Chapter 4

Efficient Verification for Systems with State-Based Transitions

4.1 Introduction

Goal network control programs can be converted to hybrid systems using the bisimulation procedure introduced in Chapter 3 and then verified using existing model checking software. However, this approach is restricted by the symbolic model checker used in the verification; oftentimes, to use these software programs to verify real systems, abstraction, model reduction, and overapproximation of the system is necessary. The limiting factor in the use of these symbolic model checkers is often the number of state variables in the system, which for real systems can be very large. The conversion procedure also has difficulty handling many passive state variables because of the way failure transitions are created. Some limiting assumptions can improve the performance of the conversion software, but at a cost.

The main contribution of this chapter is the design for verification software tool and the resulting verification algorithm. The design tool used to create goal networks with state-based transitions (defined in Section 4.2), the SBT Checker, is introduced in Section 4.3. The verification software, InVeriant, is described in Section 4.4. The application of the InVeriant model checker to a class of linear hybrid systems is discussed in Section 4.5. The capabilities, strengths and weaknesses of this verification approach are discussed in Section 4.6, followed by a summary of the contributions in Section 4.7.

4.2 State-Based Transitions

A class of goal networks, ones with state-based transitions, have special properties in the bisimilar automata.

Definition 4.2.1. Let $\mathcal{D}_k = \{d_1, d_2, ..., d_{n_k}\}$ be the set of state variables constrained by passive goals in U_k. Then, let Γ_k be the passive state space, $\Gamma_k = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_{n_k}$. If for each state $\gamma_i \in \Gamma_k$, there exists some executable set, $\theta_j \in \Theta_k$ such that the passive state satisfies the passive constraints, $\gamma_i \models \text{pcons}(\theta_j)$ for each group, $k = 1, ..., K$, and the elaboration conditions for each parent goal are based only on the states of the system, then the goal network has *state-based transitions*.

An example goal tree that does not have state-based transitions is shown in Figure 4.1. The speed limit root goal has three tactics constraining two passive state variables, `SystemHealth` and `PositionUncertainty`, whose models are included in Figure 4.1. The passive state space is also shown; it is obvious that two states (`SH` $==$ `GOOD` $\wedge$ `PU` $==$ `HIGH` and `SH` $==$ `POOR` $\wedge$ `PU` $==$ `LOW`) do not satisfy the passive constraints in any tactic. Therefore, this goal tree does not have state-based transitions. While this construction is valid, there is no way to predict what will happen to the execution when these states occur; the current tactic would fail but since there is no tactic associated with these states, any of the tactics may be chosen as the execution of the goal network breaks down. However, with two changes, the goal tree in Figure 4.2 does have state-based transitions, which is obvious from the passive state representation.

The assumption that the goal elaboration is based only on the states of the system follows the design philosophy of State Analysis and MDS. This simply says that there is no predefined order to the tactics of any goal; instead, the passive state constraints control goal elaboration. Since goal networks are bisimilar with hybrid systems, the definition of state-based transitions also applies to them. In a converted hybrid system with state-based transitions, each passive state $\gamma_i \in \Gamma_k$ satisfies the invariant, $\gamma_i \models \text{inv}(v_j)$, of some location $v_j \in V_k$ for all $k = 1, ..., K$. Because of the elaboration condition on goal networks with state-based transitions and because of the structure of goal networks, the invariants of the locations in a hybrid system with state-based transitions completely describe all transitions into and out of the locations. It is this property that is useful when trying to verify the hybrid system. Instead of finding all the transitions of the hybrid system, which can be prohibitive, the invariants could be used in the verification instead. This property also gives interesting results about the reachability of locations, which will be described in Section 4.4.

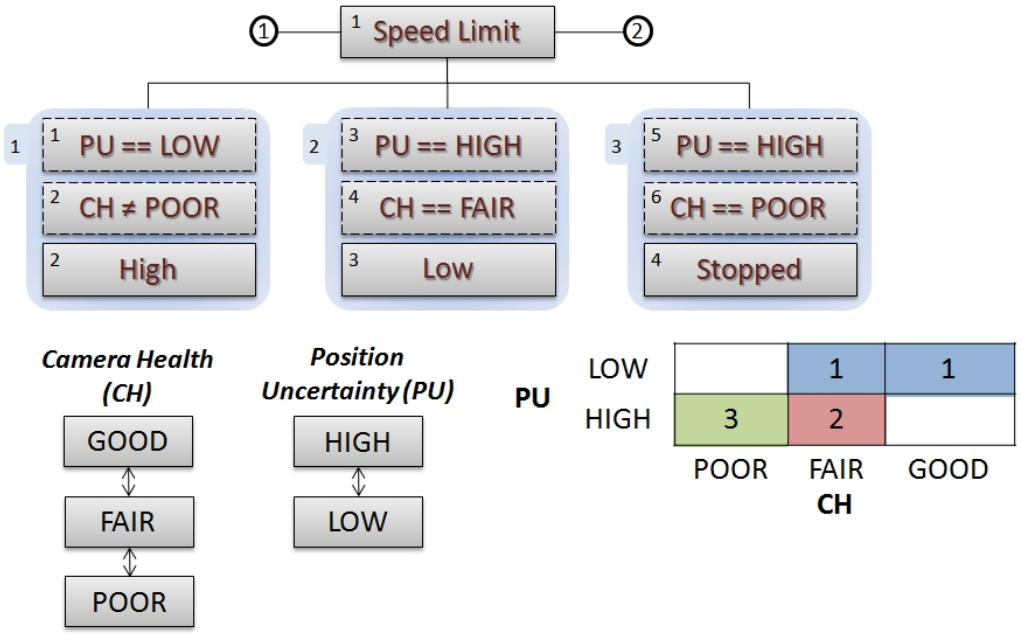

Figure 4.1: Goal tree that does not have state-based transitions with associated passive state models and passive state space

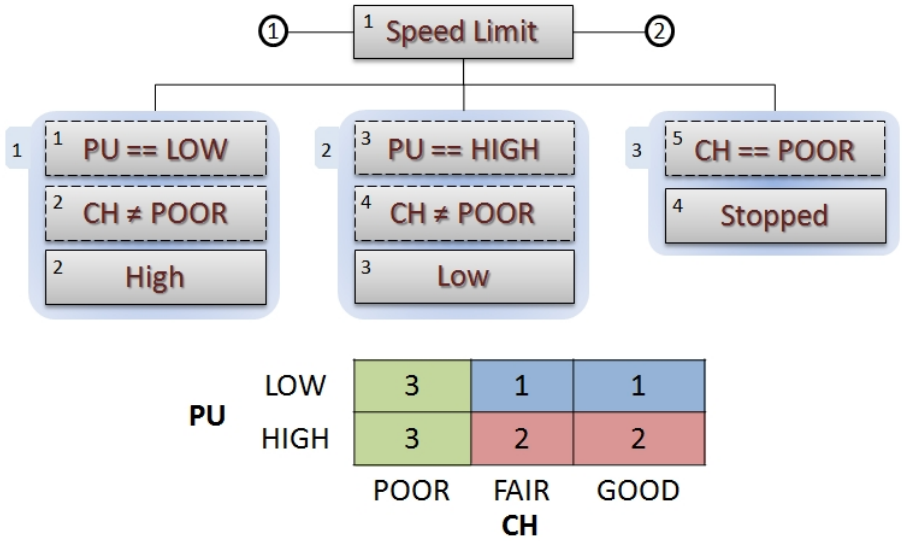

Figure 4.2: Goal tree with state-based transitions and associated passive state space

Therefore, goal networks that are designed to have state-based transitions can be verified using a very simple search algorithm that can handle complex systems.

While designing goal networks to have state-based transitions does impose some structure on the goal network, one could argue that the requirement is a good design practice. In the case where one or more passive states are not associated with an executable set of goals, it is unclear what would happen with the execution in that state. When state-based transitions are present, the goal network is maximally fault tolerant given the state model since every possible passive state is accounted for. In the case that there are passive states that satisfy the passive constraint of more than one executable set (or the invariant of more than one location), a non-deterministic execution scheme with weak fairness may be used and verified using the same method presented here.

4.3 SBT Checker

The conversion to hybrid systems and verification using symbolic model checkers presented in Chapter 3 is one approach to the verification of goal networks. However, the conversion procedure creates an automaton that captures all possible executions of the goal network; this can cause an explosion in the number of discrete modes (locations) relative to the numbers of goals and tactics present in the goal networks. While symbolic model checkers such as PHAVer handle large numbers of locations more easily than large numbers of state variables, there is still a limit. Abstractions and simplifications of the hybrid system can be used to aid in its verification, but another approach is to design the system for verification initially. To ensure that a goal network can be verified by the procedure described in this chapter, the goal network must have state-based transitions. For complex goal networks, this is not always easy to do by hand; therefore, the software program, SBT Checker, has been created to aid in the design process.

One way to verify that a goal network has state-based transitions is to convert it to a hybrid system and then compare each passive state to the locations' invariants in each group. However, for large systems with many locations and many states, this check can be time-consuming and ineffective for the iterative design process. Based on the following theorem, however, it is possible to check that the individual goal trees of each root goal in a goal network have state-based transitions, and that implies that the goal network has state-based transitions.

Let $\mathcal{D}_k$ be the set of state variables constrained by the passive goals in group U_k. Let $R_k = \{g_{r_1}^{0,0}, g_{r_2}^{0,0}, ..., g_{r_N}^{0,0}\}$ be the set of root goals that are in or have children in group $\mathcal{G}_k$. Let $\mathcal{S}_{r,k}$ be the

set of descendants of $g_r^{0,0}$, including passive goals and the root goal itself.

$$\mathcal{G}_k \subseteq \bigcup_{g_r^{0,0} \in R_k} \mathcal{S}_{r,k}, \tag{4.1}$$

because there may be extra root goals in $\mathcal{S}_{r,k}$. Let $\mathcal{D}_{r,k} = \{\text{svc}(u_m^{i_m,j_m}) | u_m^{i_m,j_m} \in \mathcal{S}_{r,k}\}$ be a set of all state variables constrained passively in $\mathcal{S}_{r,k}$. For some $\mathcal{D}_{r,k} = \{d_{n_1}, d_{n_2}, ..., d_{n_D}\}$, let $\Gamma_{r,k} = \Lambda_{n_1} \times \Lambda_{n_2} \times ... \times \Lambda_{n_D}$ be the passive state space of $\mathcal{S}_{r,k}$. Let $\gamma_i^r \in \Gamma_{r,k}$ be a passive state of the state variables in $\mathcal{D}_{r,k}$.

Let $\mathcal{L}_{r,k}$ be the set of executable branches of goals in $\mathcal{S}_{r,k}$. An executable branch of goals $L_j \in \mathcal{L}_{r,k}$ has the following properties:

1. All goals $g_n^{i_n,j_n} \in L_j$ are also in $\mathcal{S}_{r,k} \cap \mathcal{G}_k$.
2. If $g_n^{i_n,j_n} \in L_j$, its parent is also in L_j, $g_{i_n}^{i_{i_n},j_{i_n}} \in L_j$.
3. If $g_n^{i_n,j_n} \in L_j$, all its siblings are also in L_j.
4. If $g_n^{i_n,j_n} \in L_j$ and $g_n^{i_n,j_n}$ has at least one child goal in $\mathcal{S}_{r,k}$, then at least one child goal of $g_n^{i_n,j_n}$, $g_m^{i_m,j_m} \in \mathcal{S}_{r,k}$, $i_m = n$, is in L_j, $g_m^{i_m,j_m} \in L_j$.
5. All goals in L_j are compatible.
6. All goals in L_j are consistent.

Lemma 4.3.1. *For each pair of executable branches from different root goals, $L_i \in \mathcal{L}_{r_i,k}$, $L_j \in \mathcal{L}_{r_j,k}$, $r_i \neq r_j$, L_i is compatible with L_j.*

Proof. The general idea is that since the executable branches are drawn from different root goals, there are no shared parent goals across the executable branches. Assume that $L_i \in \mathcal{L}_{r_i,k}$ is incompatible with $L_j \in \mathcal{L}_{r_j,k}$, $r_i \neq r_j$. This means that there exists some $g_n^{i_n,j_n} \in L_i$ and $g_m^{i_m,j_m} \in L_j$ that have the same parent, $i_n = i_m$. Since all parent goals of each goal in an executable branch are also in that set by definition, this means that L_i and L_j have the same root goal, which negates the original assumption that $r_i \neq r_j$. □

Lemma 4.3.1 shows that executable branches from different root goals are compatible, and so they can be combined. The proposition introduced next says that if executable branch combinations are consistent, they are equivalent to executable sets of goals.

Proposition 4.3.2. *Let $\Upsilon_k = \mathcal{L}_{r_1,k} \times \mathcal{L}_{r_2,k} \times ... \times \mathcal{L}_{r_N,k}$ be the set of all combinations of executable branches from each root goal's set of goals. Let $\upsilon \in \Upsilon_k$; if all goals in υ are consistent, $\upsilon \equiv \theta_j$ for some $\theta_j \in \Theta_k$. Moreover, let $\Upsilon'_k = \{\upsilon | \upsilon$ is consistent$\}$. Then, $\Upsilon'_k \equiv \Theta_k$.*

Proof. Because Θ_k contains all possible executable sets in $\mathcal{G}_k$, if υ satisfies the definition of an executable set, there exists some $\theta_j \in \Theta_k$ such that $\upsilon \equiv \theta_j$. By the definition of the executable branches $L \in \mathcal{L}_{r,k}$, properties 1 and 3-5 of the executable set specification in Definition 3.4.3 are satisfied. All goals in each L are also in $\mathcal{G}_k$ by definition, so all the goals in the composition, $g_n^{i_n,j_n} \in \upsilon$ are also in $\mathcal{G}_k$ (property 1). Likewise, all parent goals, sibling goals, and at least one child goal are represented in each branch, so the composition of these branches originating from different root goals will have the same properties (properties 3–5). Property 6 of executable sets is satisfied by the composition of Υ_k; since all root goals with children in group $\mathcal{G}_k$ are represented by a set $\mathcal{L}_{r,k}$, at least one goal from each root goal with children in the group is represented in each υ. Property 2 is satisfied in a similar manner; if the root goal $g_r^{0,0} \in \mathcal{G}_k$, then for each $L \in \mathcal{L}_{r,k}$, $g_r^{0,0} \in L$ because of the parent goal requirement in the definition of L. Property 7 is satisfied by Lemma 4.3.1 and Property 8 is satisfied by the assumption above. Therefore, $\upsilon \equiv \theta_j$ for some $\theta_j \in \Theta_k$.

The above result can be applied to every $\upsilon \in \Upsilon'_k$ since each υ in that set is consistent. So, to prove that $\Upsilon'_k \equiv \Theta_k$, assume that there exists some $\theta_j \in \Theta_k$ such that there is not a corresponding $\upsilon \in \Upsilon'_k$. By definition, each goal in θ_j is either a root goal in $\mathcal{G}_k$ or descended from a root goal that has child goals in $\mathcal{G}_k$. Also by definition, each of those goals are present in a branch in the corresponding root goal's branch set, $\mathcal{L}_{r,k}$. Since there exists a set $\upsilon \in \Upsilon'_k$ that corresponds with every consistent combination of branches from each root goal $g_r^{0,0} \in R_k$, the executable set must either be missing a branch from at least one root goal's set or have more than one branch from at least one root goal's set. However, missing a branch from a root goal's set would violate either property 2 or 6 from Definition 3.4.3 of executable sets since either a root goal or descendants from a root goal would be missing from θ_j; so, θ_j must have two or more branches from at least one root goal's set. Let both $L_n \in \mathcal{L}_{r,k}$ and $L_m \in \mathcal{L}_{r,k}$ be subsets of θ_j; this implies that L_n and L_m are compatible. There must be some goals $g_n^{i_n,j_n} \in L_n$ and $g_m^{i_m,j_m} \in L_m$ such that $g_n^{i_n,j_n} \notin L_m$ and $g_m^{i_m,j_m} \notin L_n$ because one branch cannot be a subset of another by the definition of executable branches. The goals, $g_n^{i_n,j_n}$ and $g_m^{i_m,j_m}$, cannot be siblings and one cannot be the ancestor of the other by definition. However, they do have the same ancestor because they are in the same root

goal set. So, these goals must be incompatible by definition. Therefore, each executable set θ_j must include one and only one branch from each root goal and therefore there is a set $\upsilon \equiv \theta_j$ for each executable set $\theta_j \in \Theta_k$. Therefore, $\Theta_k \equiv \Upsilon'_k$. □

Theorem 4.3.3. *If for all controlled goals in branches from different root goals, $g_n^{i_n,j_n} \in \mathcal{L}_{r_n,k}$, $g_m^{i_m,j_m} \in \mathcal{L}_{r_m,k}$, $r_m \neq r_n$, the goals are consistent, $c(g_n^{i_n,j_n}, g_m^{i_m,j_m})$, and for all $g_r^{0,0} \in R_k$, the set of executable branches, $\mathcal{L}_{r,k}$, has state-based transitions over $\mathcal{D}_{r,k}$, then Θ_k has state-based transitions over $\mathcal{D}_k$.*

Proof. By definition, $\Upsilon_k = \mathcal{L}_{r_1,k} \times ... \times \mathcal{L}_{r_N,k}$ for all $g_{r_i}^{0,0} \in R_k$, $i = 1, ..., N$. By Proposition 4.3.2, the consistent subset, $\Upsilon'_k \subseteq \Upsilon_k$ is equivalent to Θ_k, so to prove this theorem, it is sufficient to show that Υ'_k has state-based transitions over $\mathcal{D}_k$. Let $I_k = \Upsilon_k \setminus \Upsilon'_k$ be the set of executable branch combinations with inconsistent goals. Since all active goals are consistent by assumption, each $\upsilon \in I_k$ has inconsistent passive goals only.

Let each $\mathcal{L}_{r,k}$ have state-based transitions over the corresponding passive state variable set $\mathcal{D}_{r,k}$, but assume that Υ'_k does not have state-based transitions over $\mathcal{D}_k$. That means that there exists some state $\gamma \in \Gamma_k$, where $\Gamma_k = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_D$, and D is the number of passive state variables in $\mathcal{D}_k$, such that there are no $\upsilon \in \Upsilon'_k$ such that $\gamma \models \text{pcons}(\upsilon)$. However, by the definition of state-based transitions, for each $\mathcal{L}_{r,k}$, there exists some $L_{r,\gamma} \in \mathcal{L}_{r,k}$ such that $\gamma \models \text{pcons}(L_{r,\gamma})$ because $\mathcal{L}_{r,k}$ has state-based transitions over $\mathcal{D}_{r,k} \subseteq \mathcal{D}_k$. By definition, there exists some $\upsilon \in \Upsilon_k$ such that

$$\upsilon = \bigcup_{g_r^{0,0} \in R_k} L_{r,\gamma}.$$

To satisfy the assumption that Υ'_k does not have state-based transitions, $\upsilon \in I_k$, which means that it has inconsistent passive goals. Without loss of generality, let the inconsistent goals be $u_n^{i_n,j_n}, u_m^{i_m,j_m} \in \upsilon$. By the definition of consistency, $\text{svc}(u_n^{i_n,j_n}) = \text{svc}(u_m^{i_m,j_m}) = d_i$ for some $d_i \in \mathcal{D}_k$ and for any $\lambda_j^i \in \Lambda_i$ such that $\lambda_j^i \models \text{cons}(u_n^{i_n,j_n})$, $\lambda_j^i \not\models \text{cons}(u_m^{i_m,j_m})$. Likewise, by the definition of γ, if any $\gamma \models \text{cons}(u_n^{i_n,j_n})$, then $\gamma \not\models \text{cons}(u_m^{i_m,j_m})$. Since $u_m^{i_m,j_m} \in \upsilon$, there must exist a $L_{r,\gamma} \subset \upsilon$ such that $u_m^{i_m,j_m} \in L_{r,\gamma}$. Then, $\gamma \not\models \text{pcons}(L_{r,\gamma})$, which means that $\mathcal{L}_{r,k}$ is not state-based and the original assumption is negated. □

The SBT Checker leverages this modularity of goal networks to check that each root goal's tactics have state-based transitions. The algorithm involves comparing the passive constraints in

Figure 4.3: Goal network for the state-based transitions verification example

Table 4.1: State Variable Data

State Variable	Abbreviation	Type
Position	X	Controlled
Camera Mode	CM	Controlled
Stabilizer Switch	SS	Controlled
Camera Health	CH	Passive
Position Uncertainty	PU	Passive
Vibrations	VB	Passive

each executable branch, $L_i \in \mathcal{L}_{r,k}$ for $g_r^{0,0} \in R_k$, to each passive state in the state space $\Gamma_{r,k}$ for the passive state variables in $\mathcal{D}_{r,k}$. Then, each passive state $\gamma \in \Gamma_{r,k}$ is checked against the passive constraints in each executable branch and if there exists some $\gamma \in \Gamma_{r,k}$ such that there is no $L_i \in \mathcal{L}_{r,k}$ where $\gamma \models \text{pcons}(L_i)$, those passive states are listed for the designer. The output of the SBT Checker software for the goal tree in Figure 4.1 would be $(\texttt{SH} == \texttt{GOOD} \wedge \texttt{PU} == \texttt{HIGH}) \vee (\texttt{SH} == \texttt{POOR} \wedge \texttt{PU} == \texttt{LOW})$. The output for the goal tree in Figure 4.2 would be `False`. The controlled goal consistency constraint is checked upon the goal network's conversion to a hybrid automaton in the verification software.

Because the number of executable sets, or locations, grows exponentially with the number of parent root goals, the modular approach saves computation time. It is also more conducive to an iterative and distributed design process since it gives nearly immediate feedback on the design of a root goal's goal tree.

A simple goal network example is shown in Figure 4.3. The robot's task is to drive to a point maintaining a safe velocity while taking pictures; Table 4.1 lists the state variables constrained in the goal network. The goal tree for the SpeedLimit goal is shown in Figure 4.2, and the goal tree for the TakePictures goal is shown in Figure 4.4. Both goal trees were verified to have state-based transitions by the SBT Checker which means that the entire goal network has state-based transitions. The goal network will be verified versus an unsafe set in the next section.

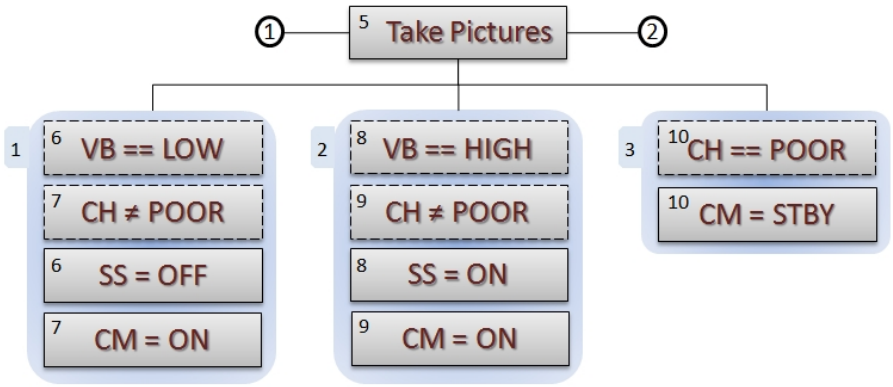

Figure 4.4: Goal tree of the TakePictures goal

4.4 InVeriant Verification Procedure

The idea behind the InVeriant software involves the special relationship between the invariant and the transition conditions for a hybrid system with state-based transitions. In a system with state-based transitions, if the state variables that are constrained in a group's locations have discrete states that are all reachable from each other, then each location in the group is reachable from any other location in the group. This is proved in Theorem 4.4.1 later. In this case, locations that satisfy the unsafe set are reachable if they exist. The InVeriant software creates the locations and invariants from the goal network and composes it with the unsafe set constraints to find unsafe locations.

While the assumption that the discrete states of the passively constrained state variables are reachable is usually a good one, there are times when it is not. In general, these passive state variables are health states or uncontrollable states of the environment that affect the way the system accomplishes a task. If there was a health or environment state value that was not reachable, it would not be modeled. However, continuous dependent state variables such as power or temperature may be constrained without knowing if a discrete set of states is reachable. These state variables, called continuous, rate-driven dependent state variables, have models whose discrete states do not match or correspond with the discrete sets of states that are passively constrained in the goal network. The discrete states in the model represent different rates of change of the state variable, which often depend on the controllable state variables, whereas the discrete sets of states constrained passively in the goal network depend on the continuous state space of the state variable. When unsafe locations constrain these state variables, their reachability must be confirmed by finding an appropriate path from the initial state to the unsafe state, a process that is aided by the state-based transition requirement.

The theorems that prove the methods used in the InVeriant software are presented next, followed by a formal treatment of the verification algorithm. The first theorem proves the reachability of locations introduced previously and the second theorem shows that within a set of locations that have the same discrete conditions on continuous, rate-driven dependent state variables, these locations are reachable.

Theorem 4.4.1. *Given a hybrid system with a set of locations $V_k = \{v_1, ..., v_n\}$ whose transitions are based on the discrete states of a set of passively-constrained state variables $\mathcal{D}_k = \{d_1, ..., d_m\}$, if all the discrete states associated with each passive state variable are reachable from each other, then all locations $v_l \in V_k$ are reachable from any other location, $v_j \in V_k$.*

Proof. Let V be a hybrid system with state-based transitions and let all discrete states $\Lambda_i = \{\lambda_1^i, ..., \lambda_{n_i}^i\}$ of each passive state variable $d_i \in \mathcal{D}$ be reachable from each other state, but assume location $v_l \in V_k$ is not reachable from $v_j \in V_k$. In order for v_l and v_j to be viable locations, they must have non-trivial invariants. That means that there must exist some $\gamma_j, \gamma_l \in \Gamma_k$ such that $\gamma_j \models \text{inv}(v_j)$ and $\gamma_l \models \text{inv}(v_l)$. Since the states of the passive state variables are all reachable, there is guaranteed to be a path from γ_j to γ_l. Let one possible path between the two system states be $p_{j,l} = \{\gamma_{i_1}, ..., \gamma_{i_n}\}$, where $p_{j,l}$ is the set of all intermediate states between γ_j and γ_l. Because V has state-based transitions, each $\gamma_i \in p_{j,l}$ has some $v_i \in V_k$ such that $\gamma_i \models \text{inv}(v_i)$. Since γ_j transitions to γ_{i_1} directly, there exists a transition condition $\tau_{j,i_1} = \text{inv}(v_{i_1})$. By induction through the path between states γ_j and γ_l, there is indeed a path between locations v_j and v_l, so the initial assumption is false. □

Theorem 4.4.2. *All locations $v_j \in V$ whose passive invariants have the same constraint, $\gamma^c \in \Gamma_k^c$, $\gamma^c \models inv(v_j)$, on the set of continuous dependent state variables, $\mathcal{D}_k^c$, are reachable from one another when all discrete passive state variables, $\mathcal{D}_k^d$ have reachable sets of states.*

Proof. Let $\mathcal{D}^c \subset \mathcal{D}$ be the set of all continuous, rate-driven dependent state variables and let

$$\Gamma^c = \prod_{d_i \in \mathcal{D}^c} \Lambda_i$$

be the state space of the passive constraints on those continuous, rate-driven dependent state variables. Likewise, let $\mathcal{D}^d \subset \mathcal{D}$ be the set of discrete passive state variables, $\mathcal{D}^d = \mathcal{D} \setminus \mathcal{D}^c$, and

let

$$\Gamma^d = \prod_{d_i \in \mathcal{D}^d} \Lambda_i$$

be the corresponding discrete passive state space. Let $V^{\gamma^c} \subset V$ be the set of locations satisfied by some state $\gamma^c \in \Gamma^c$, $V^{\gamma^c} = \{v_i | \gamma^c \models \text{inv}(v_i)\}$, where V has state-based transitions. Let $v_i, v_j \in V^{\gamma^c}$ and let v_j be unreachable from v_i. By the definition of state-based transitions, for all $\gamma \in \Gamma$ there exists some $v \in V$ such that $\gamma \models \text{inv}(v)$. It follows that for all $\gamma \in \Gamma^d_k \times \{\gamma^c\}$, there exists some $v \in V^{\gamma^c}$ such that $\gamma \models \text{inv}(v)$. Since all states $\gamma \in \Gamma^d \times \{\gamma^c\}$ are reachable from one another by design, it follows from Theorem 4.4.1 that $v_j \in V^{\gamma^c}$ is reachable from $v_i \in V^{\gamma^c}$, which contradicts the original assumption. □

The unsafe set is a collection of disjoint sets of constraints, $Z = \{\zeta_1, ..., \zeta_n\}$, where each disjoint set of constraints, $\zeta_i = \{z^i_1, ..., z^i_{n_i}\}$, has separate constraints on individual state variables, and each separate constraint $z \in (X^d \cup \dot{X}^c \cup \mathcal{D} \cup \dot{\mathcal{D}}^c) \times Q \times (\mathbb{R} \cup \Lambda)$ constrains a discrete controllable state variable (X^d), the rate of a continuous controllable state variable ($\dot{X}^c$), a passive state variable, or the rate of a continuous, rate-driven dependent state variable. The sets ζ of unsafe constraints are analogous to locations of the converted hybrid system, though the different sets in Z are not necessarily incompatible with one another; they are simply separate unsafe conditions against which the designer wishes to verify the system.

The verification algorithm goes through the following steps to verify a goal network versus the unsafe set, Z. A representation of this algorithm is shown in Figure 4.5.

1. Find all locations $v \in V$ from the goal network using the bisimulation procedure.

   (a) Find all potential executable sets of a goal network without checking for goal consistency. Find each location's passive invariant by listing all the passive constraints in that location; remove locations with inconsistent passive constraints.

   (b) Merge controlled constraints in each location and record any inconsistent controlled constraints. If there are any, stop and report which constraints are inconsistent. If not, continue.

2. For each $d_i \in \mathcal{D}$, the set of discrete states constrained passively in the goal network is Λ_i. Let $\mathcal{M}_i$ be the model of d_i, where $\mu^i_j \in \mathcal{M}_i$ is a discrete location in the model. For $d_i \in \mathcal{D}^d$, $\Lambda_i \equiv \mathcal{M}_i$. However, for $d_i \in \mathcal{D}^c$, the discrete sets are not always equivalent. Set $V' =$

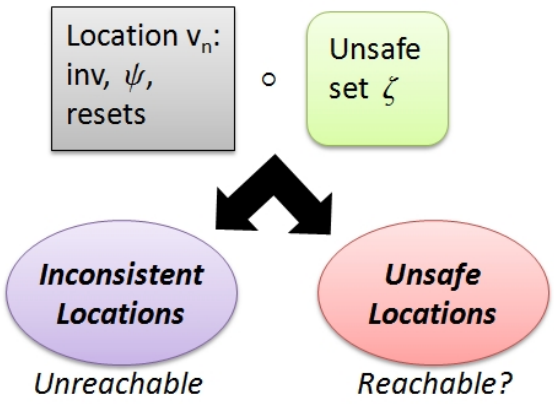

Figure 4.5: Representation of the InVeriant verification algorithm

V; for each $d_i \in \mathcal{D}^c$ such that $\Lambda_i \neq \mathcal{M}_i$, $V' = V' \circ \mathcal{M}_i = \{v_l \circ \mu_j | \forall v_l \in V, \forall \mu_j \in \mathcal{M}_i, v_l, \mu_j$ are consistent $\}$. A composed location $v' = v \circ \mu$ is defined as having a combined invariant, $\text{inv}(v') = \text{inv}(v) \wedge \text{inv}(\mu)$ and combined flow conditions, $\psi_{v'} = \psi_v \wedge \psi_\mu$.

3. For each $\zeta \in Z$:

   (a) Find the composition of the hybrid system and the unsafe set, $Y_\zeta = V' \circ \zeta = \{v_j \circ \zeta | \forall v_j \in V, v_j$ and ζ are consistent $\}$. Label all locations $y \in Y_\zeta$ as unsafe and output them.

   (b) Let the set of unsafe goals, $Y_{g,\zeta} = \{g_n^{i_n,j_n} \in y | \forall y \in Y_\zeta\}$, be the set of goals common to all unsafe locations. Output these goals.

   (c) Let the overloaded function cons() return the constrained value of a state variable when the function is given that state variable and a location (or set of constraints) as inputs. If there exists a $d_i \in \mathcal{D}^c$ such that $\text{cons}(d_i, y)$ exists, then a path must be found from $\text{init}(d_i)$ to $\text{cons}(d_i, y)$. There exists some $\lambda_j^i, \lambda_l^i \in \Lambda_i$ such that $\text{init}(d_i) \in \lambda_j^i$ and $\text{cons}(d_i, y) \cap \lambda_l^i \neq \emptyset$, where $\Lambda_i = \{\lambda_1^i, ..., \lambda_j^i, ..., \lambda_l^i, ..., \lambda_{n_i}^i\}$ is a forward or backward ordered set. Let $\Lambda_i^\zeta = \{\lambda_j^i, ..., \lambda_l^i\}$ be the set containing the two discrete sets of passively constrained values that satisfy the initial and unsafe constraints and all discrete sets of states in between. Then for each $\lambda_n^i \in \Lambda_i^\zeta$, a location $v \in V'$ must be found such that $(\lambda_n^i \models \text{inv}(v)) \wedge (\text{sign}(\text{cons}(\dot{d}_i, v)) = \text{sign}(\text{cons}(d_i, y) - \text{init}(d_i)))$ is true. If such a path can be found, the unsafe set is reachable.

This procedure is guaranteed by construction to find all locations in which unsafe conditions can occur. It can be applied to the example introduced in Section 4.3 as follows. Assume that the

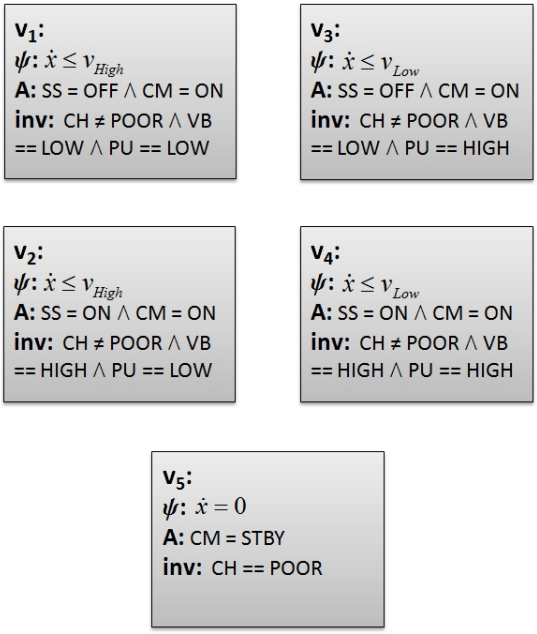

Figure 4.6: Converted locations, invariants, flow equations, and resets for the simple verification example

`StabilizerSwitch` state variable must always be in the `ON` position when the robot's velocity is high for safety reasons. Therefore, the unsafe set Z has one constraint, $\zeta = \{(\dot{\mathtt{X}}, >, v_{\text{low}}), (\mathtt{SS}, ==, \mathtt{ON})\}$. The first part of the InVeriant algorithm converted the goal network into a set of locations, invariants, flow equations, and resets, which are shown in Figure 4.6. Since none of the dependent state variables are continuous and rate-driven, the automaton did not need to be composed with any of the passive state model automata. So, the automaton in Figure 4.6 was composed with the unsafe condition ζ, and InVeriant found one location in which the unsafe set was reachable, v_1. The flow equations and reset equations of this location are consistent with the constraints in the unsafe condition. Since all of the passive states are reachable from each other passive state, this unsafe location is reachable. The combination of the first tactics in each goal tree are the culprits; these tactics must be redesigned so that they are inconsistent to be able to verify the goal network versus the given unsafe condition.

4.5 Verification of State and Completion-Based Linear Hybrid Systems

The software tools and verification method presented in the last two sections can also be applied to a broader class of linear hybrid automata. Essentially, the time point restrictions on the goal networks imposed for the conversion procedure induced a group structure on the linear hybrid automaton that resulted; this group structure, however, is not necessary for the verification method when starting from linear hybrid control systems. Lifting this requirement allows continuous controlled state variables to be reasoned about by InVeriant in the same way that continuous, rate-driven dependent state variables are. However, the state-based transition requirement must still hold for the overall (composed) hybrid system. Like in the case of the goal networks, it can be shown that the composition of hybrid automata that have state-based transitions also have state-based transitions if the dynamical flow constraints in the composed locations are consistent.

Definition 4.5.1. The dynamical flow constraints in locations of two hybrid automata are *consistent* if either they can be executed concurrently or there exists some rule that allows the constraints to be successfully merged into a composed dynamical flow constraint.

It is easy to see that this definition of consistency and the corresponding restriction are analogous to the goal network case of needing consistent controlled constraints. Often in the goal network case, the root goals would either be completion goals or would be macro goals combined with completion goals in the goal network. Since there is no good analog to the relationship that goal networks and goal trees have in the hybrid system case, a type of linear hybrid automaton called completion automata must be defined.

Definition 4.5.2. A *completion automaton* is a linear hybrid automaton in which the transition conditions and location invariants depend entirely on state variables that appear in the dynamical flow constraints or reset equations of the locations of that automaton.

Then, so-called state-based automata would have no transitions that depend on these types of state variables and instead would be automata that had state-based transitions as defined in the goal network case. Finally, the non-stochastic models of passive state variables would be called passive model automata. The composition of these types of automata is a hybrid system with certain properties as described by Theorem 4.5.3.

Theorem 4.5.3. *A hybrid automaton that is a composition of any number of completion and state-based automata has state-based transitions if all composed dynamical flow constraints are consistent.*

Proof. Let there be N state-based automata, H_n^p, where $n = 1, ..., N$. For each, the set of passive state variables constrained in the invariants of the locations is $\mathcal{D}_n \subseteq \mathcal{D}$. The composition of two automata $H' = H_i \circ H_j$ is defined as the joining of locations, $V' = V_i \circ V_j$, where each composed location $v' = v_i \circ v_j$, $v_i \in V_i$ and $v_j \in V_j$, has an invariant $\text{inv}(v') = \text{inv}(v_i) \wedge \text{inv}(v_j)$ and flow $\psi(v') = \psi(v_i) \wedge \psi(v_j)$. Assume that $H' = H_1^p \circ ... \circ H_N^p$, where H_n^p, $n = 1, ..., N$, have state-based transitions over $\mathcal{D}_n$, however, assume that H' does not have state-based transitions over

$$\mathcal{D} = \bigcup_{n=1}^{N} \mathcal{D}_n.$$

That means that there exists some passive state $\gamma \in \Gamma$, where Γ is the passive state space, such that for all $v \in V'$, $\gamma \not\models \text{inv}(v)$. However, since all H_n^p have state-based transitions over $\mathcal{D}_n \subseteq \mathcal{D}$, there exists some $v_n^\gamma \in V_n$ for all $n = 1, ..., N$ such that $\gamma \models \text{inv}(v_n^\gamma)$ by the definition of state-based transitions. The composition of all these locations, $v^\gamma = v_1^\gamma \circ ... \circ v_N^\gamma$, has an invariant

$$\text{inv}(v^\gamma) = \bigwedge_{n=1}^{N} \text{inv}(v_n^\gamma).$$

Since $\gamma \models \text{inv}(v_n^\gamma)$ for all $n = 1, ..., N$, by the linearity of the operator, $\gamma \models \text{inv}(v^\gamma)$. By the assumption, $\gamma \not\models \text{inv}(v)$ for all $v \in V'$, so v^γ must have an inconsistent invariant and so not be a location. However, if $\text{inv}(v^\gamma) = \text{False}$ and $\gamma \models \text{inv}(v^\gamma)$, then $\gamma = \text{False}$ and $\gamma \notin \Gamma$, which contradicts the original assumption. So, any number of state-based automata can be composed into an automaton that has state-based transitions.

By definition, completion automata, H_m^c, $m = 1, ..., M$, have invariants that depend only on controlled state variables, X. Since $X \cap \mathcal{D} = \emptyset$ and the transitions in the completion automata are also based on the locations' invariants, the invariants and transitions of these automata do not affect the passive state space. Let $H^c = H_1^c \circ ... \circ H_M^c$ be the composition of all completion automata. The composition, $v^\star = v^c \circ v'$, of any completion location $v^c \in V^c$ with any state-based location, $v' \in V'$, would have an invariant, $\text{inv}(v^\star) = \text{inv}(v^c) \wedge \text{inv}(v')$ that will never be inconsistent by definition since the invariants constrain different sets of state variables. Likewise, the dynamical flow constraints, $\psi(v^\star) = \psi(v^c) \wedge \psi(v')$, are consistent due to the assumption in

the theorem statement. So, no composed locations are inconsistent which means that since for all $\gamma \in \Gamma$, there exists some $v' \in V'$ such that $\gamma \models \text{inv}(v')$, the same will be true for all $v^\star \in V^\star$. Therefore, the composition of any number of completion and state-based automata will have state-based transitions, which proves the theorem. □

Because of this modularity, the SBT Checker can be used to check the state-based transitions of state-based automata. Likewise, the InVeriant verification software will check the consistent dynamical flow constraints upon composing the automata in preparation for model checking. Because the real requirement for InVeriant is a hybrid system whose invariant contains all the necessary information for the transitions of the system, if the composed completion automata have locations with unique invariants, this restriction is satisfied by that and the state-based transitions requirement. Continuous controlled state variables can then treated just like continuous, rate-driven dependent state variables (except no state variable model would need to be composed with the hybrid control system) in that a path must be found from the initial condition to the unsafe condition of those state variables to prove their reachability. Note that Theorems 4.4.1 and 4.4.2 still hold for the hybrid systems generated this way.

This result is important because it broadens the class of systems that can be verified using this method and it grants extra capabilities to the verification method by the addition of reasoning about continuous controllable state variables. This method is clearly more efficient than other symbolic model checkers such as PHAVer and HyTech that also deal with continuous states for systems that are designed with this additional structure. As discussed in the next section, at least some of the additional structure imposed is good design practice, so the SBT Checker and InVeriant verification toolbox may have many practical uses for the design and verification of real hybrid control systems.

4.6 Discussion

The verification procedure for systems that have state-based transitions has many positive attributes. First, the state-based transitions requirement can be checked modularly using a software tool, which allows for iterative and distributed design of control systems. The SBT Checker is a simple and intuitive tool that helps designers to design goal networks and hybrid automata that satisfy the state-based transitions requirement. This requirement forces certain structure in the goal network or hybrid automaton that aids in its verification; however, having state-based transitions is also a good design practice. When a system has state-based transitions, this means that every possible measured

passive state is accounted for and that there is a mode or tactic that can accommodate that state.

The InVeriant software is able to verify hybrid systems with state-based transitions quickly and efficiently. The structure imposed by the state-based transitions requirement allows the InVeriant software to ignore the individual transitions between hybrid system locations since all the necessary information about the transitions is contained in each location's invariant. The number of transitions between locations grows as the number of locations grow and as the number of passive state variables grow. Because the transitions are ignored, the complexity of the verification problem no longer depends on the number of state variables. In the verification method that involves the conversion from goal network to hybrid system followed by the use of a symbolic model checker, the number of passive state variables and particularly, the number of discrete states of each of those state variables contributed a large amount to the state space explosion of the system. Decoupling the passive state variables from the verification problem complexity will allow much larger systems to be verified using the InVeriant software.

The special structure of the systems that have state-based transitions allow for the reachability of the system to depend only on the states that the transitions constrain. Therefore, in many cases, a path to the unsafe locations does not need to be explicitly found. In the cases where a path is necessary, the structure of the system simplifies the process of finding a path since it can be shown that groups of locations are reachable from each other. Then, the path only needs to connect those groups. Rate constraints for continuous state variable can also be included in the unsafe set in InVeriant, which is not possible in many symbolic model checkers.

Because the InVeriant software was built to verify goal networks as well as hybrid systems, the information that it outputs is more conducive to the redesign of those goal networks. InVeriant will output not only the set of unsafe locations but also the goals and tactics that are common to all. PHAVer and HyTech output only the states of the state variables that allow the system to reach the unsafe set; even the unsafe locations are excluded from their output. Another benefit is not needing to translate the hybrid system into new notation or another language. Because the verification algorithm is so simple, there are only a few ways to reduce the complexity of the verification problem; either the number of locations in the hybrid system or the size of the unsafe set could be reduced. However, the modularity of the SBT Checker and the simplicity of the InVeriant verification algorithm suggest that this verification method could handle decently large goal network control programs or hybrid systems.

4.7 Conclusion

This verification method is designed for use with goal network control systems or hybrid systems that have state-based transitions. The state-based transitions requirement is a common sense restriction that leads to some very nice properties in the goal network verification. The modularity of the state-based transitions requirement makes the novel SBT Checker a useful design tool even for goal networks or hybrid systems that will not be verified. The InVeriant software leverages the reachability properties of the automaton imposed by the state-based transitions restriction and the properties of the state variables constrained to find the reachable unsafe set of the system quickly and efficiently. A significant example in Chapter 6 shows the speed of this verification method in relation to the conversion and PHAVer method described in Chapter 3. The simplicity and efficiency of this verification method suggest that it could be applicable to very large and complex systems, which is an important development in the use of reconfigurable goal-based control programs in autonomous robotic systems.