Submitted, 2009 AIAA Infotech@Aerospace Conference and Exhibit
http://www.cds.caltech.edu/~murray/papers/2008n_bm09-infotech.html

Control Program Verification for a Sample Titan Aerobot Mission

Julia M. B. Braman* and Richard M. Murray
Dept. of Mechanical Engineering, California Institute of Technology

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. A software algorithm for converting goal network control programs into linear hybrid systems exists and 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 substantial example control program based on a proposed mission to Titan, a moon of Saturn, is converted using the procedures discussed.

I. Introduction

Autonomous robotic missions, such as robotic exploration missions to distant planets and moons, 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.

A software control architecture developed at the Jet Propulsion Laboratory uses state models and reconfigurable goal-based control programs for the control of autonomous systems.[1] Mission Data System (MDS) is based on a systems engineering concept called State Analysis.[2] Using MDS, systems 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.[3] Unlike the more traditional sequences of commands used to control robotic space missions, goal networks allow for branching in the execution plan at the cost of added complexity. The complex branching nature of goal networks make control program verification necessary.

One particularly useful way to model fault tolerant control systems, including goal networks, is as hybrid systems. Much work has been done on the control of hybrid systems.[4] 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.[5] There are several software packages available that can be used for this analysis, including HyTech,[6] UPPAAL,[7] and VERITI,[8] all of which are symbolic model checkers. PHAVer, the symbolic model checker used in this paper, is a more capable extension of HyTech.[9] 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.[11] Another popular approach is to create correct by design control programs from Buchi automata on infinite words,[12] or from specifications and requirements stated in a restricted subset of LTL.[13]

*braman@caltech.edu

The goal networks associated with MDS can also be converted automatically via a bisimulation into linear hybrid systems and then verified using existing symbolic model checking software.[14] The contribution of this paper is the conversion and eventual verification of a substantial example problem based on a proposed robotic mission to Titan, a moon of Saturn. Section II briefly describes the software architecture, MDS, linear hybrid automata, and some background information on the Titan mission. Section III summarizes the conversion and verification procedure more fully described in previous works[15].[14] Section IV describes the Titan mission plan example and Section V concludes the paper and lists future work.

II. Background Information

A. 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.[2] 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 Ingham et al.[2] 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 pre-defined 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.[3]

B. Linear Hybrid Systems

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:[6]

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 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.

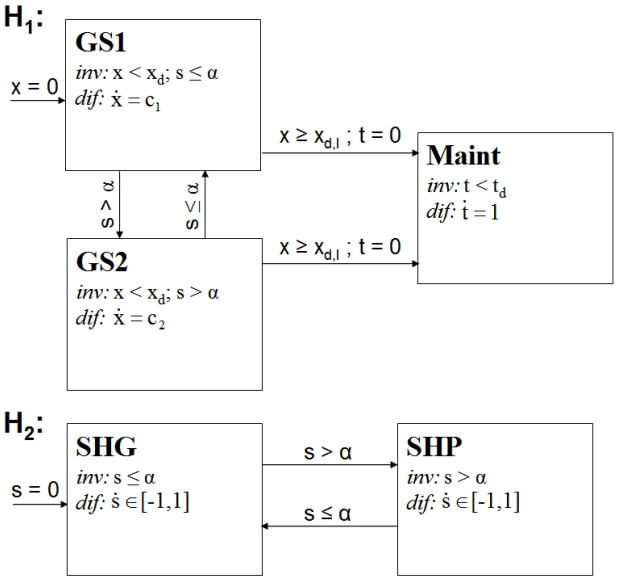

Figure 1. Hybrid automata example

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.

C. Proposed Titan Mission

Titan is the largest moon of Saturn and is remarkable for its dense atmosphere that has an estimated composition of 95% nitrogen, 3% methane, and 2% argon. The surface pressure on Titan is about 1.5 bars, which is about 1.5 times the surface pressure on Earth. The thick atmosphere and the methane haze make surface observation difficult; however near infrared observations and pictures from the Huygens probe suggest that interesting terrain is present, made of solid rock and frozen water ice littered with liquid methane and ethane bodies. Cryovolcanism has been conjectured, as well as a methane and ethane cycle like the water cycle present on Earth.[16]

A proposed mission to Titan consists of a satellite of Titan that would release an aerobot probe to the lower atmosphere of Titan. This lighter-than-air vehicle would use wind currents to explore the moon by taking advantage of Titan's unique atmosphere. The probe would have the capability to fly to points while simultaneously mapping Titan's surface; it would also be able to stationkeep. Besides wind profiling, surface

and atmospheric observations, and atmospheric composition testing, the probe would also have the capability to collect samples from the surface without landing.[17]

Because the Saturn system is far away from Earth, there is a significant light-time delay between the two planets of about 2.6 hours round-trip.[16] This means long communication latencies between the aerobot and Earth. Having an autonomous vehicle that can handle a relatively long mission plan without human interference is important. This autonomous control system must be able to run without human intervention and be able to identify and handle many types of faults and failures in a safe manner. The verification of the fault-tolerant control plans against sets of unsafe conditions will be extremely important and useful for a Titan exploration mission.

III. Verification Procedure

Goal network control programs based on State Analysis techniques and the MDS control architecture can be verified using a simple conversion procedure. This conversion from goal network control programs to linear hybrid automata (LHA), which are verifiable using existing symbolic model checking software, was introduced in Braman et al.[15] The conversion procedure is a bisimulation between the goal networks and the linear hybrid automata, and so verification of the converted LHA implies verification of the original goal network.[14]

The conversion and verification procedure of goal network control programs consists of several parts. First, the goal network itself is translated into a "goals" automaton. Some structure must be imposed on the goal network before employing this conversion; the time points in the goal network must be well-ordered (which means that they will fire in the order presented) and currently, no completion goals can be split by a time point. Completion goals are transition goals that constrain state variables to reach a certain value by applying a control input that causes the state variable to change at some rate. A simple example of this is a rover being constrained to reach a certain point; the position of the rover is controlled until the appropriate position is reached.

The goals in the goal network are then classified and grouped in different ways. All goals that are active between two consecutive time points are placed into a group. These goals are classified as controlled goals (goal constraints that require control action to achieve them) or passive goals (goals that constrain the conditions in which the tactic is valid). Consistent goals are goals whose constraints can be executed at the same time. Compatible goals are goals that either have different parent goals or are in the same tactic elaborated from a parent goal. Combinations of goals in a group can belong to the same executable set; an executable set of goals satisfies certain conditions such as compatibility and consistency. These sets enumerate all the goals that would be active during a time interval as the goal network executes; therefore they also follow ancestry rules dictated by goal network execution standards.

Locations of the linear hybrid automaton are constructed from these executable sets in each group. The three types of transitions between these locations in the goals automaton are constructed from the two types of transitions in the goal network. The two types of goal network transitions are completion transitions, where an executable set of goals transitions into another executable set of goals in the next group as a time point fires; and failure transitions, where an executable set of goals transitions into another executable set of goals in the same group due to the failure of one of the accompanying passive goals in the executable set. The three types of transitions of the linear hybrid automaton are entry transitions, which are transitions from the preceding group connector (or initially) to the location; exit transitions, which are transitions from locations to the following group connector; and failure transitions, which are transitions between locations in a group, or from a location to the Safing location. The failure transitions in the goals automaton are directly created from the goal network's failure transitions, and the entry and exit transitions are created from the completion transitions of the goal network.

The goals automaton is a linear hybrid automaton whose execution paths capture all the possible execution flows of the goal network. Once it is created, several other automata based on the state variables constrained in the passive goals must be created. Each automaton has locations based on the discrete states or the discrete sets of states that the state variable can take. The transitions of the automaton are based on the model of the state variable; the transition conditions could be stochastic or a function of other state variables.

Once the hybrid automata are created, the system can be verified against some designer-specified unsafe set of conditions using existing symbolic model checking software. PHAVer is currently the software of choice,

though the conversion algorithm software is able to create input files for a variety of model checkers. Once the hybrid automata are verified, any changes that had to be made to the hybrid system can be translated back into changes in the goal network using a reverse conversion process.

An example of the conversion of a simple goal network is shown in Figure 2. The goal network has two time points and so only one group. The parent goal is a completion goal directing the position of the system to transition to some point; there are two tactics as evidenced by the dashed line below the goal and the "OR" surrounded by dashed lines between the tactics. The first tactic constrains the speed to some limit as long as the system health (SH) is good. The second tactic constrains the speed to some smaller limit as long as the system health is fair. If the system health ever becomes poor, the system must safe.

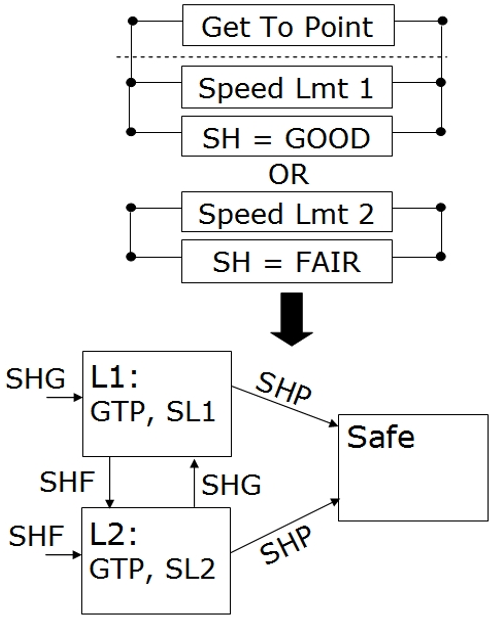

Figure 2. Example goal network conversion

The hybrid automaton that this goal network converts to has two locations in addition to the general Safing location. Each location corresponds with one of the tactics of the goal network, and the transitions between the three locations are the failure transitions based on the passive goals constraining the system health state variable. There are no exit transitions because there is only one group, and the entry transitions are based on the passive goals for each location.

IV. Titan Aerobot Example Mission

A. Problem Statement

A simplified model of the Titan aerobot was used as an example of the conversion and verification procedure. The aerobot used in this example has a mission to fly to a specific area while maintaining at least 10% power, position awareness, and appropriate safe altitudes while at the same time being aware of spontaneous science observation opportunities. The example aerobot has several sensors, including two cameras, a laser range finder (LRF), a radar, a hygrometer, and a motion sensor. The cameras and laser range finder allow the aerobot to localize and map the surface of Titan while maintaining a safe altitude above Titan's surface features. The radar, hygrometer, and motion sensor are used to detect spontaneous science events such as cloud formation, precipitation, and cryovolcanism. Figure 3 gives the state effects diagram for this example problem. The state effects diagram lists all pertinent state variables, commands, and measurements that are used to control the system. The arrows between the different bodies in the diagram indicate that the originating body has an effect on the accepting body that can be modeled.

Most of the models between state variables or between state variables and measurements or command are fairly obvious. The aerobot is able to localize using the existing map, which is generated by the orbiting

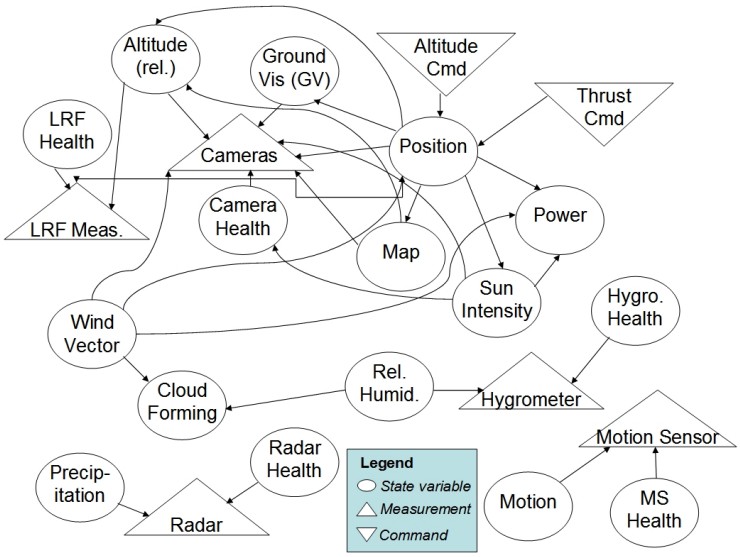

Figure 3. State effects diagram for Aerobot Example

satellite and to which details are added by the aerobot. Sunlight intensity and ground visibility are affected by the aerobot's position, and these state variables affect the quality of the measurements taken by the cameras. Relative altitude is the height that the aerobot is above the ground and is the state that the LRF is measuring. The aerobot's position is controlled by the thrust and altitude commands and affected by the wind vector. Power is also affected by the wind vector because more or less control effort may be needed to drive the aerobot based on the direction and magnitude of the wind. The aerobot is assumed to have some regenerative power capability based on solar energy, so sunlight intensity also affects the percentage of power remaining on the probe. (However, with Titan located so far from the sun, solar energy could at best be a back-up power system.) It is also assumed that altitude affects sunlight intensity, with more intensity near the top of the atmosphere.

B. Goal Networks

The goal networks for this example problem are based on the control of the position and altitude of the aerobot as it flies to a specified point. The position is controlled via three modes: a "fly to" mode where the aerobot heads towards a specified area; a "stationkeeping" mode where the aerobot maintains its current position; and a "float" mode where the aerobot drifts without controlling its position. There are also three control modes for the altitude: maintain an altitude, climb unrestricted, and fly to an altitude.

The first concurrently executed goal network, shown in Figure 4, involves the task of flying to a specified area. There are two tactics available for doing this that are chosen based on the relative wind vector. When the wind vector is favorable or small, the aerobot attempts to maintain a minimum velocity in the direction of the specified area. When the wind vector is large and unfavorable, the aerobot instead profiles the wind; in a more complex example, this technique would be used to find a new altitude at which to fly.

How well the aerobot can constrain its position on the existing map contributes to the position uncertainty; when the uncertainty is high, the aerobot must ensure that it is at a safe altitude to avoid controlled flight into terrain. The second goal network, shown in Figure 5, gives tactics that accomplish the task of simultaneous localization and mapping. SLAM continues as usual when the position and map uncertainty are low. If the position uncertainty is low and the map uncertainty is high, the aerobot flies at a lower altitude to achieve more detailed mapping. When both uncertainties are high, the aerobot ascends so as to clear all possible obstacles and to get a better view through its cameras so it can match its position with the less detailed map.

The third task for the aerobot is power management, which is controlled in the goal network shown in Figure 6. Overall, the aerobot must maintain at least 10% power; if it does not, the aerobot safes to floating until the power increases. While the power value is above 10%, there are several tactics used to ensure that

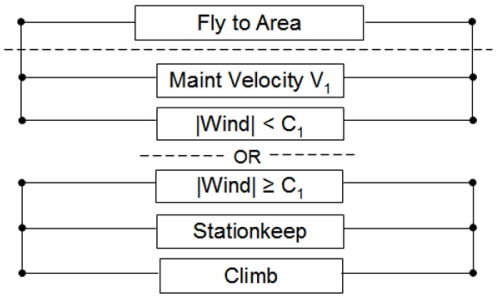

Figure 4. Goal network for flying to a specified area

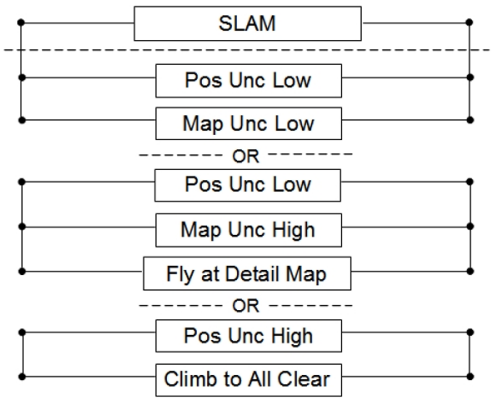

Figure 5. Goal network for simultaneous localization and mapping

the power level does not drop below the safing level. When the power drops below 50%, the aerobot climbs to increase the sunlight intensity that it is receiving. If the power dips below 30%, the aerobot discontinues its trek to the specified point and instead stationkeeps to preserve power.

Spontaneous science observation is an important part of the Titan aerobot's mission, so the goal network shown in Figure 7 deals with this task. When no motion, precipitation, or cloud formation is detected, the aerobot continues on with its current task. However, if motion on the surface (such as cryovolcanism) or precipitation is detected, the aerobot descends and stationkeeps to observe it. Likewise, if cloud formation is detected, the aerobot ascends to observe it.

Several other factors also affect the altitude at which the aerobot flies, such as the health of the position sensors (the cameras and the LRF), the ground visibility, and sunlight intensity. These conditions make up the six tactics of the final goal network controlling the altitude of the aerobot; this goal network is shown in Figure 8.

C. Conversion and Verification

All the goal networks introduced in the previous section are executed concurrently while the aerobot flies to the specified area. Some examples of safety conditions that the aerobot should always avoid are the following:

1. power $< 5\%$,
2. power $< 10\%$ and altitude less than the minimum terrain avoidance altitude for the area, and
3. ground visibility low, position uncertainty high, and altitude less than the minimum overall terrain avoidance altitude.

The total possible number of executable sets (and thus locations) is the product of the number of equivalent tactics for each concurrently executed goal network. The fly-to goal network has two tactics, the SLAM goal

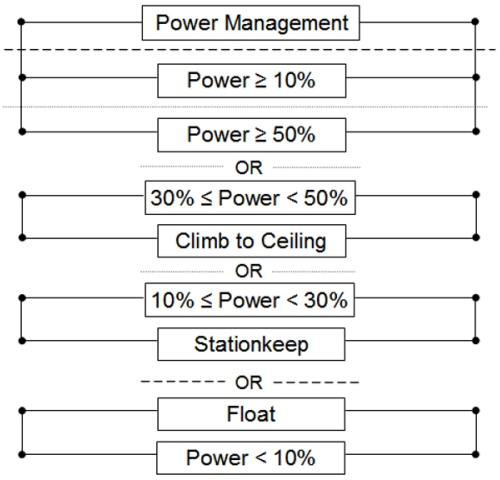

Figure 6. Goal network for power management

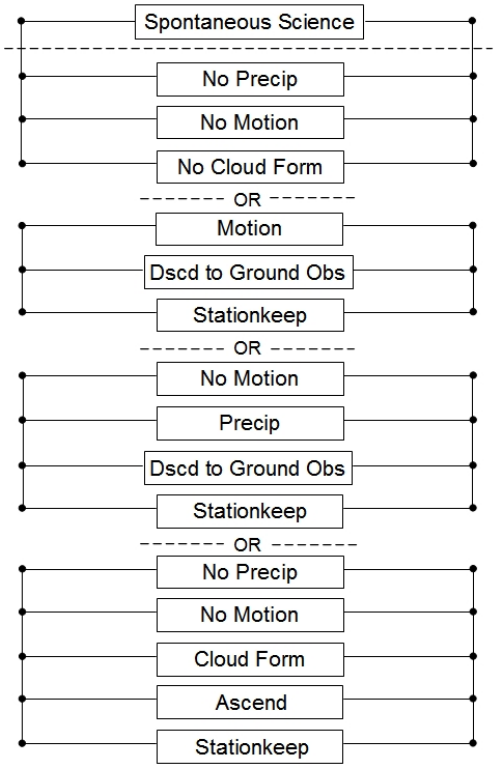

Figure 7. Goal network for observing spontaneous science

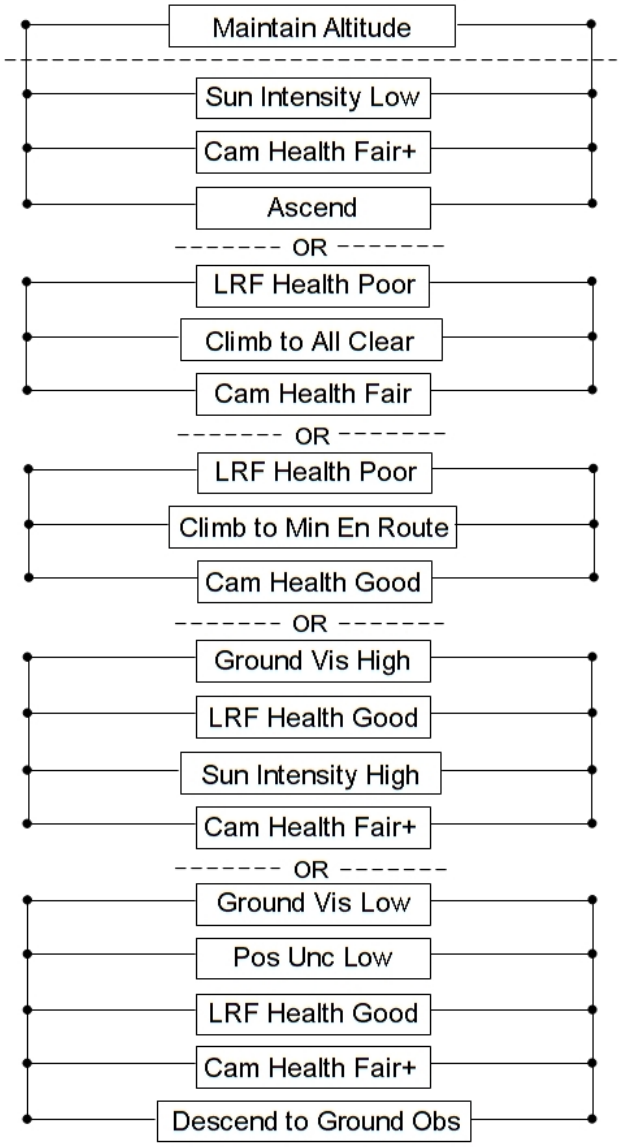

Figure 8. Goal network for controlling the altitude of the aerobot

network has three, the power management goal network and the spontaneous science goal networks each have four, and the altitude control goal network has six. The product of all these tactics is 576; however, due to some combinations having inconsistent passive goal constraints, the actual number of executable sets is 544.

Because the symbolic model checking software is not able to handle very large state spaces, the number of state variables that this problem needs was reduced by combining some state variables. For example, the motion state variable and the motion sensor health state variable were combined into a new motion state variable. This state variable has both of the functions of the previous two state variables; it is true when there is motion sensed and the health of the motion sensor is good and false otherwise. Since this is the only state that the control program cares about, this reduction is a good one. Due to these sorts of state space reductions, there are only eleven state variable constrained by passive goals in the control program. These state variables along with the number of discrete states (or discrete sets of states) constrained in the control program are given in Table 1.

Table 1. Passively Constrained State Variables

State Variable	Number of States
Camera Healths	3
LRF Health	2
Sun Intensity	2
Ground Visibility	2
Wind Vector	2
Position Uncertainty	2
Map Uncertainty	2
Precipitation	2
Motion	2
Cloud Formation	2
Power	6

The number of passive goals in each location and the number of ways that each of those passive goals can fail dictate the number of failure transitions out of each location. There is only one exit transition out of each location since there is only one group; the only exit transition goes to the Success location. The resulting hybrid system specified in PHAVer code has twelve automata and nearly 14,000 lines. The automatic conversion software was able to create the PHAVer input file from the goal network specification in less than eight hours.

However, the PHAVer verification is not as simple. Since the state space is so large for this problem, model reduction techniques must be employed. This work is currently in progress, and the results encountered so far are promising. Once the hybrid automata is verified, any changes will be translated back to the original goal network, and the control program will then be verified with respect to the unsafe set.

V. Conclusions and Future Work

The Titan aerobot mission example is a sufficiently large example to test the conversion software used for goal network control programs. Many algorithmic inefficiencies were found and corrected in the conversion software and many more remain; however, the performance of the conversion software far exceeds the performance of the symbolic model checking software for problems of this size. The verification of such large problems is a known research area, and work in model reduction and abstraction techniques is ongoing. The verification efforts of this problem are promising and will be completed to finish the verification of this goal-based control program.

Extensions on this conversion and verification procedure include finding correct model reduction techniques for abstracting the control program without losing information. The verification of goal networks is a tool for the design and use of better control programs; a step beyond verification would be to have rules or algorithms for creating correct-by-design control programs. Examples such as this Titan aerobot mission are useful to understand the dynamics of goal network control programs and can help improve the conversion

and verification processes.

Acknowledgments

The authors would like to gratefully acknowledge Kenneth Meyer, Michel Ingham, David Wagner, Robert Rasmussen, and the rest of the MDS team at JPL for feedback, suggestions, answered questions, and MDS and State Analysis instruction. The authors would also like to thank Alberto Elfes for the discussion on the Titan Aerobot mission. This work was funded by AFOSR.

References

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

[2]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.

[3]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.

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

[5]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.

[6]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.

[7]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.

[8]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.

[9]Frehse, G., "PHAVer: Algorithmic Verification of Hybrid Systems past HyTech," *International Conference on Hybrid Systems: Computation and Control*, 2005.

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

[11]Bordini, R. H., Fisher, M., Visser, W., and Wooldridge, M., *Programming Multi-Agent Systems*, Vol. LNAI 3067, chap. Verifiable Multi-agent Programs, 2004, pp. 72–89.

[12]Giacomo, G. D. and Vardi, M. Y., *ECP-99*, Vol. LNAI 1809, chap. Automata-Theoretic Approach to Planning for Temporally Extended Goals, 2000, pp. 226–238.

[13]Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J., "Wheres Waldo? Sensor-Based Temporal Logic Motion Planning," *IEEE International Conference on Robotics and Automation*, 2007, pp. 3116–3121.

[14]Braman, J. M. and Murray, R. M., "Automatic Conversion Software for the Safety Verification of Goal-Based Control Programs," Submitted, International Conference on Software Engineering.

[15]Braman, J. M., Murray, R. M., and Ingham, M. D., "Verification Procedure for Generalized Goal-based Control Programs," *AIAA Infotech@Aerospace*, 2007.

[16]Elfes, A., Montgomery, J. F., Hall, J. L., Joshi, S. S., Payne, J., and Bergh, C. F., "Autonomous Flight Control for a Titan Exploration Aerobot," *Robotics Science and Systems*, 2005.

[17]Elfes, A., Hall, J. L., Montgomery, J. F., Bergh, C. F., and Dudik, B. A., "Towards a Substantionally Autonomous Aerobot for Exploration of Titan," *IEEE International Conference on Robotics and Automation*, 2004, pp. 2535–2541.