ;
loc GetToP1_stay: while True wait{ dx in [ 1/10, 1 ], dth = 0 }
        when x >= 10 & s < alpha & s < beta goto Success ;
        when True sync Fair goto Safe ;
loc GetToP2_turn: while True wait { dx in [ 1/10, 1/2 ], dth = -1}
        when th <= 0 & s >= alpha & s < beta goto GetToP2_stay;
        when True sync Poor goto Safe ;
loc GetToP2_stay: while True wait{ dx in [ 1/10, 1/2 ], dth = 0}
        when x >= 10 & s >= alpha & s < beta goto Success ;
        when True sync Poor goto Safe ;
loc Success: while True wait{ dx = 0, dth = 0 }
loc Safe: while True wait { dx = 0, dth = 0 }
end
```

Figure 8: HyTech code for the system health state variable's automaton and the position and orientation state variables' automaton. Notice the synchronization between the automata.

6 Conclusion and Future Work

This paper describes a systematic way to verify goal networks using a general procedure to translate certain types of goal networks into linear hybrid systems. A software package specializing in the analysis of linear hybrid systems can then be used to verify the safety of this system. The process was used successfully on a simple example problem, though due to the way multiple controllable and continuous dependent state variables are handled by the process, it is likely that this procedure will easily handle more complicated problems. This result is important for the development and use of reconfigurable goal networks as a method to robustly control complex embedded systems.

Future work includes the proof and automation of this procedure to translate goal networks to hybrid systems. It may also be possible to extend this procedure to apply to even more complex goal networks by using certain MDS attributes, like projections based on state models, in the transition conditions of the hybrid automata. Another extension would be to include estimation uncertainty of uncontrollable state variables in the verification procedure.

7 Acknowledgements

The authors would like to gratefully acknowledge Kenny Meyer for his many efforts in enabling this collaborative work; a special thanks to Michel Ingham for his help with the goal net design; David Wagner, Robert Rasmussen, Matthew Bennett, Mark Indictor, Daniel Dvorak, and the MDS team at JPL for feedback, suggestions, answered questions, and MDS and State Analysis instruction; and Stefano Di Cairano for his help with hybrid systems, Stateflow, and HyTech. This work was funded by NSF and AFOSR.

References

[1] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks, "Software architecture themes in JPLs Mission Data System," *IEEE Aerospace Conference*, 2000.

[2] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada, "Engineering complex embedded systems with State Analysis and the Mission Data System," *AIAA Journal of Aerospace Computing, Information and Communication*, vol. 2, pp. 507–536, December 2005.

[3] R. D. Rasmussen, "Goal-based fault tolerance for space systems using the Mission Data System," *IEEE Aerospace Conference Proceedings*, vol. 5, pp. 2401–2410, March 2001.

[4] Z.-H. Duan, Z.-X. Cai, and J.-X. Yu, "Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey," *IEEE Int'l Conference on Robotics and Automation*, pp. 3428–3433, 2005.

[5] C. Ferrell, "Failure recognition and fault tolerance of an autonomous robot," *Adaptive Behaviour*, vol. 2, no. 4, pp. 375–398, 1994.

[6] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, "A dynamic fault tolerance framework for remote robots," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 4, pp. 477–490, 1995.

[7] M. W. Hofbaur and B. C. Williams, "Hybrid estimation of complex systems," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 34, no. 5, pp. 2178–2191, 2004.

[8] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis [robot fault diagnosis]," *IEEE Robotics and Automation Magazine*, vol. 11, no. 2, pp. 56–66, 2004.

[9] Y. Diao and K. M. Passino, "Intelligent fault-tolerant control using adaptive and learning methods," *Control Engineering Practice*, vol. 10, pp. 801–817, 2002.

[10] Y. Zhang and J. Jiang, "Fault tolerant control system design with explicit consideration of performance degradation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 838–848, July 2003.

[11] G. Labinaz, M. M. Bayoumi, and K. Rudie, "A survey of modeling and control of hybrid systems," *Annual Reviews of Control*, 1997.

[12] R. Alur, T. Henzinger, and P.-H. Ho, “Automatic symbolic verification of embedded systems,” *IEEE Transactions on Software Engineering*, vol. 22, no. 3, pp. 181–201, 1996.

[13] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, “HyTech: A model checker for hybrid systems,” *International Journal on Software Tools for Technology Transfer*, 1997.

[14] K. Larsen, P. Pettersson, and W. Yi, “UPPAAL in a nutshell,” *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[15] D. Dill and H. Wong-Toi, *CAV 95: Computer-aided Verification*, ch. Verification of real-time systems by successive over and under approximation, pp. 409–422. Springer, 1995.

[16] D. Dvorak, R. Rasmussen, and T. Starbird, “State knowledge representation in the Mission Data System,” *IEEE Aerospace Conference*, 2002.

[17] F. Lu and E. Milios, “Robot pose estimation in unknown environments by matching 2D range scans,” *Journal of Intelligent and Robotic Systems*, vol. 20, pp. 249–275, 1997.

[18] L. Drolet, F. Michaud, and J. Côté, “Adaptable sensor fusion using multiple Kalman filters,” *IEEE Int’l Conference on Intelligent Robots and Systems*, vol. 2, pp. 1434–1439, 2000.

[19] J. M. Braman, R. M. Murray, and D. A. Wagner, “Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[20] J. M. Braman, R. M. Murray, and M. D. Ingham, “Verification procedure for generalized goal-based control programs,” *AIAA Infotech@Aerospace*, 2007.