Verification Procedure for Generalized Goal-based Control Programs

Julia M. B. Braman* and Richard M. Murray*
California Institute of Technology, Pasadena, CA, 91125, U.S.A.
Michel D. Ingham†
Jet Propulsion Laboratory, Pasadena, CA, 91109, U.S.A.

Safety verification of fault-tolerant control systems is 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, the development of a method for converting a goal network control program into a hybrid system is given and a process for converting logic associated with the goal network into transition conditions for the hybrid automata is developed. The resulting hybrid system can then be verified for safety in the presence of failures using existing symbolic model checkers. An example task and goal network is designed, converted to hybrid automata, and verified using symbolic model checking software for hybrid systems.

I. Introduction

By nature, autonomous robotic missions 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. To satisfy requirements for more complex missions, there is a need for a systematic way to incorporate fault tolerance in autonomous robotic control systems. One way to accomplish this is 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.

A great deal of work to date has focused on issues related to detecting and recovering from sensor failures in the control of autonomous systems.[1] 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, such as those in Ferrell[2] and Visinsky et al.[3] Another fault tolerant control architecture, ALLIANCE, is a behavior-based control system for multi-robot cooperative tasks described in Parker.[4] In ALLIANCE, the distributed control system re-allocates tasks between robots in response to failures. Although many fault tolerant control systems achieve reconfigurability, few actually change the commands given to the system. The system described in Diao and Passino[5] uses adaptive neural/fuzzy control to reconfigure the control system in the presence of detected faults, and another described in Zhang and Jiang[6] reconfigures both the control system design and the inputs to the control system, though neither adjusts the intent of the commands in response to failures.

Mission Data System (MDS) is a software control architecture developed at the Jet Propulsion Laboratory (JPL).[7] It is based on a systems engineering concept called State Analysis.[8] Systems that use MDS are controlled by goals instead of by sequences of commands. Goals directly express intent as constraints on physical states over time. By encoding the strategies for achievement of intent in the control system, MDS naturally allows more fault response options to be autonomously explored.[9]

Fault tolerant control systems are modeled in several ways, but one particularly useful method is to model them as hybrid systems. Hybrid systems consist of sets of continuous dynamics that describe the different modes of the system. Logical statements condition the transitions between these modes of operation. Much work has been done on the control of different kinds of hybrid systems.[10] When the continuous dynamics of these systems are sufficiently simple,

*Department of Mechanical Engineering
†Senior Software Systems Engineer, Flight Software Systems Engineering and Architecture Group, AIAA Senior Member

it is possible to verify that the execution of the hybrid control system will not fall into an unsafe or incorrect regime.[11] There are several software packages available that can be used for this analysis, including HyTech,[12] UPPAAL,[13] and VERITI,[14] 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 $\mathbf{A}\dot{\mathbf{x}} \leq \mathbf{b}$.[11] 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.

In this paper, MDS is used as a goal-based control architecture for a representative robotic task involving sensor failures and goal re-planning (or "re-elaboration"). A process for creating a hybrid automaton from the goals on a controllable state variable has been developed.[15] Contributions of this work are the expansion of the process to more general goal networks that may have more than one goal on a state variable at a time and the development of the process to the level that automation of most of the process is possible. This process is used on two example goal networks and the resulting hybrid automaton is verified using Hytech.

An outline of this paper is as follows. Section II summarizes important concepts of MDS which pertain to this work. Section III introduces the example task and goal networks. Section IV describes the major contribution of this work, the detailed process for converting goal networks on two state variables into hybrid automata. Section V returns to the example, discussing the hybrid automata that were created and the results of the verification of the safety of the system to sensor failures. Finally, Section VI concludes the paper and discusses future directions of research.

II. Mission Data System Overview

II.A. State Analysis

State Analysis is a systems engineering methodology that focuses on a state-based approach to the design of a system.[8] In State Analysis, the control system and the system under control are treated 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.

State Analysis is applied in the following fashion. The state variables of the system under control are identified. A model of the system under control is developed, and the 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 Ingham et al.[8]

II.B. 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. Figure 1 gives a visual representation of MDS.

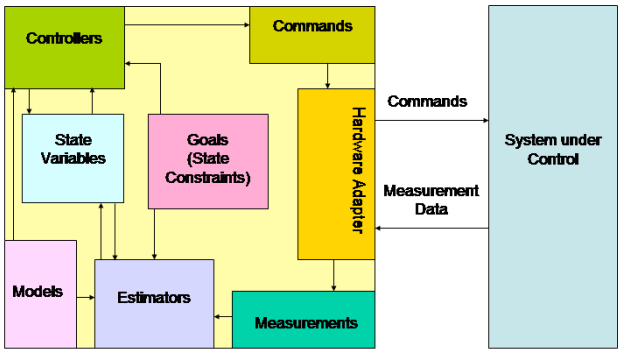

Figure 1. Graphical representation of Mission Data System

Data structures called state variables are central to MDS.[16] A state variable can contain 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. A goal network consists of a set of goals with their associated starting and ending timepoints and temporal constraints. Each goal must start at a specified timepoint and end at a different timepoint, but these timepoints can be constrained in different ways. Two timepoints can have an inflexible constraint of some finite time, can have a minimum or a maximum temporal constraint (or both), or can be unconstrained.

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. Another type of goal, called the macro goal, may elaborate goals on a state variable or state variables without being associated with any particular state variable. 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 output of the scheduler is a single sequence of executable goals on each state variable. The executable 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, if there are many ways to accomplish the same task, or if there are degraded modes of operation that are also 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. It gives the control system some ability to reason about a failure situation and attempt to recover from it.[9]

MDS is still under development at JPL and some large scale examples of its use are being studied. Current work includes using a version of MDS to control the proposed Deep Space Network Array, both in nominal operation and in the presence of failures that induce goal re-elaboration.[17]

III. 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.

III.A. 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 2, 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.

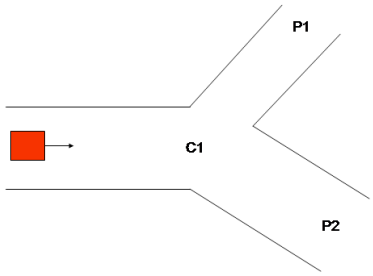

Figure 2. Simulated robotic task

III.B. 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,[18] 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. Finally, the health of the overall robotic sensing system is described by the system health derived state variable using the same discrete labels as the other health variables.[16] 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.[19] 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, 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 health state variables of the three sensors. The system's health is GOOD when the GPS sensor (the most accurate sensor) is in GOOD or FAIR health and one other sensor is not FAILED. The system health is FAIR when the GPS is the only healthy sensor (GOOD or FAIR health) or if the GPS has POOR health but the other two sensors are healthy. The system health is POOR when at least one sensor isn't FAILED, and is FAILED when all sensors are FAILED.

III.C. Goal Design

There are at least two ways to construct a goal network for the example that was introduced in Section III.A. The first way, shown in Figure 3, is to elaborate all goals from one overall maintenance goal that constrains the position of the robot to be at location p_1 or p_2 and is called BeAt1or2Goal. This goal elaborates into two goals on the robot's position, GetToC1Goal and GetToC2Goal. The first, GetToC1Goal, tells the robot's position controller to move to the first checkpoint, c_1. It also elaborates one of two tactics. The first tactic puts a constraint on the system health to be GOOD and constrains the robot's velocity to be below an upper bound, v_1. The second tactic constrains the system health to be FAIR and the robot's velocity to be below an upper bound of v_2, where $v_2 < v_1$. The second goal, GetToC2Goal, also has two tactics it can elaborate; the first tactic is GetToP1Goal and the second is GetToP2Goal. These goals tell the robot's position controller to drive the robot to the second checkpoint, which is either p_1 or p_2. GetToP1Goal and GetToP2Goal elaborate goals constraining upper bound on the robot's velocity to be v_1 and v_2, respectively. Additionally, GetToP1Goal elaborates a concurrent goal constraining the system health to be GOOD and GetToP2Goal elaborates a concurrent goal constraining the system health to be FAIR. Since the speed limits are assigned to discrete road segments and system health values, this goal network will be referred to as the local speed limit case throughout the paper.

The second way to construct a goal network for this task is shown in Figure 4. As in the first goal network, there is a goal called BeAt1or2Goal that elaborates into two goals on the robot's position, GetToC1Goal and GetToC2Goal.

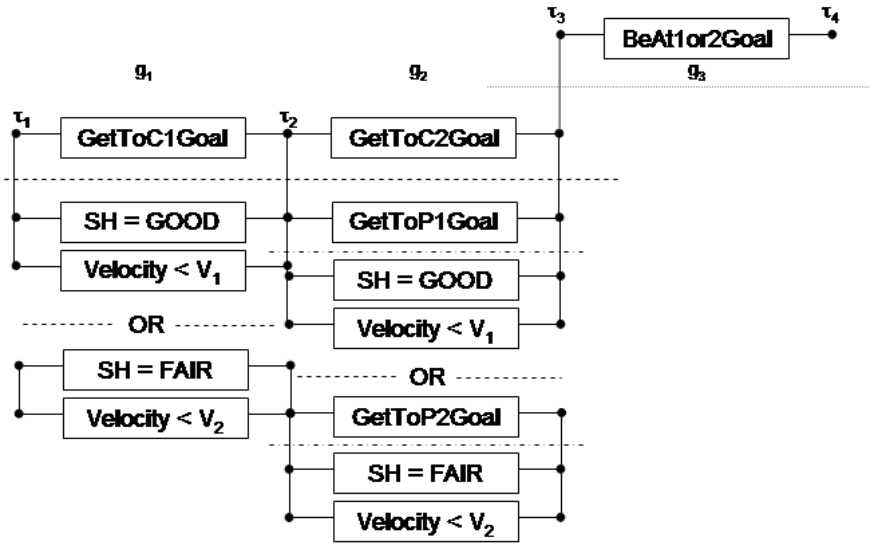

Figure 3. Goal network and elaborations for the local speed limit case; dashed lines under a goal indicate that goals beneath the line are elaborated from it with the "OR's" delineating tactics

These goals are essentially the same as in the last example except that no constraints on the robot's velocity are elaborated from them or their children. GetToC1Goal only elaborates to one tactic, which is simply a goal constraining the system health to be FAIR or better. GetToC2Goal elaborates two tactics, GetToP1Goal and GetToP2Goal, however, these goals only elaborate the constraints on the system health state variable.

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 bound 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. The extension of the macro goal to end concurrently with the end of BeAt1or2Goal introduces concepts that are beyond the scope of this paper. Since the speed limit is assigned for the entire road map, this case will be referred to as the global speed limit case throughout the paper.

IV. Verification Procedure

In this section, a process to convert a certain class of goal networks into verifiable linear hybrid automata will be discussed. It is written in a way that is conducive to the eventual automation of most of the process. The set of goal networks that can be converted using the process in this paper consists of goal networks on two state variables, one of which is controllable and the other is uncontrollable. Goals on either state variable can elaborate other goals on the same state variable and, if the state effects models allow it, goals on the other state variable. There also may be more than one goal on a state variable at a time as long as the goals are able to be merged. The goals in a network are scheduled and merged by the scheduler software component in MDS before the goal network is promoted for execution. The elaboration, merging, and scheduling will be referred to in the process below, however the details of how these are completed are outside the scope of this paper.[8]

IV.A. Conversion to Hybrid Automata

The first hybrid automaton that will be created from the goal network is based on all the goals constraining the controllable state variable. The hybrid automaton created from the uncontrollable state variable will be discussed in Section IV.B. The process to create the controllable state variable's hybrid automaton is as follows.

1. Check that all goals on the controllable state variable that are active during the same time periods are able to be merged.

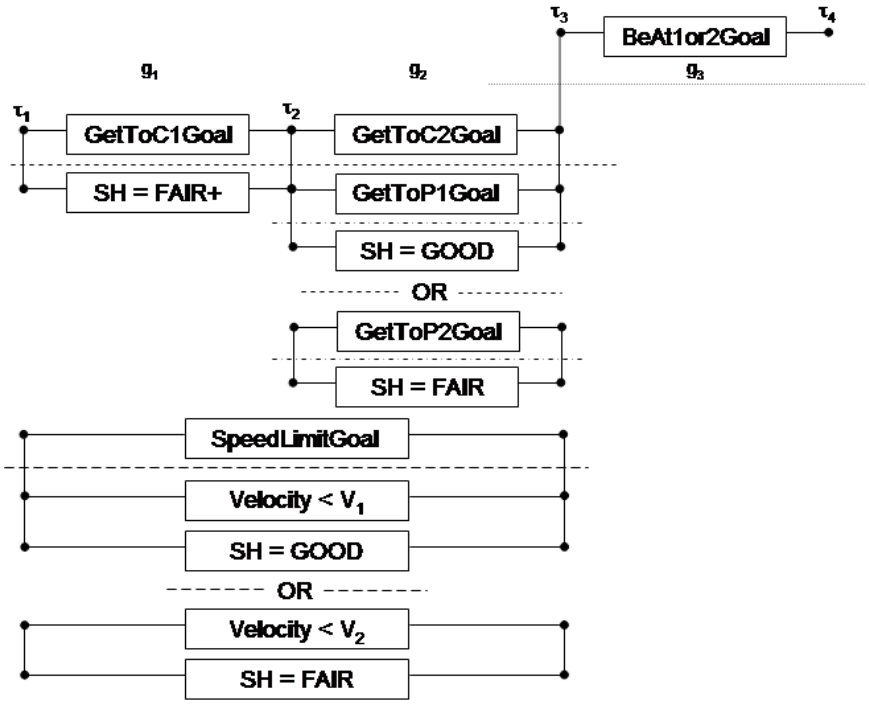

Figure 4. Goal network and elaborations for the global speed limit case; dashed lines under a goal indicate that goals beneath the line are elaborated from it with the "OR's" delineating tactics

2. Elaborate (keeping track of parent and tactic information) and "schedule" goals. Elaboration can be layered. Number each timepoint that is associated with a goal on the controllable state variable sequentially as $\tau_1, \tau_2, ..., \tau_{N+1}$, where N is the number of sets of goals between timepoints (see Figures 3 and 4).

3. Group goals between consecutive timepoints 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 $g_i, i = 1, 2, ..., N$, find all the branch goals in the elaboration tree for each root goal in the group. Branch goals are child goals on the controllable state variable that are not also ancestors of other goals on the same state variable. Root goals are goals that have no ancestors in the group. Root goals can be childless. Branch goals are combined with their ancestor goals that constrain the controllable state variable. These branch goal combinations are then combined with all other branch goal combinations from each different root goal into a location so that each root goal is represented only once in every location by itself or by one of its branch goal combinations and no two locations are the same. Locations are the modes or the discrete states of the hybrid system. Branch goals from the same root goal will never be present in the same location. If n_{ij} is the number of branch goals of the j^{th} root goal in the i^{th} group or $n_{ij} = 1$ if the j^{th} root goal is childless, then the total number of locations created in g_i will be $\prod_{j=1}^{M_i} n_{ij}$ where M_i is the number of root goals in g_i.

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

6. For each parent goal with tactics that constrain the 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, what happens when a goal in the tactic fails in the "Fail to" column, and failure conditions for concurrently elaborated goals on the uncontrollable state variable in the tactic or elaborated from the tactic, if appropriate, in the "Fail Conditions" column.

7. For each type of constraint on the controllable state variable and each possible combination of constraint types, fill out the table of transition logic between types of constraints, as outlined in Table 2. Constraint types are designed for each state variable based on the different ways the state variable will be controlled or estimated. This transition logic is associated with the isReadyToTransition function in MDS.[8]

8. For each constraint type, 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. This failure logic is associated with the isStillSatisfiable function in MDS.[8]

9. 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 more than one branch goal, 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.

10. 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 from the location to the appropriate failure location. If a goal in a location is a root goal or if it is a child of a parent goal that has only one tactic, use the goal's or the parent's failure response as an indication of the direction of the failure arrow. For each arrow, use the "Fail Conditions" elaboration logic for that tactic and the failure logic for the constraint type(s) from the transition table combined with logical 'OR' connector as the transition condition. If from any location 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."

11. 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) to the constraint type (or merged constraint type) 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. Eliminate any transition arrows whose transition conditions logically reduce to "False."

12. 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). Eliminate any transition arrows whose transition conditions logically reduce to "False."

13. 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 transition conditions that are the same as the transition conditions of an exit transition. While keeping the overall connection between the matching entry and exit transitions intact, remove all other exit transitions from that location. Remove any location that has an exit transition condition that is always "True." Keep the connection between each entry transition to the location and the destination of the "True" exit location, and remove all other exit conditions from the location.

Table 1. Outline of elaboration logic table

Tactic	Starts in	Fail to	Fail Conditions
1			
2			
:			

IV.B. Hybrid System Verification

The process in the previous section results in a hybrid automaton for the controllable state variable; the process for creating a hybrid automaton for the uncontrollable state variable is described in this section. Since the transitions between discrete or continuous states for this state variable are not controllable, they generally happen stochastically

Table 2. Outline of transition, success, and failure logic table

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

or at a given rate. This information will be used to create the hybrid automaton for the uncontrollable state variable and for setting up the verification problem.

The process outlined below details the creation of the second hybrid automaton (steps 1-3) and the set up of the verification problem (steps 4-6).

1. 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 variable's automaton, make corresponding locations in the uncontrollable state variable's hybrid automaton.
3. Introduce the appropriate (or modeled) transitions between the locations and allow the dynamics and the transitions to be controlled by rates or stochastic ranges of rates. Parameterize the transitions between the locations in the uncontrollable state variable's automaton.
4. Convert the controllable state variable's 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.
5. Synchronize the transitions between locations in the uncontrollable state variable's automaton to the transitions in the controllable state variable's hybrid automaton whose transition conditions depend on the value of the uncontrollable state variable.
6. Find "incorrect" or "unsafe" sets and search over the parameters to ensure that the hybrid system does not enter into these sets.

While the procedure for the creation of the controllable state variable's hybrid automaton could conceivably be automated, the procedure for creating the second hybrid automaton and initiating the verification is clearly not as automatic. However, this general procedure can be followed to complete the verification of general goal networks with one controllable and one uncontrollable state variable.

V. Verification Results

Returning to the example task and goal networks described in Section III, this section will describe the safety verification of these examples.

V.A. Hybrid System Design

V.A.1. Local Speed Limit Case

In this case, position is the controllable state variable and system health is the uncontrollable state variable. The goal network for this problem is shown in Figure 3. Starting with the process for the position state variable's hybrid automaton, the following steps are taken, which correspond with the numbered steps of the process in Section IV.A.

1. All goals are able to be merged since each combination of constraints on the position state variable has a legal constraint type (see Step 7).

2. Goal elaboration and labeled timepoints are shown in Figure 3.

3. There are three groups (g_1, g_2, g_3) for this goal network between pairs of consecutive timepoints, as labeled in Figure 3.

4. For g_1, GetToC1Goal has two tactics that elaborate constraints on the position state variable, which includes velocity. Since the root goal, GetToC1Goal, also constrains the position state variable, both branch goals on velocity must combine with the parent goal in the two locations that are created. These are shown in Figure 5a. For g_2, the two tactics of GetToC2Goal, GetToP1Goal and GetToP2Goal, are control constraints on the position state variable and elaborate branch goals that are velocity constraints on the position state variable. These branch goals combine with their parent goals into a location for each tactic of GetToC2Goal. Finally, for g_3, BeAt1or2Goal is a maintenance constraint on the position, and so becomes a location in the hybrid automaton. For parent goals that have more than one tactic, the child goals present in the locations are marked with the tactic number next to it in parentheses.

5. Success and Safing locations are added, as shown in Figure 5a.

6. There are two parent goals that have more than one tactic in this goal network, GetToC1Goal and GetToC2Goal, and the elaboration logic tables for each can be found in Tables 3 and 4, respectively.

7. The transition logic for the position state variable is found in Table 5. There are several types of relevant constraints on the position state variable, such as the control constraint, which tells the controller to drive the robot to the given position; the velocity constraint, which limits the maximum velocity that the robot can command; the maintenance constraint, which constrains the robot to maintain its position and zero velocity; and the combo constraint, which combines the control and velocity constraints.

8. The success and failure columns of the transition logic table can be seen in Table 5. The success logic in this case are the transition conditions from a constraint type to an unconstrained state.

9. For g_1, the "Starts in" elaboration logic from Table 3 dictates which tactic would be entered first and depends on the system health (sh) state variable's value. This is due to the concurrent constraints on the system health state variable in each tactic of GetToC1Goal. For g_2, elaboration logic from Table 4 gives the transition conditions from the group connector to each tactic, which also depend on the system health state variable due to the concurrent goals on it in the elaborations of each tactic of GetToC2Goal. In g_3, since there is only one location, the default "True" transition is to start in that location. These transition arrows and conditions are shown in Figure 5b.

10. In g_1, the elaboration logic in Table 3 is used along with the failure logic in Table 5 for the failure transitions from each location. The "Fail to" logic in Table 3 gives the direction of the arrow or arrows originating from each tactic. For both g_1 and g_2, the conditions on the failure transitions are derived from the logic in both the "Fail Conditions" column of the elaboration logic table and the failure logic in the transition logic table, though for clarity, only the elaboration logic is shown in Figure 5b. In g_2, the failure transition arrows are drawn in response to the "Fail to" logic in Table 4. For g_3, the failure response for the BeAt1or2Goal is to fail to Safing, so the failure transition arrow is drawn to that location. The conditions on the failure transition are derived from the maintenance constraint's failure condition in the "Fail" column of the transition logic table, Table 5.

11. In both g_1 and g_2, the transition arrows are drawn from each location to the group connector, as shown in Figure 5b. The transition condition logic for these transitions is derived from the transition logic table, Table 5. For the locations in g_1, the logic from the combo constraint type to the combo constraint type is used, since in this step, the constraints in each location are not considered separately, but as a merged constraint. For the locations in g_2, the logic from the combo constraint type to the maintenance constraint type is used.

12. For g_3, the transition arrow from the BeAt1or2 location to the Success location has the transition condition of "True," from the success column and the maintenance constraint row of the transition logic table, Table 5.

13. All locations in each group have entry transitions. All locations in each group have a single entry transition, but none have an exit transition with the same transition condition as the entry transition. However, there is one location, the one in g_3, that has an exit transition that is always "True." Therefore, this location is removed (along with the group connector, since it is the only location in the group). The transitions into the location are kept and redirected to the Success location, since that was the destination of the "True" transition arrow. The other exit transition from this location is removed, as shown in Figure 5b.

Table 3. Elaboration logic table for GetToC1Goal for the local speed limit case

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
2 (v_2)	if sh = FAIR	Safing	sh = POOR

Table 4. Elaboration logic table for GetToC2Goal for the local and global speed limit cases

Tactic	Starts in	Fail to	Fail Conditions
1 (*GetToP*1*Goal*)	if sh = GOOD	Safing	sh = FAIR
2 (*GetToP*2*Goal*)	if sh = FAIR	Safing	sh = POOR

Table 5. Transition logic table for the position state variable for local and global speed limit cases (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	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x \neq c$ & $v = 0$
Velocity	$v = 0$	True	$v \leq nc$	$v \leq nc$	True	$v > c$
Combo	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	$x = c$ & $v = 0$	X

The final hybrid automaton is shown in Figure 5c. The process for the system health state variable and the hybrid system verification for this example will be addressed in Section V.B.

V.A.2. Global Speed Limit Case

In the somewhat more complicated global speed limit case, many of the steps taken are the same as the steps taken in the local speed limit case. The process of creating the hybrid automaton for the position state variable is outlined below, with references to work done in the previous example where appropriate.

1. As before, all goals are able to be merged.
2. Goal elaboration and labeled timepoints are shown in Figure 4.
3. There are three groups (g_1, g_2, g_3) for this goal network between pairs of consecutive timepoints, as seen in Figure 4.
4. For g_1, GetToC1Goal has only one tactic, which elaborates to a constraint on the system health state variable and is not considered in this step. However, the SpeedLimitGoal is active in this group, and has two tactics which both elaborate branch goal velocity constraints on the position state variable. Therefore, two locations are created from the combination of GetToC1Goal and the two tactics of SpeedLimitGoal, which can be seen in Figure 6a. For g_2, the two tactics of GetToC2Goal, GetToP1Goal and GetToP2Goal, are control constraints and branch goals on the position state variable. These two tactics combine with the two tactics from the SpeedLimitGoal, creating a total of four locations. Finally, for g_3, BeAt1or2Goal is a maintenance constraint on the position and so becomes a location in the hybrid automaton.

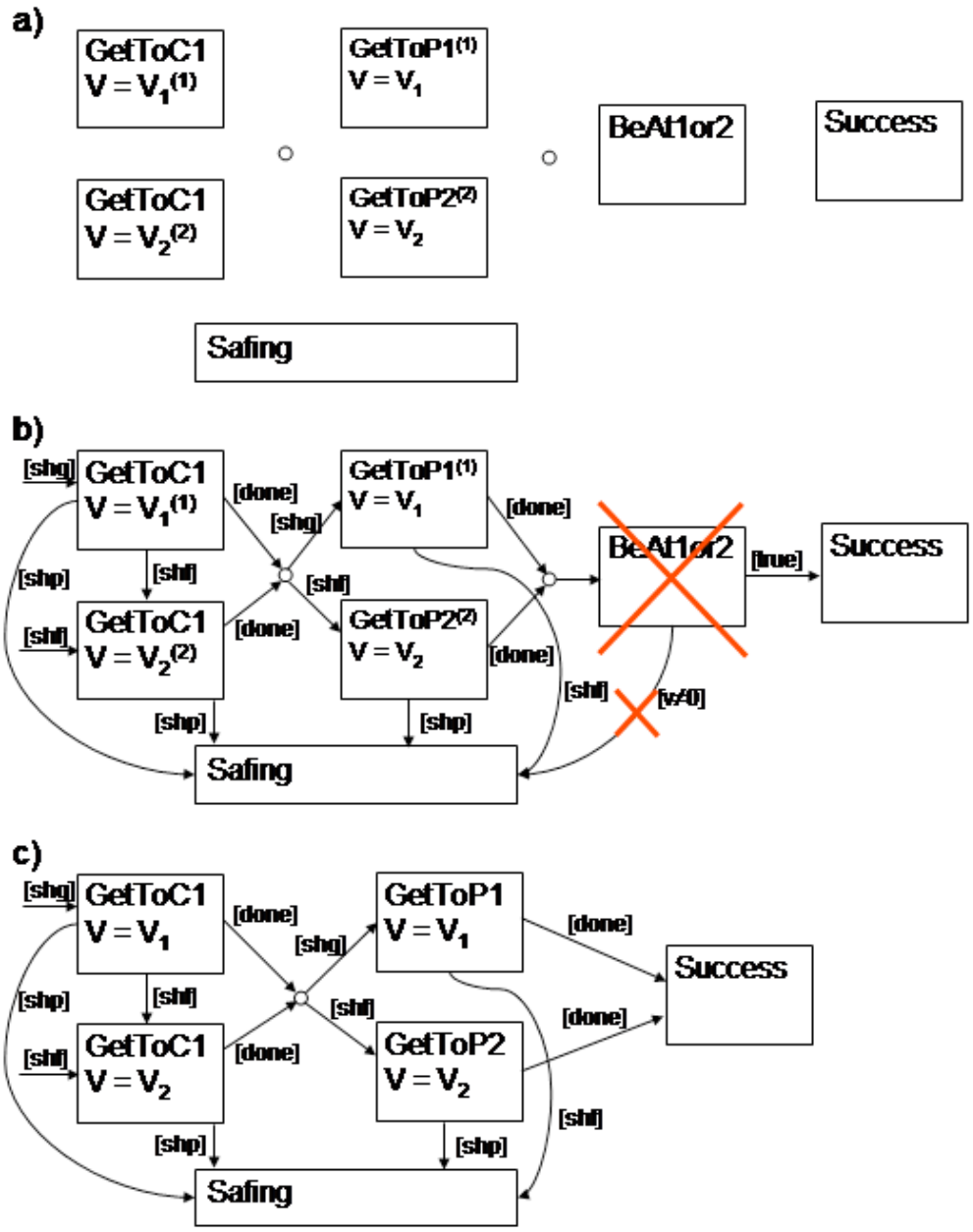

Figure 5. Creation of the position hybrid automaton for the local speed limit case: a) Location creation process (steps 3-5); b) Transition creation process (steps 9-13); c) Final position automaton (shg = system health is GOOD, shf = system health is FAIR, shp = system health is POOR, done = $[x = c \& v = 0]$ from Table 5)

5. Success and Safing locations are added, as shown in Figure 6a.

6. The two parent goals that have more than one tactic are the GetToC2Goal, whose elaboration logic is in Table 4, and SpeedLimitGoal, whose elaboration logic is in Table 6.

7. The transition logic table is the same as in the previous example and can be found in Table 5.

8. The success and failure columns of the transition logic table can be found in Table 5.

9. The "Starts in" elaboration logic from Table 6 dictates which tactic in g_1 is initially entered, and depends on the system health state variable's value due to the concurrent constraints on that state variable in each tactic. For g_2, the transition conditions from the group connector to each of the locations depends on "Starts in" elaboration logic from both Table 4 and Table 6. The transition conditions from both tables to the appropriate location are combined with a logical 'AND' connector and can be seen in Figure 6b. For two of the locations, the transition condition logic reduces to "False" because the system health is constrained to be both GOOD and FAIR; therefore those two transitions are eliminated. In g_3, since there is only one location, the default "True" transition is to start in that location.

10. In g_1, the "Fail to" elaboration logic in Table 6 is used to draw the arrows that are due to the SpeedLimitGoal tactic goals for the failure transitions from each location. For the GetToC1Goal root goal represented in each location, the failure transition arrow points to Safing due to the failure response for that goal. One of the failure transition conditions is derived from the elaborated constraint on the system health state variable; the others are due to the fail logic column and either the control or velocity constraint row in the transition logic table. For both g_1 and g_2, the conditions on the failure transitions are derived from the logic in both the elaboration logic table and the transition logic table, but for clarity, only the elaboration logic is shown in Figure 5b. In g_2, the failure transition arrows are drawn in response to the "Fail to" logic in Table 4 for the tactics of GetToC2Goal and separate failure transition arrows are drawn from the logic in Table 6 for the tactics of SpeedLimitGoal. For g_3, the failure response for the BeAt1or2Goal is to fail to Safing, so that failure transition arrow is drawn. The conditions on this failure transition come from the maintenance constraint's failure condition in the fail column of the transition logic table, Table 5.

    In both g_1 and g_2, there are failure transitions that can be combined. The two transitions from each location in g_1 that end at the Safing location can be combined and the transition conditions, which are the same in both cases, are combined using a logical 'OR' connector, which just reduces to the original transition condition. In g_2, the two failure conditions from the locations that result from the second tactic of the SpeedLimitGoal can be combined and the transition logic can be combined using a logical 'OR' connector. These simplifications can be seen in Figure 6c.

11. In both g_1 and g_2, the transition arrows are drawn from each location to the group connector, as shown in Figure 6b. The transition conditions are derived from the transition logic table, Table 5. For the locations in g_1, the logic from the combo constraint type to the combo constraint type is used for the transition conditions. For the locations in g_2, the logic from the the combo constraint type to the maintenance constraint type is used.

12. For g_3, the transition arrow from the BeAt1or2 location to the Success location has the transition condition of "True" from the success column and maintenance constraint row of the transition logic table, Table 5.

13. The location resulting from the second tactic of GetToC2Goal and the first tactic of SpeedLimitGoal in g_2 has no remaining entry conditions (the only one was eliminated in Step 9), and so it is removed along with all exit transitions from the location. Also in g_2, the location resulting from the first tactic of GetToC2Goal and the second tactic of SpeedLimitGoal has an entry transition condition that is the same as one of the exit transition conditions. Therefore, this location is removed. The entry and exit transitions that logically match are kept and become a transition from the location resulting from the first tactics of both GetToC2Goal and SpeedLimitGoal to the Safing location, as seen in Figure 6c. The other exit transitions from the deleted location are removed.

    There is one location, the one in g_3, that has an exit location that is always "True." Therefore, this location is removed (along with the group, since it is the only location in the group). The transitions into the location are kept and redirected to the Success location, since that was the destination of the "True" transition arrow. The other exit transition from this location is removed, as shown in Figure 6b.

Table 6. Elaboration logic table for SpeedLimitGoal for the global speed limit case

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
2 (v_2)	if sh = FAIR	Safing	sh = POOR

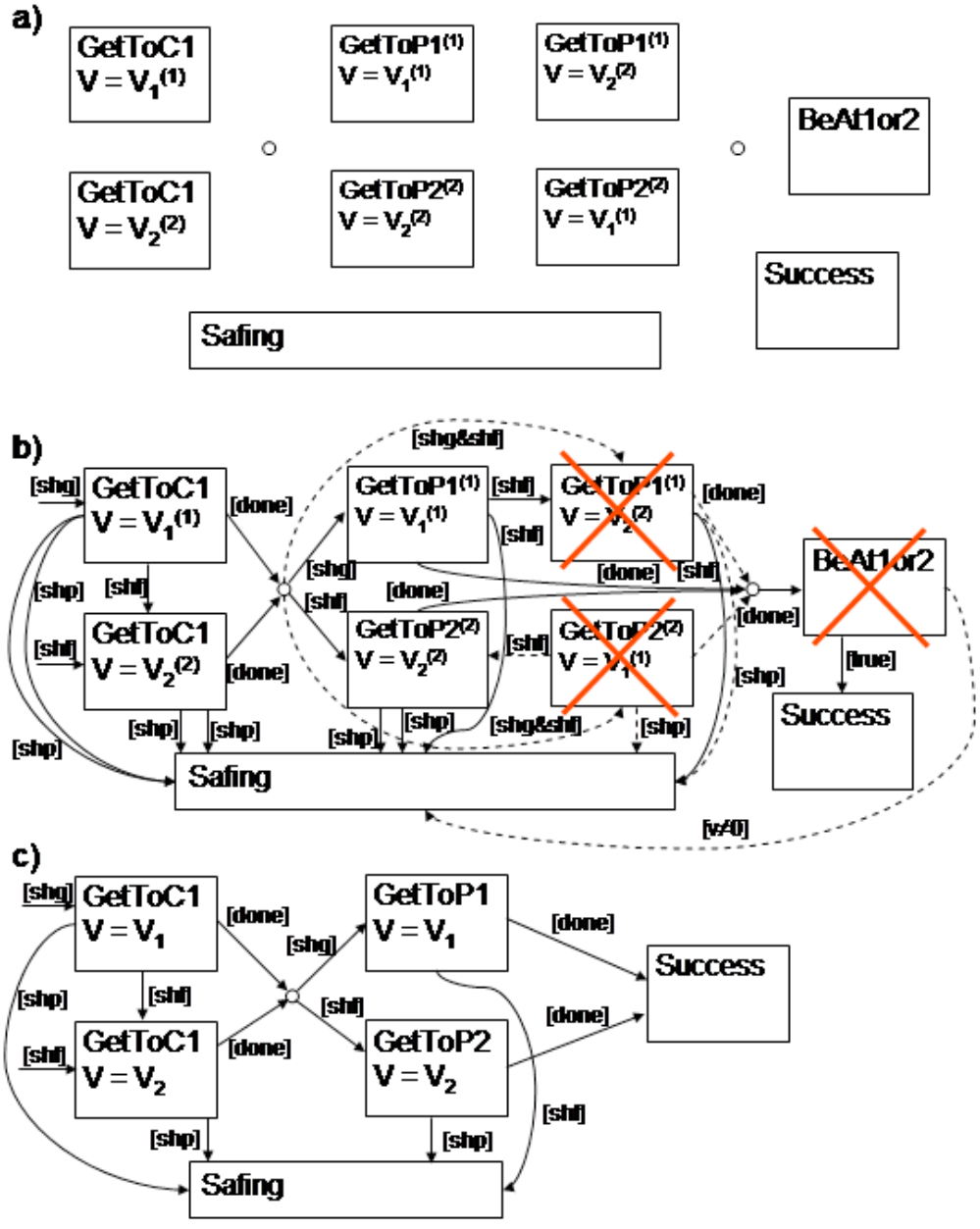

Figure 6. Creation of the position hybrid automaton for the global speed limit case: a) Location creation process (steps 3-5); b) Transition creation process (steps 9-13); c) Final position automaton (shg = system health is GOOD, shf = system health is FAIR, shp = system health is POOR, done = [x=c&v=0] from Table 5). Dashed lines indicate that the transition arrow is deleted.

The final hybrid automaton for the position state variable is shown in Figure 6c, and is the same as the hybrid automaton created from the local speed limit example, which is shown in Figure 5c. This result speaks to the correctness of this conversion process, shows that the goal networks are equivalent, and demonstrates that the verification of both goal networks can be achieved concurrently.

V.B. Results

The verification of the local and global speed limit cases starts with the creation of the hybrid automaton for the system health state variable, the transition of the automata to HyTech code, and the synchronization of the two automata. For this, the process outlined in Section IV.B is used as follows.

1. The system health state variable has discrete states (GOOD, FAIR, POOR, and FAILED).
2. From the hybrid automaton for the position state variable shown in Figure 6c, 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. The HyTech code for the system health automaton can be seen in Figure 8. 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.
4. The translation of the position state variable's hybrid automaton to HyTech code can be seen in Figure 8. The transition from the GetToC1_sl2 location (which corresponds to the location that is represented by the second tactic of either GetToC1Goal or SpeedLimitGoal in the local or global case, respectively) to the GetToP1 location was removed because the logical combination reduced to "False."
5. As shown in Figure 8, the locations of the system health automaton are synchronized with the appropriate transition conditions in the position automaton.
6. The union of "incorrect" sets is chosen to consist of the following subsets:
   (a) Position is Safe and $s < \alpha$
   (b) Position is GetToC1_sl1 and $s > \alpha$
   (c) Position is GetToC1_sl2 and $s > \beta$
   (d) Position is GetToP1 and $s > \alpha$
   (e) Position is GetToP2 and $s > \beta$

Figure 7. System health hybrid automaton

Both forward and backward analysis can be used for this verification.[11] The initial conditions start the position automaton in location GetToC1_sl1 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.

Analysis on the same set of hybrid automata using different incorrect sets gives insight into the hybrid system and into the execution of HyTech. For example, changing the incorrect set involving the position hybrid automaton being in location GetToC1_sl2 from $s > \beta$ to $s > \alpha$ disallows the system health to be FAIR when in that location, which is clearly allowed by its entry transition in Figure 6c. HyTech returns the result that there are conditions on α and β that would result in the above condition being true and shows a potential path that illustrates this faulty response. The HyTech output for this example can be seen in Figure 9. Likewise, adding a legal transition from GetToC1_sl2

to GetToP1 (upon completion of the goal but removing the necessary constraint on the system health state variable in the transition condition) can cause the verification of the system to fail. The HyTech output of this case is shown in Figure 10.

```
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 position
initially GetToC1_s11 & x=0 & y=0 ;
synclabs: Fair, Poor ;

loc GetToC1_s11: while True wait {dx in [ 1/10, 1],dy=0 }
        when x >= 6 & s >= alpha & s < beta goto GetToP2 ;
        when s < alpha & x >= 6 & s < beta goto GetToP1 ;
        when True sync Fair goto GetToC1_s12 ;
loc GetToC1_s12: while True wait {dx in [ 1/10, 1/2],dy=0 }
        when x>=6 & s>=alpha & s<beta goto GetToP2 ;
        when True sync Poor goto Safe ;
loc GetToP1: while True wait {dx=0, dy in [1/10, 1]}
        when y >= 4 & s < alpha & s < beta goto Success ;
        when True sync Fair goto Safe ;
loc GetToP2: while True wait {dx=0, dy in [-1/2, -1/10]}
        when y <= -4 & s >= alpha & s < beta goto Success ;
        when True sync Poor goto Safe ;
loc Success: while True wait {dx=0, dy=0}
loc Safe: while True wait {dx=0, dy=0}
end
```

Figure 8. HyTech code for the system health and position hybrid automata

VI. Conclusion and Future Work

This paper describes a mostly systematic way to verify goal networks using a general procedure to translate certain types of goal networks into linear hybrid systems. It may be possible to automate much, if not all, of this process. A software package specializing in the analysis of linear hybrid systems can then be used to verify the safety of this control program. The process was used successfully on two different goal networks which accomplish the same task. This result is important for the verification 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. The procedure developed in this paper does not consider using goal merging logic in the conversion, which may extend the procedure's application to more complex goal networks. The procedure could also be adjusted to more completely consider the dynamics that are involved in the controllable state variable's automaton locations. It should also be possible to extend this procedure in order to reduce some of the limitations placed on the goal nets and the analysis, such the number of state variables allowed. The analysis of goal networks with more state variables may be possible with the current procedure if consideration of the interactions between the state variables via the state effects models is included. Other extensions could involve using MDS attributes like projections based on state models in the transition conditions of the hybrid automata.

```
Safety requirement is not satisfied
 ====== Generating trace to specified target region =======
Time: 0.000
Location: good.GetToC1_sl1
     x = 0   & y = 0   & s = 1   & alpha = 1   & beta = 3
------------------------------
  VIA: Fair
------------------------------
Time: 0.000
Location: fair.GetToC1_sl2
     x = 0   & y = 0   & s = 1   & alpha = 1   & beta = 3
--------------
 VIA 1.000 time units
--------------
Time: 1.000
Location: fair.GetToC1_sl2
     10x = 1   & y = 0   & s = 2   & alpha = 1   & beta = 3
 ========== End of trace generation ==========
Conditions for a faulty system:
Number of iterations required for reachability: 4
     alpha >= 1   & alpha < beta
```

Figure 9. HyTech output for incorrect set error (Position = GetToC1_sl2, $s > \alpha$)

```
Safety requirement is not satisfied
 ====== Generating trace to specified target region =======
Time: 0.000
Location: good.GetToC1_sl1
     x = 0   & y = 0   & s = 1   & alpha = 61   & beta = 62
--------------
 VIA 60.000 time units
--------------
Time: 60.000
Location: good.GetToC1_sl1
     x = 6   & y = 0   & s = 61   & alpha = 61   & beta = 62
------------------------------
  VIA: Fair
------------------------------
Time: 60.000
Location: fair.GetToC1_sl2
     x = 6   & y = 0   & s = 61   & alpha = 61   & beta = 62
------------------------------
  VIA:
------------------------------
Time: 60.000
Location: fair.GetToP1
     x = 6   & y = 0   & s = 61   & alpha = 61   & beta = 62
--------------
 VIA 1.000 time units
--------------
Time: 61.000
Location: fair.GetToP1
     x = 6   & 10y = 1   & s = 62   & alpha = 61   & beta = 62
 ========== End of trace generation ==========
Conditions for a faulty system:
Number of iterations required for reachability: 3
     alpha < beta   & alpha >= 1
```

Figure 10. HyTech output for transition error (from GetToC1_sl2 to GetToP1)

Acknowledgments

The authors would like to gratefully acknowledge Kenny Meyer, David Wagner, Robert Rasmussen, Daniel Dvorak, and the MDS team at JPL for feedback, suggestions, answered questions, and MDS and State Analysis instruction. This work was funded by NSF and AFOSR.

References

[1]Duan, Z.-H., Cai, Z.-X., and Yu, J.-X., "Fault Diagnosis and Fault Tolerant Control for Wheeled Mobile Robots under Unknown Environments: A Survey," *IEEE Int'l Conference on Robotics and Automation*, 2005, pp. 3428–3433.

[2]Ferrell, C., "Failure Recognition and Fault Tolerance of an Autonomous Robot," *Adaptive Behaviour*, Vol. 2, No. 4, 1994, pp. 375–398.

[3]Visinsky, M. L., Cavallaro, J. R., and Walker, I. D., "A Dynamic Fault Tolerance Framework for Remote Robots," *IEEE Transactions on Robotics and Automation*, Vol. 11, No. 4, 1995, pp. 477–490.

[4]Parker, L. E., "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 2, 1998, pp. 220–240.

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

[6]Zhang, Y. and Jiang, J., "Fault Tolerant Control System Design with Explicit Consideration of Performance Degradation," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 39, No. 3, July 2003, pp. 838–848.

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

[8]Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *AIAA Journal of Aerospace Computing, Information and Communication*, Vol. 2, No. 12, December 2005, pp. 507–536.

[9]Rasmussen, R. D., "Goal-Based Fault Tolerance for Space Systems Using the Mission Data System," *IEEE Aerospace Conference Proceedings*, Vol. 5, March 2001, pp. 2401–2410.

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

[11]Alur, R., Henzinger, T., and Ho, P.-H., "Automatic symbolic verification of embedded systems," *IEEE Transactions on Software Engineering*, Vol. 22, No. 3, 1996, pp. 181–201.

[12]Henzinger, T. A., Ho, P.-H., and Wong-Toi, H., "HyTech: A Model Checker for Hybrid Systems," *International Journal on Software Tools for Technology Transfer*, 1997.

[13]Larsen, K., Pettersson, P., and Yi, W., "UPPAAL in a Nutshell," *International Journal on Software Tools for Technology Transfer*, Vol. 1, No. 1-2, 1997, pp. 134–152.

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

[15]Braman, J. M., Murray, R. M., and Wagner, D. A., "Safety Verification of a Fault Tolerant Reconfigurable Autonomous Goal-Based Robotic Control System," Submitted, 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[16]Dvorak, D., Rasmussen, R., and Starbird, T., "State Knowledge Representation in the Mission Data System," *IEEE Aerospace Conference*, 2002.

[17]Dvorak, D., Indictor, M., Ingham, M., Rasmussen, R., and Stringfellow, M., "A Unifying Framework for Systems Modeling, Control Systems Design, and System Operation," *IEEE Conference on Systems, Man, and Cybernetics*, October 2005.

[18]Lu, F. and Milios, E., "Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans," *Journal of Intelligent and Robotic Systems*, Vol. 20, 1997, pp. 249–275.

[19]Drolet, L., Michaud, F., and Côté, J., "Adaptable Sensor Fusion using Multiple Kalman Filters," *IEEE Int'l Conference on Intelligent Robots and Systems*, Vol. 2, 2000, pp. 1434–1439.