Chapter 2

Background Information

The goal-based control systems that are used in this work are modeled after the goal networks designed for the Mission Data System (MDS) control architecture. A useful way to represent these reconfigurable control programs is as hybrid systems because model checking is then possible. An introduction to State Analysis, the design methodology upon which MDS is based, and a notational introduction to linear hybrid systems are given in this chapter, along with a brief introduction of stochastic hybrid systems, which are useful for understanding the uncertainty analysis of linear hybrid systems.

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 [26]. 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 [76]. 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. During the design process, the state variables are linked in a state effects diagram. The relationships between state variables, commands, and measurements denoted in this diagram are associated with corresponding state effects models.

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. [26]. 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.

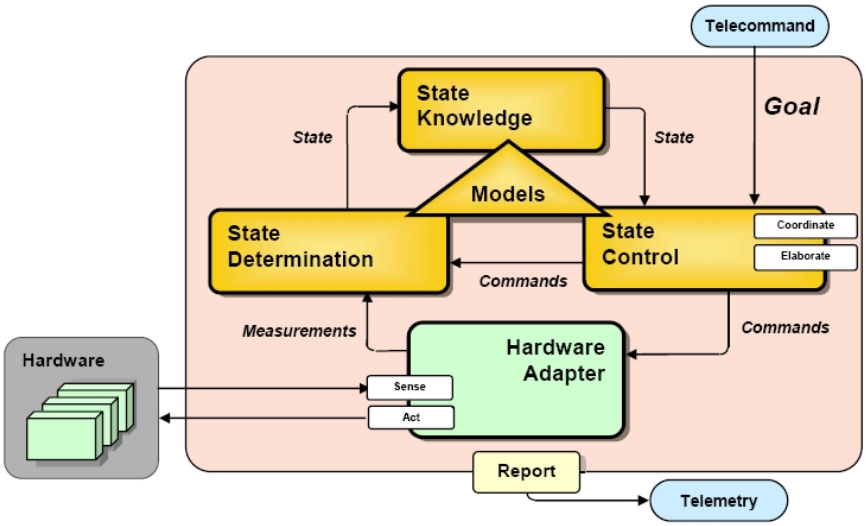

Figure 2.1: A depiction of the state and model-based architecture of the Mission Data System, from Dvorak [78]

Therefore, the software architecture, MDS, is closely related to State Analysis [77]. A depiction of the MDS control architecture is shown in Figure 2.1.

Goal networks in MDS replace command sequences in traditional control architectures as the 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. Controlled goals cause some control action to be taken, either because of its constraint or because of a constraint in an elaborated child goal. Passive goals constrain the state of a state variable to be some specific value or set of values without an associated control action. Passive goals are used to choose between the tactics of a controlled goal; for example, the health values of redundant heaters are constrained such that the tactics of a temperature maintenance goal are achieved using the minimum power rate.

There are several types of controlled goal constraints [26]. *Macro goals* are goals that do not constrain any state variable, but elaborate controlled goals with constraints. A *maintenance goal*, mentioned above, uses control action to maintain the constrained value of a state variable. A maintenance goal on temperature, i.e., keep the temperature of an instrument between $15 - 20°$C, could elaborate control constraints on heater switches to achieve the temperature constraint. *Reset con-*

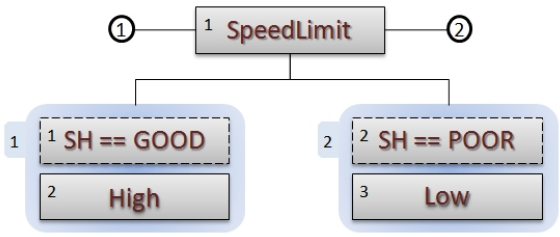

Figure 2.2: Goal tree example; circles are time points which bound the root goal. Rectangles are goal constraints (controlled goals have solid borders and passive goals have dashed borders; these types of goals are numbered independently in the upper left corner of the goal), and elaboration is signified by the tree structure. The parent goal's tactics are numbered in the shaded tabs.

straints command changes in discrete state variables (such as switches and states that have control modes). A *rate constraint* puts an upper or lower bound on the rate of some state variable. Finally, a controlled goal could have a *transition constraint*. This constraint drives a state variable from its current state to an end point; an example is a constraint to pan a camera to a given angle from some arbitrary starting value. The goal is achieved, or completed, when the state variable reaches the constrained value. For this reason, goals with this type of constraint are called *completion goals*.

Elaboration allows MDS more flexibility 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 for a goal can include several pre-defined tactics. These tactics, or collections of concurrently elaborated and executed goals, are simply different ways to accomplish the intent of the goal, and passive goals constrain the conditions in which a tactic may be executed. A simple example of a speed limit goal and its elaborations (called a goal tree) is shown in Figure 2.2. The passive goals constrain the `SystemHealth` state variable (`SH`); the two tactics have controlled goals constraining different maximum speeds that could be allowed based on the `SystemHealth`'s value. Elaboration allows for many types and combinations of faults to be accommodated automatically by the control system [27].

The goal networks used for this work are closely related to the ones implemented by MDS. The biggest difference is how the goal networks are executed. In MDS, a goal network is initially scheduled and at that time, all parent goals elaborate only one tactic. The tactics chosen are based on specified elaboration order and/or the projections of the values of the constrained state variables. Projections of state variables are functions that predict the future value of a state variable based on its current (or initial) state, its state model and the goals constraining that state variable. Then,

as the goal network execution reaches an elaborated tactic, execution of that tactic begins with a check of the satisfaction of the constraint of the tactic based on the current values of the constrained state variables. If it is not achievable, or if the execution of that tactic is initially achievable but becomes unachievable during the execution, then the tactic will fail and re-elaboration will be triggered. During the re-elaboration, the entire goal network is rescheduled and re-elaborated with new projections while the failed tactic continues to execute. Once the rescheduling is complete, which could be several time steps later depending on the size of the goal network, the execution of the new tactic in the rescheduled goal network begins if it is still achievable.

Because these time delays in switching the execution tactics upon goal failure are problematic when one wishes to verify the behavior of a goal network, the execution of the goal networks in this work is assumed to be different. First, the goal network is scheduled initially, but elaboration does not occur in the same way. Instead of choosing one tactic per parent goal, essentially all possible goal network executions are elaborated. Then, tactics are chosen instantaneously based on the state variable values as the goal network execution reaches the parent goal. Upon goal failure, a new tactic of only the failed parent goal occurs, and the execution switches to the new tactic in the next time step. Because the goal network does not reschedule upon goal failure, some flexibility is lost, but in the structure that is imposed, a very useful property is gained. The time points are guaranteed to fire in the order that they are originally scheduled, which means the goal network is well-ordered. This property will be important to the verification process.

In a fault-tolerant control program with conditional branching, one execution branch or tactic is chosen over another based on the states of the system because it is the safest and best control tactic available. Therefore, continuing to execute a failed or incorrect tactic while the goal network re-elaborates is contrary to the intent of safety. For this reason, re-elaboration is assumed to happen in the same time step as goal failure. This is not an impossible assumption to execute; localized re-elaboration or total initial elaboration at scheduling could be implemented as a design choice.

Other new design choices are implemented for these goal networks. First, only controlled goals can elaborate other goals. If more than one tactic exists in a parent goal's elaborations, at least one goal in each tactic must be a passive goal. For convenience, controlled goals should not fail independently; instead, an accompanying passive goal would cause the tactic to fail. For example, assume there was a controlled goal constraining a heater switch to turn off, but that the heater was failed on. Instead of the controlled goal failing, an accompanying passive goal constraining the `HeaterSwitchHealth` state variable to be `GOOD` would fail, since its estimated state should be

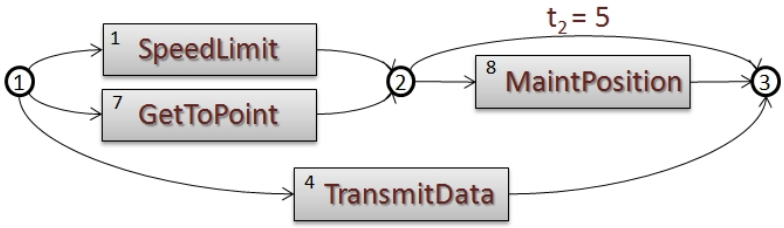

Figure 2.3: Goal network for simple example

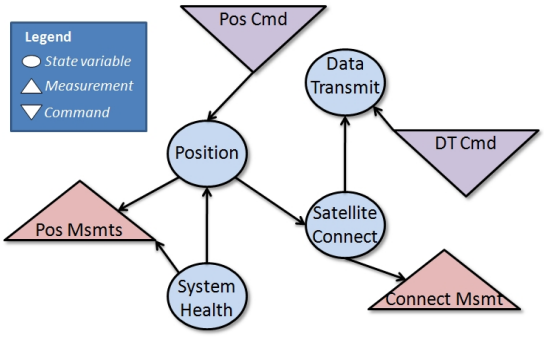

Figure 2.4: State effects diagram for simple example. Arrows indicate that the originating body has a modeled effect on the accepting body.

FAILON.

An example of a simple goal network that follows these design choices is shown in Figure 2.3. The goal network for this example has three time points, four root, or parent, goals, and it constrains four state variables, whose state effects diagram is shown in Figure 2.4. Two state variables are controlled, `Position` and `DataTransmission`; the former is a continuous state variable while the latter is discrete. The other two state variables are passive and have non-deterministic discrete models that are shown in Figure 2.5. Two of the root goals in the goal network do not have elaborations; however the goal trees for the SpeedLimit goal and the TransmitData goal are shown in Figure 2.6. The tactics shown in the goal trees contain passive goals on two state variables, `SystemHealth` and `SatelliteConnection`, that drive the choice of the tactics. The rest of the goals are controlled goals on the two controllable state variables and they cover several constraint types, which are shown in Table 2.1.

The overall task that this goal network is attempting to achieve is to drive a robot to a point while maintaining some safe velocity and transmitting data to a satellite. The execution of this goal network occurs as follows. The first time point fires, which starts the execution of the SpeedLimit, GetToPoint, and TransmitData goals. In addition, the SpeedLimit and TransmitData goals each elaborate one tactic based on the estimated states of the `SystemHealth` and `SatelliteConnection`

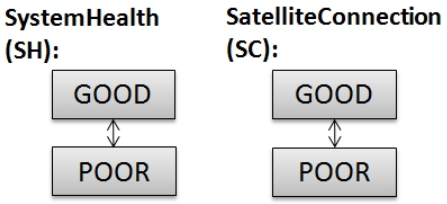

Figure 2.5: Non-deterministic models for the example passive state variables

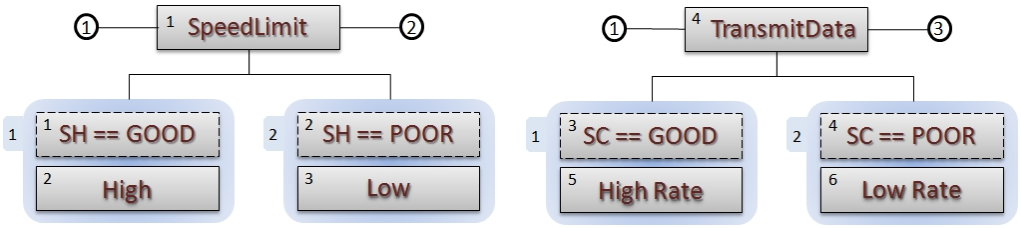

Figure 2.6: Goal trees for SpeedLimit and TransmitData goals

Table 2.1: Controlled Goal Constraint Types

Goal Index	Goal Name	State Variable	Constraint Type
1	SpeedLimit	None	Macro
2	High	`Position`	Rate
3	Low	`Position`	Rate
4	TransmitData	`DataTransmission`	Reset
5	High Rate	`DataTransmission`	Rate
6	Low Rate	`DataTransmission`	Rate
7	GetToPoint	`Position`	Transition
8	MaintPosition	`Position`	Maintenance

state variables. If the states of these two state variables change at any time during the execution, the tactic constraining that state fails and the other tactic is elaborated. For example, if the state of both passive state variables is `GOOD` initially, the first tactics of both goal trees are elaborated. However, if the `SystemHealth` state variable becomes `POOR` in the next time step, the first tactic of the SpeedLimit goal will simultaneously fail and the second tactic will elaborate in its place. The firing of the second time point will occur when the transition goal, GetToPoint, completes. At that time, the TransmitData and MaintPosition goals will be active along with one tactic of the TransmitData goal. The third time point fires, ending the goal network execution, after the constrained amount of execution time ($t_2 = 5$) has passed.

2.2 Linear Hybrid Automata

Hybrid systems exhibit discrete modes of execution that have different continuous behavior or control. Switching between discrete modes can be random, timed, or guarded by some state-based condition. Hybrid systems are very prevalent and have a range of uses; therefore, there are several different ways to model them [79]. Two of the most common ways depend on the more interesting control interface for the system; hybrid automata are focused on the discrete mode switching whereas other hybrid systems, sometimes called ODE models, are focused on the continuous dynamics in the discrete modes. In this work, a linear hybrid automata model is used, as the continuous dynamics are restricted to be piecewise constant first derivatives.

A linear hybrid automaton H consists of the following components [39]:

1. A finite, ordered list of controlled state variables and clock timers, $X = \{x_1, x_2, ..., x_n\}$.

2. A finite, ordered list of passive state variables, $\mathcal{D} = \{d_1, ..., d_m\}$. The set of discrete states (or discrete sets of states) of the state variable $d_i \in \mathcal{D}$ is $\Lambda_i = \{\lambda_1, ..., \lambda_{n_i}\}$.

3. 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.

4. The set of invariants for each location, $inv(v)$, which are the conditions on the state variables, $X \cup \mathcal{D}$, that must be true in that location.

5. The set of flow conditions, $\psi_i : X \rightarrow X$, for location $v_i \in V$, which are the equations that dictate how state propagates in each location.

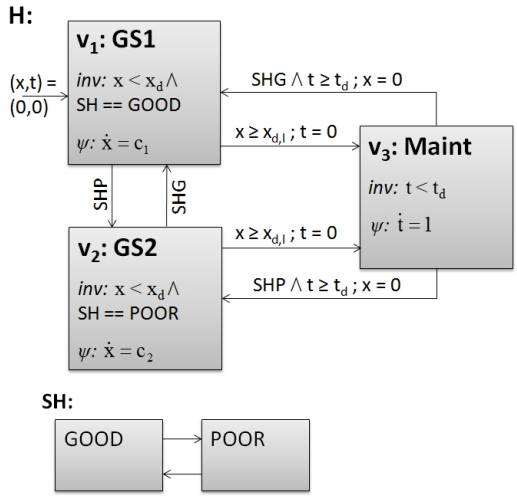

Figure 2.7: Hybrid automaton and state model example; boxes are locations or state values and arrows are edges labeled with transition conditions and resets where appropriate.

6. The set of transition conditions associated with each edge, Σ.

7. The set of transition actions or reset equations associated with each edge, A.

8. The initial conditions of the state variables, *init*.

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 an automaton, H and the model of the `SystemHealth` state variable, shown in Figure 2.7. The sets of state variables associated with this automaton are $X = \{x, t\}$, where x is the position and t is a timer, and $\mathcal{D} = \{\texttt{SH}\}$, where `SH` is the `SystemHealth` state variable. The locations and transition arrows (minus the conditions) compose sets V and E. The initial conditions are $(x, t, \texttt{SH}) = (0, 0, \texttt{GOOD})$. The transition conditions from v_1 and v_2 to v_3 are $x \geq x_{d,l}$ (these same transition edges have reset conditions on the timer, $t = 0$); the related invariants of v_1 and v_2 are $x < x_d$, where $0 < x_{d,l} < x_d$; $x_d, x_{d,l} \in \mathbb{R}$. This means that as soon as x reaches $x_{d,l}$, which may be the distance at which the UAV is in range of x_d, the transitions can occur, but the transition will definitely occur when $x = x_d$. Likewise, the other invariant and transition conditions dictate when the discrete transitions will occur. Finally, the differential equations with piecewise constant rates that control the flow of the variables are listed in each location.

There are several symbolic model checkers available that are capable of verifying linear hybrid

automata. These components 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.

2.3 Stochastic Hybrid Systems

Stochastic hybrid models are hybrid systems with some uncertainty in the continuous dynamics, the discrete mode switching, or both. These systems can be classified as one of three types of models [80]. Piecewise Deterministic Markov Processes (PDMP) have random discrete transitions; the hybrid state is reset based on some probability distribution upon transitions, however between transitions, the continuous state evolution is based on ordinary differential equations. In Switching Diffusion Processes (SDP), uncertainty is included in both the continuous and discrete state evolution. A Markov chain directs the discrete switching while stochastic differential equations describe the continuous state. Finally, Stochastic Hybrid Systems (SHS) have stochastic differential equations describing the continuous state evolution while the discrete mode switching is deterministic. A discrete-time SDP will be used in this work because discrete-time execution is assumed [81].

Definition 2.3.1. A *discrete-time switching diffusion process*, $\mathcal{H}$, consists of the following components:

1. A finite, ordered list of controlled state variables and timers, $X = \{x_1, x_2, ..., x_n\}$.

2. A finite, ordered list of passive state variables, $\mathcal{D} = \{d_1, ..., d_m\}$. The set of discrete states (or discrete sets of states) of the state variable $d_i \in \mathcal{D}$ is $\Lambda_i = \{\lambda_1, ..., \lambda_{n_i}\}$.

3. The set of discrete locations, $\mathcal{V} = \{v_1, v_2, ..., v_m\}$, where $m \in \mathbb{N}$.

4. The stochastic flow of a location, $\phi : X \times \mathcal{V} \rightarrow \mathbb{R}^{d(\mathcal{V})}$, where $d(v)$ is the dimension of the continuous state space in location v.

5. The initial conditions of the system, *init*, based on some probability model.

6. The set of edges between locations, $\mathcal{E}$.

7. The transition probability associated with a given edge, $\mu : \mathcal{V} \times (X \cup \mathcal{D}) \times \mathcal{E} \rightarrow [0, 1]$, that depends on the location and the continuous and discrete state.

8. The set of transition actions or reset equations associated with each edge, A.

This definition is close to the definition of the linear hybrid automata with stochasticity added to the flow equations, and transition probabilities replacing transition conditions and invariants.