Submitted, 2009 International Conference on Software Engineering (ICSE)
http://www.cds.caltech.edu/~murray/papers/2008t_hm09-icse.html

Automatic Conversion Software for the Safety Verification of Goal-Based Control Programs

Julia M. B. Braman
braman@caltech.edu

Richard M. Murray
murray@caltech.edu

California Institute of Technology
Department of Mechanical Engineering
1200 E. California Blvd., MC 104-44
Pasadena, CA 91125

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 (MDS), developed at the Jet Propulsion Laboratory, takes a goal-based control approach. In this paper, a software algorithm for converting goal network control programs into linear hybrid systems is described. The conversion process is a bisimulation; the resulting linear hybrid system can be verified for safety in the presence of failures using existing symbolic model checkers, and thus the original goal network is verified. A moderately complex goal network control program is converted to a linear hybrid system using the automatic conversion software and then verified.

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 reconfigurability could reduce the system's effective fault tolerance.

One particularly useful way to model fault tolerant control systems is as hybrid systems. Much work has been done on the control of hybrid systems [13]. 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 [1]. There are several software packages available that can be used for this analysis, including HyTech [9], UPPAAL [14], and VERITI [5], all of which are symbolic model checkers. PHAVer, the symbolic model checker used in this paper, is a more capable extension of HyTech [7]. PHAVer is able to exactly verify linear hybrid systems with piecewise constant bounds on continuous state derivatives and is able to handle arbitrarily large numbers due to the use of the Parma Polyhedra Library. 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.

Often times, the control program used for an application is not in a form that is conducive to verification. The control program must then be transformed to an acceptable form by some suitable means. One such tool exists for the conversion of AgentSpeak, a reactive goal-based control language, into two languages: Promela, which is associated with the Spin model checker [10], and Java, for which JPF2 is a general purpose model checker [2]. Another popular approach is to create correct by design control programs from Buchi automata on infinite words [8], or from specifications and requirements stated in a restricted subset of LTL [12].

The software control architecture that the control programs in this paper are based on is Mission Data System (MDS), developed at the Jet Propulsion Laboratory [6]. MDS is based on a systems engineering concept called State Analysis [11]. Systems that use MDS are controlled by networks of 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 [15].

The main contribution of this work is formal description and analysis of a goal network conversion algorithm that is based on the procedure described in a previous work [4]. This algorithm converts goal networks into hybrid automata in a sound and complete manner; this hybrid system can then be parsed into a form that is verifiable using an existing model checking software. In Section 2, some supporting information about State Analysis, MDS, and linear hybrid systems is briefly described. In Section 3, a technical description of goal networks, the conversion and verification process, including an outline of a proof of the algorithm's soundness and completeness, and a brief overview of the software is given. A moderately complex example of the software's execution is described in Section 4, and Section 5 concludes the paper and describes some future work.

2 Background Information

2.1 State Analysis and Mission Data System

State Analysis is a systems engineering methodology that focuses on a state-based approach to the design of a system [11]. Models of state effects in the system that is under control are used for 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 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.

Goals and goal elaborations are created 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 [11]. 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, MDS, is closely related to State Analysis.

Goal networks in MDS replace command sequences in more traditional control architectures as the control input to the system. A goal network consists of a set of goals, a set of 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. 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. Each scheduled goal is then achieved by the estimator or controller of the state variable that is constrained. A goal that is achieved at some level by a controller is called a controlled goal; all other goals are passive.

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 predefined tactics. These tactics are simply different ways to accomplish the intent of the goal, and passive goals constrain the conditions in which a tactic may be executed. This capability allows for many common types and combinations of faults to be accommodated automatically by the control system [15].

2.2 Linear Hybrid Automata

There are several symbolic model checkers available that are capable of verifying linear hybrid automata. A linear hybrid automaton H consists of the following components [9]:

1. A finite, ordered list of continuous state variables, $X = \{x_1, x_2, ..., x_n\}$, and their time derivatives, $\dot{X} = \{\dot{x_1}, \dot{x_2}, ..., \dot{x_n}\}$.

2. A control graph, (V, E), where V is the set of control modes or locations of the system, and E is the set of control edges or transitions between the different modes of the system.

3. The set of invariants for each location, $inv(v)$, the set of initial conditions for each location, $init(v)$, and the set of flow conditions, which are equations that dictate how state propagates in each location, $dif(v, \dot{X})$, where $v \in V$.

4. The set of transition labels, Σ, and transition actions or reset equations, A.

This hybrid automaton specification can be illustrated using a simple example. Suppose there is an autonomous unmanned air vehicle (UAV) whose task is to fly to a point and then maintain that position for some amount of time. The speed at which the UAV can fly is determined by its system health (the combined health value of all its sensors). This system is described by two automata, H_1 and H_2, shown in Figure 1. The sets of continuous variables associated with these automata are $X_1 = \{x, t\}$, where x is the position and t is a timer, and $X_2 = \{s\}$, where s is a measure of system health; the associated time derivatives belong to the appropriate sets. The locations and transition arrows (minus the conditions) make up sets V_n and E_n, where $n = 1, 2$, respectively. The initial conditions are $x = 0$ for H_1 and $s = 0$ for H_2. The transition conditions from GS1 and GS2 to Maint are $x \geq x_{d,l} \in \Sigma_1$ (these same transition edges have reset conditions on the timer, $t = 0 \in A$); the related invariants of those two locations are $x < x_d$, where $x_{d,l} < x_d$. This means that as soon

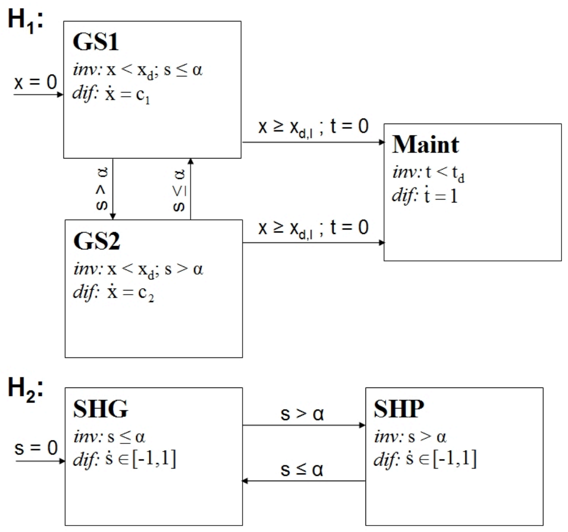


Figure 1: Hybrid automata example

as x reaches $x_{d,l}$, which may be a lower bound on the desired transition state value, the transitions can occur, but the transition will definitely occur when $x = x_d$. Likewise, the other invariant conditions and transitions in both automata dictate when the transitions will occur. Finally, the differential equations that control the flow of the variables are listed in each location.

These components fully describe a linear hybrid system that can be successfully verified using HyTech or PHAVer. The reachability analysis used in the safety verification of these hybrid automata finds the set of all states that are connected to a given initial state by a valid run. This can cause a huge explosion of the state space, however, so symbolic model checkers partition the state space into sets that are similar in the given reachability analysis. For example, given some interesting condition, PHAVer will return the set of the state space in which it is reachable; these interesting conditions given to the software are generally “unsafe” conditions for the particular system.

3 Conversion and Verification Procedure

3.1 Formal Description of Goal Networks

Let $\mathcal{G}$ be the set of all goals in a goal network, where $\mathcal{G} = G \cup U$. The set $G = \{g_1^{i_1,j_1}, g_2^{i_2,j_2}, ..., g_N^{i_N,j_N}\}$ consists of all controlled goals in a goal network, where i_n is the parent goal index and j_n is the tactic number into which the goal is elaborated, for $n = 1, ..., N$. The set of passive goals is $U = \{u_1^{i_1,j_1}, u_2^{i_2,j_2}, ..., u_M^{i_M,j_M}\}$, where i_m is the index of the goal's parent goal (which is always a controlled goal) and j_m is the goal's tactic number for $m = 1, ...M$. Let $\mathcal{T} = \{T_1, T_2, ..., T_{K+1}\}$ be the set of time points in the goal network, where $T_1 < T_2 < ... < T_{K+1}$ for increasing time. Each goal has several functions associated with it.

1. start($g_n^{i_n,j_n}$) returns the goal's starting time point.
2. end($g_n^{i_n,j_n}$) returns the goal's ending time point.
3. svc($g_n^{i_n,j_n}$) returns the state variable constrained by the goal.
4. c($g_m^{i_m,j_m}, g_n^{i_n,j_n}$) returns true if the two goals have constraints that are consistent (see Definition 3.1 below) and false otherwise.
5. cons($g_n^{i_n,j_n}$) returns the constraint (or invariant) on the state variable; cons($g_n^{i_n,j_n}$) $\in Q \times \Re$, where $\Re$ is the set of real numbers and $Q = \{=, \neq, <, >, \leq, \geq\}$.
6. entry($g_n^{i_n,j_n}$) $\in Q \times \Re$ returns the condition on the goal's constrained state variable that must be true when entering the goal.
7. exit($g_n^{i_n,j_n}$) $\in Q \times \Re$ returns the condition on the goal's constrained state variable that must be true before exiting the goal.

Definition 3.1. Two goals $g_m^{i_m,j_m}$ and $g_n^{i_n,j_n}$ are *consistent* if $svc(g_m^{i_m,j_m}) \neq svc(g_n^{i_n,j_n})$, or if $svc(g_m^{i_m,j_m}) = svc(g_n^{i_n,j_n})$ and the goals' constraints are able to be merged or executed concurrently.

Passive goals have the same functions associated with them as the controlled goals do, except the entry logic is just the goal constraint and the exit logic is always true. Since passive goals constrain the conditions in which a tactic may be executed, if those conditions become false, the tactic fails. The following two functions give elaboration logic and failure destination of each tactic, which is a group of goals with the same parent index numbers (i_n) and tactic numbers (j_n).

1. startsin(i_n, j_n) returns the condition in which the tactic is elaborated initially.
2. failto(i_n, j_n) returns the goal index to which execution goes upon failure. If the goal fails to Safing, the output of this function is “Safe.”

The following goal classifications and relationships are important in describing the execution of a goal network.

Definition 3.2. A goal is *root goal* if it has no parent goal. It is signified by $i_n = 0$ and $j_n = 0$.

Definition 3.3. Two goals $g_m^{i_m,j_m}$ and $g_n^{i_n,j_n}$ are *compatible* if $i_m \neq i_n \vee j_m = j_n$.

Definition 3.4. Two goals $g_m^{i_m,j_m}$ and $g_n^{i_n,j_n}$ are *siblings* if $i_m = i_n \wedge j_m = j_n$.

Root goals are the ancestors of all the other goals in a goal network. Two goals are compatible as long as they are not elaborated into different tactics from the same parent goals. Incompatible goals can never be executed at the same time in a goal network. Sibling goals are goals that are elaborated from a parent goal into the same tactic; sibling goals are always compatible and always executed at the same time if active during the same time points.

A valid execution of the goal network consists of a sequence of alternating flow and transition conditions,

$$\phi_{I_f}(t_f)...\phi_{I_2}(t_2)\rho_{I_2 I_1}\phi_{I_1}(t_1)X_0, \tag{1}$$

where X_0 is the set of initial conditions of the controlled state variables, $\phi_{I_n}(t_n)$ is the set of flow conditions associated with the executable set of goals θ_{I_n} and propagated forward in time t_n steps, and $\rho_{I_{n+1}I_n}$ is the transition between the executable sets of goals θ_{I_n} and $\theta_{I_{n+1}}$. Due to the structure imposed on the time points (described in the next section), goals can be grouped into K groups, G_k, $k = 1, ..., K$ where

$$G_k = \{g_n^{i_n,j_n} \in G | \\ \text{start}(g_n^{i_n,j_n}) \leq T_k \wedge \text{end}(g_n^{i_n,j_n}) \geq T_{k+1}\}. \tag{2}$$

Certain subsets of the goals in each set G_k can be executed at the same time; these subsets are called executable sets, θ_I. The set of all executable sets that are built from the goals in G_k is Θ_k. Each executable set of goals $\theta_I \in \Theta_k$ satisfies the following properties:

1. All goals in the executable set are active between the appropriate time points; for all $g_n^{i_n,j_n} \in \theta_I$, $g_n^{i_n,j_n} \in G_k$.

2. All root goals in the group are in each executable set; for all $g_n^{0,0} \in G_k$, $g_n^{0,0} \in \theta_I$.

3. If a parent goal in the executable set has children in the group, at least one of those children will also be in the executable set; for all $g_n^{i_n,j_n} \in \theta_I$, if there exists $g_m^{i_m,j_m} \in G_k, m \neq n$, s.t. $i_m = n$, then there exists $g_l^{i_l,j_l} \in \theta_I$ s.t. $i_l = n$.

4. The parent goals of all goals in an executable set are also in the set; for all $g_n^{i_n,j_n} \in \theta_I$, if there exists $g_m^{i_m,j_m} \in G_k, m \neq n$, s.t. $i_n = m$, then $g_m^{i_m,j_m} \in \theta_I$.

5. The siblings of all goals in the executable set are also in the set; for all $g_n^{i_n,j_n} \in \theta_I$, if there exists $g_m^{i_m,j_m} \in G_k, m \neq n$, s.t. $i_m = i_n \wedge j_m = j_n$, then $g_m^{i_m,j_m} \in \theta_I$.

6. Let D_n be the set of goals descended from $g_n^{0,0} \notin G_k$. Then, if $D_n \cap G_k \neq \emptyset$ then there exists $g_l^{i_l,j_l} \in D_n \cap G_k$ s.t. $g_l^{i_l,j_l} \in \theta_I$.

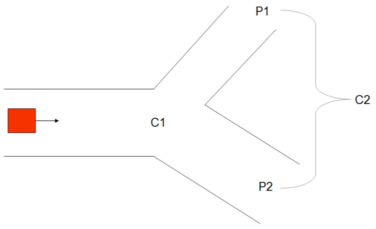


Figure 2: Path for the simple rover example

7. For all $g_n^{i_n,j_n}, g_m^{i_m,j_m} \in \theta_I$, $g_n^{i_n,j_n}$ and $g_m^{i_m,j_m}$ are compatible.

8. For all $g_n^{i_n,j_n}, g_m^{i_m,j_m} \in \theta_I$, $g_n^{i_n,j_n}$ and $g_m^{i_m,j_m}$ are consistent.

9. Let $\nu_I \subset \theta_I$ be the set of all passive goals associated with the control goals $g_n^{i_n,j_n} \in \theta_I$. Then, for all $u_n^{i_n,j_n}, u_m^{i_m,j_m} \in \nu_I$, $u_n^{i_n,j_n}$ and $u_m^{i_m,j_m}$ are consistent.

There are two different types of transitions between the goals in the goal network, transitions based on goal completion, $\rho^c_{IJ,k} \in S^{comp}$, and transitions based on goal failure, $\rho^f_{IJ,k} \in S^{fail}$, where S^{comp} and S^{fail} are the sets of all completion and failure transitions, respectively. The transition $\rho^c_{IJ,k}$ is from $\theta_I \in \Theta_k$ to $\theta_J \in \Theta_{k+1}$, and

$$\rho^c_{IJ,k} = \bigwedge_{\theta_I} \text{exit}(g_n^{i_n,j_n}) \wedge \\ \bigwedge_{\theta_J} \text{entry}(g_m^{i_m,j_m}) \wedge \bigwedge_{\theta_J} \text{startsin}(g_m^{i_m,j_m}). \tag{3}$$

The transition $\rho^f_{IJ,k}$ is from $\theta_I \in \Theta_k$ to $\theta_J \in \Theta_k$. Let F be the set of failing accompanying goals in executable set θ_I and let $H = \{\forall u_n^{i_n,j_n} \in F | \text{failto}(i_n, j_n)\}$. Then, if Safe $\in H$,

$$\rho^f_{ISafe,k} = \bigwedge_{F} \neg\text{cons}(u_n^{i_n,j_n}) \wedge \bigwedge_{\nu_I \backslash F} \text{cons}(u_m^{i_m,j_m}). \tag{4}$$

Else,

$$\rho^f_{IJ,k} = \bigwedge_{F} \neg\text{cons}(u_n^{i_n,j_n}) \wedge \\ \bigwedge_{\nu_I \backslash F} \text{cons}(u_m^{i_m,j_m}) \wedge \bigwedge_{\nu_J} \text{cons}(u_l^{i_l,j_l}). \tag{5}$$

For both $\rho^c_{IJ,k}$ and $\rho^f_{IJ,k}$, only transition conditions that evaluate to true are taken. Valid transitions are all those that are not invariantly false.

A simple example to illustrate these concepts involves a robot with three sensors traversing a path, shown in Figure 2, to get to one point of interest, whose selection depends on

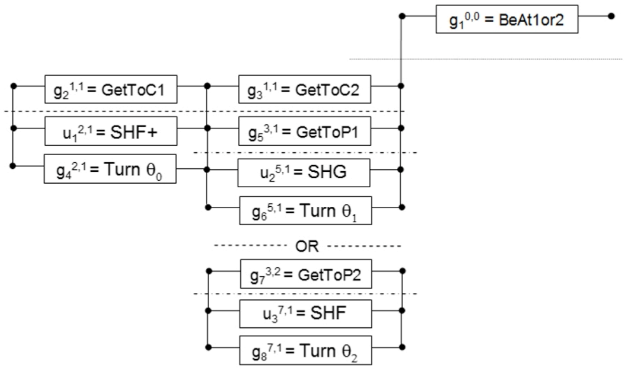


Figure 3: Goal network for the simple rover example. The circles represent time points and the rectangles represent goals. Child goals elaborated from a parent goal are under the dashed line that is below the parent goal and the tactics are separated by the "OR."

Table 1: Select Goal Function Outputs for Simple Rover Example

Goal	svc	cons	entry	exit
$g_1^{0,0}$	Pos (rate)	$(=, 0)$	$(=, 0)$	True
$g_2^{1,1}$	Pos	$(=, C1)$	True	$(=, C1)$
$g_4^{2,1}$	Ori	$(=, \theta_0)$	True	$(=, \theta_0)$
$g_5^{3,1}$	Pos	$(=, P1)$	True	$(=, P1)$
$g_7^{3,2}$	Pos	$(=, P2)$	True	$(=, P2)$
$u_1^{2,1}$	SH	$(\neq,$ P)	$(\neq,$ P)	True
$u_2^{5,1}$	SH	$(=,$ G)	$(=,$ G)	True

the health of the sensors. The state variables in this problem are as follows: robot position, robot orientation, and system health, which depends only on the values of the three sensor healths. Figure 3 depicts the goal network used to control the rover and Table 1 has the function outputs for some goals; goals that are similar to one listed are not shown. The rover can traverse the first segment of the path as long as the composite system health is fair or good. The rover then decides to go to P1 or P2 based on the system health; it picks P1 if the health is good and P2 if the health is fair. If the health at any time is poor, the robot safes due to the failto functions of all tactics. The startsin logic for all tactics is related to the accompanying passive goals.

3.2 Procedure Description

Hybrid system analysis tools can be used to verify the safe behavior of a hybrid system; therefore, a procedure to convert goal networks into hybrid systems is an important tool for goal network verification. A manual process for converting certain types of goal networks is described in [4] and [3]. These goal networks can have several state variables and several layers of goal elaborations, however time points must be well-ordered, which means the time points fire in the order that they are listed in the elaboration. This restriction only states that the goal network has already been scheduled and that goals cannot switch order; it gives no restriction on the amount of time between time points. For the software, one more restriction is currently necessary; no goals with non-trivial exit conditions can be split by a time point due to the way that the exit condition must be handled.

Each state variable constrained in the goal network is labeled as either controllable, uncontrollable, or dependent. A controllable state variable (CSV) is directly associated with a command class; an example of a CSV is a heater switch that can be commanded on and off. An uncontrollable state variable (USV) is not associated with a command class in any way; an example of a USV is wind velocity. A dependent state variable (DSV) has model dependencies on at least one controllable state variable; an example of a dependent state variable could be the temperature of an instrument, if there are ways to affect that temperature (heater, powering off the instrument).

The first automaton created in the conversion is called the "goals" automaton. This automaton has execution paths of the form

$$\psi_{I_f}(t_f)\tau_{I_f I_{f-1}}...\psi_{I_2}(t_2)\tau_{I_2 I_1}\psi_{I_1}(t_1)X_0 \qquad (6)$$

where X_0 is the set of initial conditions on the controlled state variables, $\psi_{I_n}(t_n) = \text{dif}(v_{I_n}, X)(t_n)$ is the flow associated with location v_{I_n} for t_n time steps, and $\tau_{I_n I_{n-1}}$ is the transition from location v_{I_n} to $v_{I_{n-1}}$. First, some definitions are listed.

Definition 3.5. A *branch goal* is a goal $g_n^{i_n,j_n} \in G_k$ such that for all $g_m^{i_m,j_m} \in G_k, i_m \neq n$; in other words, it is a goal that has no child goals in the same group as itself.

Definition 3.6. Two locations, v_{I_n} and v_{I_m}, are *compatible* if for all $g_n^{i_n,j_n} \in v_{I_n}$ and for all $g_m^{i_m,j_m} \in v_{I_m}$, $g_n^{i_n,j_n}$ and $g_m^{i_m,j_m}$ are compatible.

There are four sets of procedures used to create the goals automaton. First, locations of the goals automaton are created. For each group of goals G_k, $k = 1, ..., K$, a group of locations V_k is created using these procedures:

1. Let $V_k = \{v_{I_1}, v_{I_2}, ..., v_{I_B}\}$ where B is the number of branch goals in G_k, $v_{I_n} = \{g_{b_n}^{i_{b_n},j_{b_n}} | g_{b_n}^{i_{b_n},j_{b_n}}$ is a branch goal$\}$, and $I_n = \{b_n\}$.

2. For all $g_n^{i_n,j_n}, g_m^{i_m,j_m} \in G_k$ s.t. $g_n^{i_n,j_n} \in v_{I_n}$ and $g_m^{i_m,j_m} \in v_{I_m}$, if the goals are compatible, $i_m = i_n \wedge j_m = j_n \wedge n \neq m$, then let $v_{I_l} = v_{I_m} \cup v_{I_n} = \{g_m^{i_m,j_m}, g_n^{i_n,j_n}\}$, where $I_l = \{m, n\}$ and remove v_{I_n} and v_{I_m} from V_k.

3. For all $v_{I_n} \in V_k$ and for all $g_m^{i_m,j_m} \in v_{I_n}$:

   (a) If there exists $g_a^{i_a,j_a} \in G_k$ s.t. $a = i_m$ then $g_a^{i_a,j_a} \in v_{I_n}$.
   (b) If there exists $g_a^{i_a,j_a} \in G_k$ s.t. $i_a = i_m \wedge j_a = j_m \wedge a \neq m$ then $g_a^{i_a,j_a} \in v_{I_n}$.
   (c) If there exists $g_a^{0,0} \in G_k$ then $g_a^{0,0} \in v_{I_n}$. This step is not needed, since all root goals will be added to the location with the previous two steps.

4. Combine compatible locations using the following procedure:
   (a) Let $V_k^i, i = 1$, be the set of original locations.
   (b) For all $v_{I_n}, v_{I_m} \in V_k^i, m \neq n$, if v_{I_n} and v_{I_m} are compatible, let $v_{I_j} = v_{I_n} \cup v_{I_m}, v_{I_j} \in V_k^{i+1}$.
   (c) For all $v_{I_n} \in V_k^i$, if for all $v_{I_j} \in V_k^{i+1}$, $\neg(v_{I_n} \subset v_{I_j})$, add v_{I_n} to V_k^{i+1}. For all $v_{I_n}, v_{I_m} \in V_k^{i+1}, m \neq n$, if $v_{I_n} = v_{I_m}$, remove v_{I_n}.
   (d) Increment i. Repeat Steps (b)-(d) until for all $v_{I_n}, v_{I_m} \in V_k^i, m \neq n$, v_{I_n} and v_{I_m} are incompatible.
   (e) Set $V_k = V_k^i$.

5. For all $v_{I_n} \in V_k$ and for all $g_m^{i_m,j_m}, g_l^{i_l,j_l} \in v_{I_n}$, if $\neg c(g_m^{i_m,j_m}, g_l^{i_l,j_l})$, remove v_{I_n}.

6. For all $v_{I_n} \in V_k$ and for all $u_m^{i_m,j_m}, u_l^{i_l,j_l} \in v_{I_n}$, if $\neg c(u_m^{i_m,j_m}, u_l^{i_l,j_l})$, remove v_{I_n}.

The first two steps of the procedure basically set up the initial locations to contain each of the lowest level goals; if the group of goals was drawn in a tree structure with the children descended from the parent goals, these goals are the furthest branches. The next step adds all the ancestor goals and siblings of ancestor goals up the goal tree from the branch goal to each of the branch goal locations. Parent goals may be represented in many locations, but each branch goal is represented in only one location and the combination of the goals in each location is the set of all goals in the group. These first three steps guarantee that each goal in a group is represented in at least one location. From the properties of executable sets, properties 1, 2, 4, and 5 are satisfied by these three steps; all goals are in the same group, and so are active between the same consecutive time points, and all root goals, parent goals, and sibling goals of each goal in the location are also in each location.

The fourth step is the location combining procedure. The locations that are compatible are combined in such a way so that all possible combinations of compatible locations are created and so that the final outcome is a set of incompatible locations. It can be shown that this combination procedure produces all possible executable sets for each group. This step satisfies the properties 3, 6, and 7 of executable sets; each parent in a location can be shown to have at least one child goal in the location as a result of this procedure, descendants of root goals that are not in the group are present in each location because they are compatible with the root goals (and descendants) that are in the group, and due to construction, all goals in the location are compatible. Finally, the last two steps remove locations that have inconsistent goals. This satisfies properties 8 and 9 of executable sets.

Transitions for the goals automaton are created using three different procedures, one for each type of transition condition. The first two are based on the completion transitions in the goal network. First, the procedure for creating transitions from the preceding group connector (or initially for the first group, V_1) to each location is as follows, for all $V_k, k = 1, ..., K$:

1. For all $v_{I_n} \in V_k$, transition edges are created from the preceding group connector (or initially for $k = 1$) to the location.

2. Transition conditions for each edge are created,
$$\tau_{n,k}^s = \bigwedge_{v_{I_n}} \text{entry}(g_m^{i_m,j_m}) \wedge \bigwedge_{v_{I_n}} \text{startsin}(g_m^{i_m,j_m}). \quad (7)$$

3. For all $g_m^{i_m,j_m} \in v_{I_n}$, if svc$(g_m^{i_m,j_m})$ returns a discrete controllable state variable, cons$(g_m^{i_m,j_m})$ is added to $\tau_{n,k}^s$ as a reset action.

4. If $\tau_{n,k}^s$ is invariantly false, the corresponding transition edge is deleted.

The procedure for creating exit transitions from the locations to the following group connector (or to the Success location for $k = K$) is as follows, for all $V_k, k = 1, ..., K$:

1. For all $v_{I_n} \in V_k$, the transition edges are created from the location to the following group connector (or to the Success location if $k = K$).

2. Transition conditions for each edge are created,
$$\tau_{n,k}^e = \bigwedge_{v_{I_n}} \text{exit}(g_m^{i_m,j_m}). \quad (8)$$

3. If $\tau_{n,k}^e$ is invariantly false, the corresponding transition edge is deleted.

Finally, the procedures for creating failure transitions between locations within groups is as follows, for all $V_k, k = 1, ..., K$:

1. For all $v_{I_n} \in V_k$, let $U_n = \{F_1, F_2, ...\}$ where F_m are all possible combinations of $u_l^{i_l,j_l} \in A_n$, where set A_n contains all unique $u_l^{i_l,j_l} \in v_{I_n}$ (repeated constraints are excluded).

2. For all $F_m \in U_n$, let $f_m = \{\forall u_l^{i_l,j_l} \in v_{I_n}, \text{failto}(i_l, j_l)\}$.

3. For all $v_{I_n} \in V_k$ and for all $F_m \in U_n$, if Safe $\in f_m$, create a transition edge from v_{I_n} to the Safing location. The transition condition associated with the edge is

$$\tau^f_{nSafe,k} = \bigwedge_{F_m} \neg\text{cons}(u_l^{i_l,j_l}) \wedge \bigwedge_{A_n \backslash F_m} \text{cons}(u_l^{i_l,j_l}). \tag{9}$$

4. For all $v_{I_n} \in V_k$ and for all $F_m \in U_n$, if Safe $\notin f_m$,

$$\tau^f_{na,k} = \bigwedge_{F_m} \neg\text{cons}(u_l^{i_l,j_l}) \wedge \bigwedge_{A_n \backslash F_m} \text{cons}(u_l^{i_l,j_l}) \wedge \bigwedge_{A_a} \text{cons}(u_l^{i_l,j_l}). \tag{10}$$

   If $\tau^f_{na,k}$ is not invariantly false, create a transition edge from v_{I_n} to v_{I_a} whose transition condition is $\tau^f_{na,k}$.

5. Remove any transition edge whose condition, $\tau^f_{nSafe,k}$ or $\tau^f_{na,k}$ is invariantly false.

The first two steps create all possible sets of failure conditions for a given location and then the set of locations to which the execution continues for each of these sets of failure conditions. The third and fourth steps create the transition and condition depending on the contents of the "fail to" set. If Safe is not in the set, then the location that the failure transition goes to depends on which location's passive goal conditions agree with the failure conditions. Finally, the last step removes any invariantly false transition. This concludes the procedure to create the goals automaton.

Next, separate hybrid automata are created for each USV and DSV in the following way. These automata drive state transitions for the state variables that are not propagated by the flow equations in the goals automaton's locations.

1. The locations of each automaton are created from the discrete states or discrete sets of states that the state variable is constrained to be in the goal network if there are passive goals on that state variable or if the state variable is discrete. If not, then the locations are based on the different rates of change that the variable can have.

2. The transitions between the locations are based on the model of the state variable; the transitions may be modeled as stochastic if they are uncontrollable or dependent on something that is not modeled, such as time of day. The transitions may be based on state variables on which there are model dependencies, and the transition conditions are derived from those models.

Once the hybrid system is created, the verification work begins.

1. Specify the unsafe set. This is what the hybrid system is verified against; when the system is said to be "verified," that means that the unsafe set is not reached.

2. Run the hybrid system with the unsafe set through model checking software; currently, PHAVer is the default symbolic model checking software used.

3. Make and record any changes needed to verify the hybrid automaton. Translate these changes into adjustments to the goal network.

3.3 Soundness and Completeness

It is possible to prove that this conversion procedure is a bisimulation between the goal network and the goals automaton. By construction, the locations of the hybrid automaton correspond exactly to the executable sets of the goal network, and the transitions of the goals automaton are exactly those of the goal network. The following two lemmas are used to show that the locations of the goals automaton correspond one to one with all of the executable sets of the goal network.

Lemma 3.7. *For all $\Theta_k, k = 1, ..., K$ and for all $\theta_I, \theta_J \in \Theta_k, I \neq J$, θ_I is incompatible with θ_J.*

Lemma 3.8. *If there exists $v_{I_r} \in V_k$ s.t. there exists $g_n^{i_n,j_n} \in v_{I_r}$ where there is a $g_m^{i_m,j_m} \in G_k, i_m = n$ but for all $g_l^{i_l,j_l} \in v_{I_r}, i_l \neq n$ then there exists $v_{I_s} \in V_k$ s.t. v_{I_r} is compatible with v_{I_s}.*

Lemma 3.7 says that each executable set of goals is incompatible with every other executable set. This just means that each executable set of goals in a group contains at least one different tactic from a common parent goal than every other executable set in the group. The proof is by contradiction; one can show using the properties of executable sets that if two executable sets are compatible, they are the same set. Lemma 3.8 says that if a location in the goals automaton contains a parent goal but none of the parent goal's children, that location will be compatible with another location in the group. Because of the construction procedure that combines locations until all in a group are incompatible, this lemma shows that the locations satisfy property 3 in the executable set specifications. The proof is by induction; one can show that this lemma is true in the initial set of locations V_k^1 and then that is also true in all following sets.

The following proposition uses the lemmas to show that all executable sets are represented one to one by locations. It is easy to see from this proposition that the flow conditions $\phi_{I_n} = \psi_{I_n}$ for corresponding executable sets and locations.

Proposition 3.9. *For all $\Theta_k, k = 1, ..., K$ and for all $\theta_I \in \Theta_k$, there exists $v_{I_n} \in V_k$ s.t. $\theta_I \equiv v_{I_n}$.*

The proof of Proposition 3.9 uses the procedure steps for the location creation and the two previous lemmas to show that all of the executable set properties are satisfied by the locations that result from the location creation procedure. Likewise, the two following lemmas relate the transitions of the goal network to the transitions of the hybrid automaton and the proofs are also by construction using the transition creation procedures.

Lemma 3.10. *For all $\rho^c_{ij} \in S^{comp}$, there exists an equivalent transition $\tau_{ij,k} \in \Sigma^c_k = \Sigma^e_k \times \Sigma^s_{k+1}$.*

Lemma 3.11. *For all $\rho^f_{IJ,k} \in S^{fail}$ and for all $\rho^f_{ISafe,k} \in S^{fail}$ in the goal network, there exists an equivalent transition $\tau^f_{IJ,k} \in \Sigma^f_k$ and $\tau^f_{ISafe,k} \in \Sigma^f_k$ in the resulting hybrid automaton.*

Finally, the Lemma 3.12 proves that the transitions are the same between the goal network and the goals automaton and Theorem 3.13 proves that the conversion procedure is a bisimulation.

Lemma 3.12. *All transitions of the goal network are represented in the goals automaton.*

Proof. Since only two types of transitions are allowed in the goal network, this statement is true due to Lemmas 3.11 and 3.10. □

Theorem 3.13. *The conversion procedure is a bisimulation between the goal network and the goals automaton.*

Proof. By Proposition 3.9 and Lemma 3.12, all executions of the goal network are represented by paths in the hybrid automaton constructed from the goal network by using the conversion procedure. Because of this, all executions of the goal network are represented by an execution path through the hybrid automaton. There are no executions in the hybrid automaton that do not represent an execution of the goal network because each transition and location in the hybrid automaton was constructed from a corresponding transition and executable set of goals in the goal network. □

Therefore, the conversion procedure is sound in that if the hybrid automaton is verified for some unsafe set, the goal network is also verified. This is easy to see since every execution path in the goal network is represented in the hybrid automaton; so, if there exists a path in the goal network in which the given unsafe set is reachable, that path will also be present in the hybrid automaton. The conversion procedure is also complete, in that if the goal network is verifiable, the hybrid automaton will also be verifiable. There are no extra execution paths in the hybrid automaton

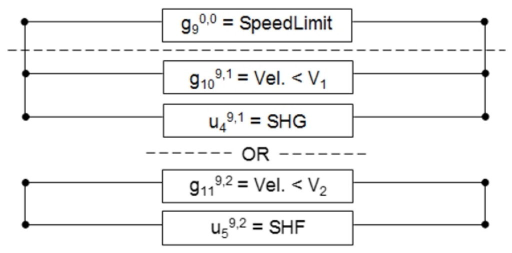


Figure 4: Speed limit goal network. Time points 1 and 3 are shown.

that are not present in the goal network; in fact, there is a way to rebuild the original goal network and goal logic from the hybrid automaton.

3.4 Simple Rover Example

The conversion and verification procedure can be illustrated using an extension of the simple rover example introduced in Section 3.1. In this version, a speed limit goal with two tactics (two speed limits) is added; the speed limit is chosen by the system health value, much like the example in Section 2.2. This extension to the goal network is depicted in Figure 4. The same state variables are used in this example, and both position and orientation are controllable state variables and the system health state variable is uncontrollable. The startsin logic of both speed limit tactics are based on the accompanying passive goals. The failto location depends on the system health value; if poor, both tactics fail to Safe, but if fair for the first tactic or good for the second tactic, the tactics fail to the other tactic. All control goal combinations in the goal network are consistent.

The goal network has four time points and therefore three groups. The first group has three sets of sibling branch goals ($\{g_4^{2,1}, u_1^{2,1}\}$, $\{g_{10}^{9,1}, u_4^{9,1}\}$, and $\{g_{11}^{9,2}, u_5^{9,2}\}$ from steps 1 and 2 of the location creation algorithm) that combine to form two incompatible locations in step 4 of the location creation algorithm, created from the combination of the one tactic of $g_2^{1,1}$ ($\{g_2^{1,1}, g_4^{2,1}, u_1^{2,1}, g_9^{0,0}\}$ from step 3 of the location creation algorithm) and the two tactics of $g_9^{0,0}$ ($\{g_9^{0,0}, g_{10}^{9,1}, u_4^{9,1}\}$ and $\{g_9^{0,0}, g_{11}^{9,2}, u_5^{9,2}\}$). The second group starts with four sets of sibling branch goals that combine into four locations, which covers all possible execution paths of the goal network between those time points. However, two locations were removed due to inconsistent accompanying goals due to step 6 of the location creation algorithm ($u_2^{5,1}$ and $u_5^{9,2}$ for one location, and $u_3^{7,1}$ and $u_4^{9,1}$ for the other). The third group has only one goal, and therefore only one location.

The transitions into the locations either initially (group 1) or from the group connector are conditioned by startsin elaboration logic, which is just the accompanying passive

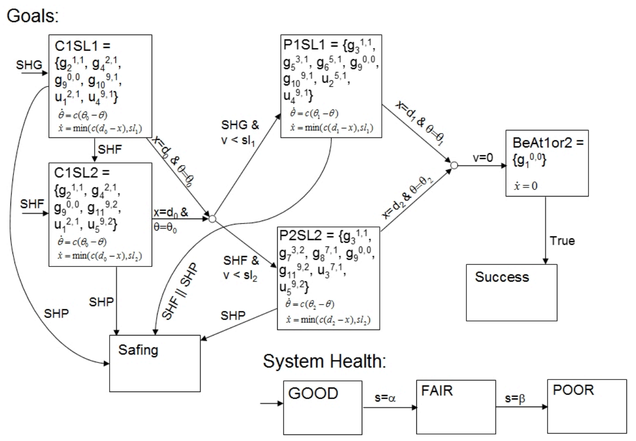

Goals:
C1SL1 = {$g_2^{1,1}$, $g_4^{2,1}$, $g_9^{0,0}$, $g_{10}^{9,1}$, $u_1^{2,1}$, $u_4^{9,1}$}
$\dot{\theta} = c(\theta_0 - \theta)$
$\dot{x} = \min(c(d_0 - x), sl_1)$
C1SL2 = {$g_2^{1,1}$, $g_4^{2,1}$, $g_9^{0,0}$, $g_{11}^{9,2}$, $u_1^{2,1}$, $u_5^{9,2}$}
$\dot{\theta} = c(\theta_0 - \theta)$
$\dot{x} = \min(c(d_0 - x), sl_2)$
P1SL1 = {$g_3^{1,1}$, $g_5^{3,1}$, $g_6^{5,1}$, $g_9^{0,0}$, $g_{10}^{9,1}$, $u_2^{5,1}$, $u_4^{9,1}$}
$\dot{\theta} = c(\theta_1 - \theta)$
$\dot{x} = \min(c(d_1 - x), sl_1)$
P2SL2 = {$g_3^{1,1}$, $g_7^{3,2}$, $g_8^{7,1}$, $g_9^{0,0}$, $g_{11}^{9,2}$, $u_3^{7,1}$, $u_5^{9,2}$}
$\dot{\theta} = c(\theta_2 - \theta)$
$\dot{x} = \min(c(d_2 - x), sl_2)$
BeAt1or2 = {$g_1^{0,0}$}
$\dot{x} = 0$
Success
Safing
SHG
SHF
SHP
SHG & v < sl_1
SHF & v < sl_2
SHF || SHP
$x=d_0$ & $\theta=\theta_0$
$x=d_1$ & $\theta=\theta_1$
$x=d_2$ & $\theta=\theta_2$
v=0
True
System Health:
GOOD
FAIR
POOR
s=α
s=β

Figure 5: Automata for rover example

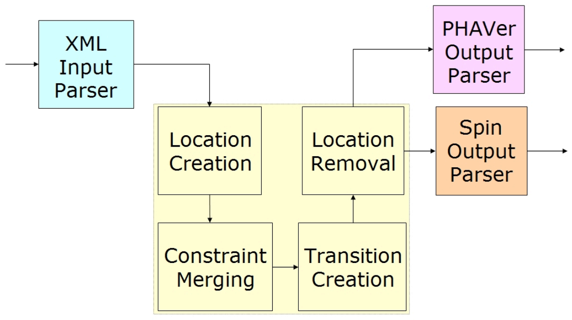


Figure 6: Flow chart of the conversion software execution

goal constraints in each tactic, and entry transition logic contributions from all goals in the group. The failure transitions between the locations in the groups or from the locations to the Safe location are due to the accompanying goals and the failto function of the goals present; if the system health becomes poor at any time, the goal network safes. The transition logic out of the locations to the following group connector or to the Success location (group 3) are the exit logic conditions for each of the completion goals present in the location (the "GetTo" goals). The final version of the goals automaton can be found in Figure 5.

The only pertinent uncontrollable state variable is the system health state variable, which can be modeled as three discrete state values, which become locations, with stochastic transitions between them. The system health automaton is shown in Figure 5. Finally, the unsafe set is determined; this is any condition that the designer decides the rover should never reach. The automata and thus the goal network can now be verified using model checking software.

3.5 Conversion Software Design

The automatic conversion software converts a given goal network when the goals, state variables, goal logic information, and the unsafe set specification are input. The output of the software is an input file for a model checking software. The software is written in Mathematica because of the list structure it employs and its extensive library of pattern-matching functions. The general outline for the structure of the conversion software is shown in Figure 6. In addition to the input and output parsers, there are four main parts to the actual conversion algorithm: location creation, constraint merging, transition creation, and location removal.

The conversion software's input parser takes an XML file with a specified structure and translates it into several lists that the Mathematica code can use. Input specifications consist of all the pertinent goal network information. A DTD file for the XML input file can be found at http://www.its.caltech.edu/ braman/files/PHAVer.txt. The four main parts of the conversion software generally follow the conversion procedure outlined in Section 3. The output of these algorithms describe the goals automaton and all of the uncontrollable and dependent state variables' automata in a very generic form so that they may be used with an output parser that translates the lists into code for any model checker that uses automata theory to verify systems. The final output is a file that can be run through the respective model checker.

The location creation algorithm and the transition creation algorithms follow the conversion algorithms presented in the previous section exactly. The constraint merging algorithm combines the constraints on all of the controlled goals that constrain the same state variable in each location. If the goals are consistent, the resulting merged constraint is found from the original constraints. This algorithm also assigns dynamical update equations to each location once the final merged constraints have been found. The location removal algorithm checks if any location lacks entry conditions and if so, removes the location and all other failure transitions originating from that location for state space considerations. The algorithm also checks for other locations that would warrant removal, however it can be shown that none of these conditions will ever occur due to the way transitions and locations are created.

4 Complex Rover Example

This example involves an autonomous rover whose ultimate goal is to follow a given path to a specified end point. This rover has three main sensor systems: GPS, LADAR, and an inertial measurement unit (IMU). The path choice and speed limit along the chosen path is dependent on the combined health of these sensors. Each sensor degrades or fails in a specified way. The GPS can experience periods of reduced accuracy (e.g. satellite dropouts) or failure (elec-

Table 2: State Variable Types

Controllable	Dependent	Uncontrollable
Position	Rel. Sun Orientation	GPS Health
Heading	LADAR Health	Ambient Temp.
IMU Power	System Health	Sun Angle
Heater Switch	IMU Temperature	
	IMU Health	

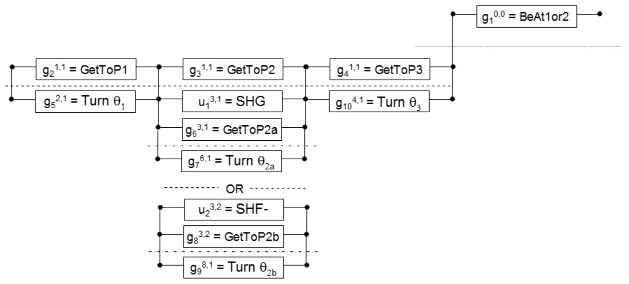


Figure 7: Path goal network. Time points 1 through 5 are present in this goal network.

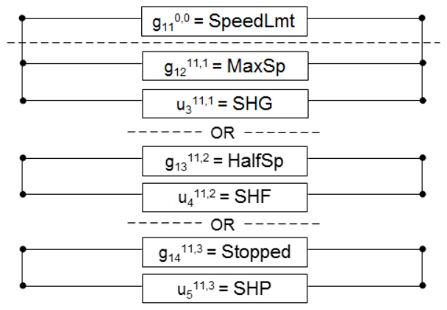
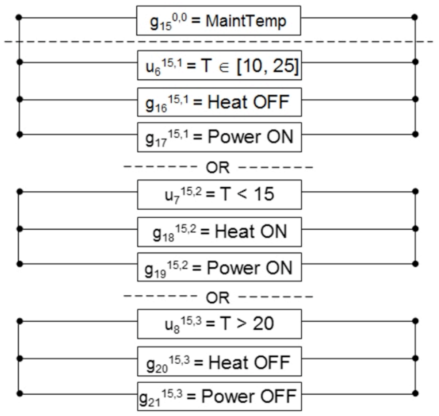


Figure 8: Speed limit goal network. Time points 1 and 5 are present in this goal network.

trical or structural signal interference), and these can both be modeled as recoverable stochastic events. The LADAR health depends on the location of the sun in the sky. If the sun is shining directly into the LADAR, its measurements cannot be used. Some degradation of the LADAR's capabilities occur at less direct sun angles, as well. Finally, the IMU health is dependent on the temperature of the device. If the temperature of the IMU is too low, a heater can be used to heat the sensor. If the IMU temperature gets too high, the unit must be powered off.

The lists of state variable types for each state variable can be found in Table 2. The goal networks for this task are shown in Figures 7-9. The first goal network describes the path the robot will take, the second is the speed limits that will apply to the robot, and the third describes the IMU temperature management method.

There are 21 controlled goals, including eight parent



$g_{15}^{0,0}$ = MaintTemp
$u_6^{15,1}$ = T ∈ [10, 25]
$g_{16}^{15,1}$ = Heat OFF
$g_{17}^{15,1}$ = Power ON
OR
$u_7^{15,2}$ = T < 15
$g_{18}^{15,2}$ = Heat ON
$g_{19}^{15,2}$ = Power ON
OR
$u_8^{15,3}$ = T > 20
$g_{20}^{15,3}$ = Heat OFF
$g_{21}^{15,3}$ = Power OFF

Figure 9: IMU Temperature goal network. Time points 1 and 5 are present in this goal network.

goals. There are four CSVs, five DSVs and three USVs. The conversion software found 46 locations (including the Success location) in four groups. In all, there are nine automata with 67 total locations. The conversion software took less than five seconds to generate the PHAVer code for this system.

The unsafe set for this problem consists of the following conditions:

1. The rover is not stopped when the IMU is off and the GPS is degraded.

2. The rover moves forward when the sun is pointing directly into the LADAR unit.

The hybrid automata could be verified after employing several reduction techniques that are outside the scope of this paper and some corrections to the automata. The verification software found reachable states in the original automata that entered the unsafe set, so the automata had to be corrected to ensure that the unsafe set was not entered. The transitions into the locations where the IMU power is off and the speed is not zero must also have a condition that the GPS Health is GOOD or FAIR to satisfy the unsafe set. These changes were added, verified, and then translated back into the goal network by adding a new tactic in the speed limit goal network, which can be found in Figure 10. This makes the control program conservative but verifiable with respect to the given unsafe set.

5 Conclusion and Future Work

The goal network conversion software presented in this paper is capable of quickly and accurately converting complicated goal networks into a bisimilar linear hybrid automata that can be verified using existing symbolic model

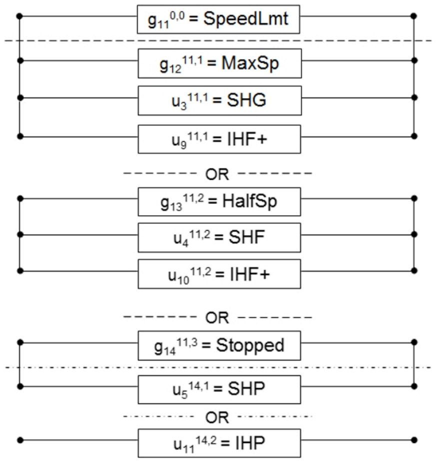


Figure 10: Redesigned speed limit goal network.

checking software such as PHAVer. This automatic tool allows more goal networks to be verified due to its speed and ease of use, which is a promising development for goal-based control programs, particularly for control architectures such as MDS.

There are many directions in which future work may go. The most natural specification of the unsafe set is in terms of constraints on state variables, while some model checking software need the unsafe set in terms of locations as well. It may be possible to automatically translate the natural specification of the unsafe set to the syntax needed by the specific model checker. The verification step of the process is not automated, but certain parts of it may have the potential to be. There are several reduction techniques that work well with the structure of the hybrid automata that is output from the conversion procedure; for example, it may be possible to verify the system group by group or to combine uncontrollable state variable automata to reduce the state space (both depend on the unsafe set specified). Also, the changes that are made to the hybrid automata during the verification procedure must be tracked and translated back into changes to the goal network, something that also may be automated. Finally, using Spin as the model checker for systems that may have multiple goal networks that run asynchronously seems like a promising direction for goal networks that have restricted state spaces. While these changes would be beneficial to this process, the software as it currently stands is a tremendous improvement over manual methods and allows more complex goal networks to be quickly converted for verification.

6 Acknowledgements

The authors would like to gratefully acknowledge Kenny Meyer, Michel Ingham, David Wagner, Robert Rasmussen, Kirk Reinholtz, and the rest of the MDS team at JPL for feedback, suggestions, answered questions, and MDS and State Analysis instruction. This work was funded by AFOSR.

References

[1] R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[2] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. *Programming Multi-Agent Systems*, volume LNAI 3067, chapter Verifiable Multi-agent Programs, pages 72–89. 2004.

[3] J. M. Braman and R. M. Murray. Conversion and verification procedure for goal-based control programs. Technical report, California Institute of Technology, 2007. CaltechCDSTR:2007.001, http://caltechcdstr.library.caltech.edu/view/.

[4] 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.

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

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

[7] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Conference on Hybrid Systems: Computation and Control*, 2005.

[8] G. D. Giacomo and M. Y. Vardi. *ECP-99*, volume LNAI 1809, chapter Automata-Theoretic Approach to Planning for Temporally Extended Goals, pages 226–238. 2000.

[9] 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.

[10] G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[11] 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*, 2(12):507–536, December 2005.

[12] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Wheres Waldo? Sensor-based temporal logic motion planning. *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

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

[14] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

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