American Control Conference, June 2012

The Motion Grammar Calculus for Context-Free Hybrid Systems

Neil Dantam Mike Stilman

***Abstract*— This paper provides a method for deriving provably correct controllers for Hybrid Dynamical Systems with Context-Free discrete dynamics, nonlinear continuous dynamics, and nonlinear state partitioning. The proposed method models the system using a Context-Free Motion Grammar and specifies correct performance using a Regular language representation such as Linear Temporal Logic. The initial model is progressively rewritten via a calculus of symbolic transformation rules until it satisfies the desired specification.**

I. Introduction

Computer-controlled mechanisms such as robotic systems are being applied to increasingly complicated and critical tasks such as bomb-disposal, search-and-rescue, and medical assistance. To make these systems *safe and reliable*, we need models to describe the full and often complex requirements and then *verify* correct operation. Existing Hybrid Dynamical System models provide many capabilities to derive and verify these systems; however, models limited to Regular discrete dynamics or affine continuous dynamics struggle to handle complicated tasks such as robot manipulation and human-robot interaction. For systems requiring more complex behavior, the Motion Grammar [1, 2] provides the ability to represent, derive, and verify correct hybrid controllers.

This paper presents a method to derive correct controllers for complex hybrid dynamical systems such as robot manipulators. By modeling the hybrid system using our Motion Grammar [2], we can describe a broader class of systems than related approaches [3–5] at the cost, presently, of a less automated derivation procedure. Compared to other approaches, the use of a Context-Free language for discrete dynamics is a unique feature of the Motion Grammar in hybrid-systems models and provides the critical ability to use *memory* during the online operation of the system. This is strictly more powerful than the common Regular language class. A preliminary report on this derivation method was presented in [6]. We have extended this work by proving the validity of the derivation procedure, considering safe operating regions, and relating to existing techniques.

This paper is organized as follows: Sect. II considers related work. Sect. III provides background on our model for hybrid systems, the Motion Grammar. Sect. IV explains the challenge of how to partition the state space and describes a simple reachability test. Next, we introduce the primary goal of the paper, to transform a grammar that describes the system into one that correctly controls the system. Sect. V defines correctness for hybrid systems and introduces a lemma for proving allowable derivations. Sect. VI uses that lemma to produce and prove a set the symbolic transformations that can be applied to Motion Grammars. Sect. VII discusses safe and unsafe operating regions. Sect. VIII gives a sample derivation. Sect. IX concludes with future work.

This work was supported by NSF grants CNS1146352, *EAGER: Linguistic Task Transfer for Humans and Cyber Systems*, and CNS1059362, *The Motion Grammar Lab*. The authors are with the Robotics and Intelligent Machines Center in the Department of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA. ntd@gatech.edu, mstilman@cc.gatech.edu

II. Related Work

Hybrid Control is a quickly advancing research area describing systems with both discrete, event-driven, dynamics and continuous, time-driven, dynamics. Language and Automata Theory [7] was first applied to Discrete Event Systems (DES) by [8]. Hybrid Automata combine a Finite State Machine (FSM) with differential equations for each FSM state. This is a widely studied and utilized model [9–13]. The Motion Description Language is another approach that describes a hybrid system switching though a sequence of continuously-valued input functions [14, 15]. In this paper, we model hybrid systems using the Motion Grammar which represents continuous dynamics with differential equations and discrete dynamics using a Context-Free Grammar (CFG) [1]. This provides a few key advantages. A CFG is a more powerful language model than an FSM, so we can represent a broader class of discrete dynamics [7] while still allowing offline verification and efficient online control [2]. Additionally, while grammars have equivalent counterparts as push-down automata, we find the notation of a grammar more convenient as it simplifies both hierarchical decomposition and the symbolic transformations described in this paper. Thus we provide a hybrid systems model which builds on existing approaches in useful ways.

Model Checking is a technique for verifying discrete and hybrid systems by systematically testing whether the model satisfies a specified property [16, 17]. In the area of Hybrid Control, it is common to specify the desired properties using Linear Temporal Logic (LTL) and there are several approaches for automatically generating hybrid controllers from these specifications [3–5]. These approaches make assumptions about the language class, continuous dynamics, and discrete state. Namely, the language class is Regular, the continuous dynamics are affine, and the discrete state is a polytopic partitioning of the continuous state space. In this paper, we relax these assumptions. Our language class is Context-Free rather than Regular, our continuous dynamics may be arbitrary smooth functions, and we partition the continuous state space with nonlinear analytic surfaces. Additionally, our transformations permit the introduction of new regions or switching surfaces into the system language which is not considered in these previous methods. In this paper we

provide tools to derive controllers for a more general class of hybrid systems.

To model-check a hybrid system, we must know the feasibility of discrete transitions resulting from the continuous dynamics. In other words, it is important to know whether or not discrete transitions from continuous region A to B are possible. This is particularly important in the case where region B is a system failure state that should be avoided. The general answer to this question can be determined by solving the Hamilton-Jacobi-Isaacs partial differential equation (HJI PDE) to compute the backwards reachable set from the region of the transition [18, 19]. However, solving these HJI PDEs can be very difficult. The method of Barrier Certificates is a simpler approach that verifies avoidance of unsafe regions using a local test for uncrossable boundaries [20]. We apply this method in our approach for deriving safe system paths.

Model checkers also use the simulation and bisimulation relations between two systems, which show that one system may match the stepwise behavior of the other [16]. These relations are useful because they allow properties proven for one system to transfer to the other. Bisimulation for continuous and hybrid systems is studied in [21]. We use a simplified simulation relation in Sect. V to determine allowable steps in the stepwise derivation of a correct system.

III. Hybrid Systems and the Motion Grammar

Hybrid Dynamical Systems combine discrete and continuous dynamics; this is a useful model for digitally controlled mechanisms such as robots. The discrete dynamics of a hybrid system evolve as discrete state changes in response to *events*. The continuous dynamics evolve as continuous state varies over *time*. We define a hybrid system as,

Definition 1: A hybrid system is a tuple $F = (\mathcal{X}, \mathcal{Z}, \mathcal{U}, Q, Z, \delta, \rho)$ where,

$\mathcal{X} \subseteq \Re^m$	is the continuous state
$\mathcal{Z} \subseteq \Re^n$	is the continuous observation
$\mathcal{U} \subseteq \Re^p$	is the continuous input
Q	is the set of discrete state
Z	is the set of discrete events
$\delta : Q \times \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$	is the continuous dynamics
$\rho : Q \times Z \mapsto Q$	is the discrete dynamics

Note that in this definition, there are two different notions of a *path* for the system. First, there are paths as trajectories through the continuous state space $\mathcal{X}$. Second, there are paths as sequences of discrete events in Z. From this second case, the set of all paths as sequences of discrete events is the *language* of the system. We define a language using a *grammar* [7].

Definition 2: The language of system F, $\mathcal{L}_F$, is the set of all sequences of discrete events $z \in Z$ that F may encounter. $\mathcal{L}_F \subseteq Z^*$.

To analyze hybrid systems, we adopt an explicit representation for the combined dynamics called the *Motion Grammar*. The Motion Grammar combines Formal Language Theory and Modern Control Theory to represent hybrid systems using a Context-Free Grammar (CFG) and State-Space differential equations. A CFG represents the discrete dynamics using a set of production rules. Its representative power is equivalent to a Finite State Machine augmented with a pushdown stack which allows the controller to keep memory and select production expansions according to events that have long passed. The differential equations represent the continuous dynamics. We define the Motion Grammar as:

$\langle \text{task} \rangle$	$\rightarrow$	$\langle \text{load} \rangle \langle \text{task} \rangle \langle \text{unload} \rangle$	(1)
	$\mid$	$[\text{all}]$	(2)
$\langle \text{load} \rangle$	$\rightarrow$	$\{\dot{x} = f_1(x)\}\,[\text{loaded}]$	(3)
$\langle \text{unload} \rangle$	$\rightarrow$	$\{\dot{x} = f_2(x)\}\,[\text{unloaded}]$	(4)

Fig. 1. Example of a Motion Grammar

Definition 3: The Motion Grammar is a tuple $\mathcal{G}_M = (Z, V, P, S, \mathcal{X}, \mathcal{Z}, \mathcal{U}, \eta, K)$ where,

Z	is the set of events, or tokens
V	is the set of nonterminals
$P \subset V \times (Z \cup V)^*$	is the set of productions
$S \in V$	is the starting nonterminal
$\mathcal{X} \subseteq \Re^m$	is the continuous state space
$\mathcal{Z} \subseteq \Re^n$	is the continuous observation space
$\mathcal{U} \subseteq \Re^p$	is the continuous input space
$\eta : \mathcal{Z} \mapsto Z$	is the tokenizing function
$K \subset \mathcal{X} \mapsto \mathcal{X} \times \mathcal{U}$	is the set of semantic rules

We illustrate the notation for the Motion Grammar through the example in Fig. 1. Using the standard Backus-Naur Form for CFGs [7], we write nonterminals $v \in V$ between angle brackets $\langle\rangle$, tokens $z \in Z$ between square brackets $[]$, and semantic rules $k \in K$ between curly braces $\{\}$. Each line of Fig. 1 is a production $p \in P$. This grammar describes a loading and unloading task. The starting nonterminal $\langle \text{task} \rangle$ expands either to $\langle \text{load} \rangle \langle \text{task} \rangle \langle \text{unload} \rangle$, (1), or to the token $[\text{all}]$, (2). From this expansion, we see the system will repeatedly perform $\langle \text{load} \rangle$ operations until receiving an $[\text{all}]$ token. Then the system will perform $\langle \text{unload} \rangle$ operations of the same number as the prior $\langle \text{load} \rangle$ operations. This simple use of *memory* is possible with Context-Free systems. Regular systems are not powerful enough. The other two productions, (3) and (4), show the continuous dynamics and terminating condition for the $\langle \text{load} \rangle$ and $\langle \text{unload} \rangle$ operations. This example demonstrates the continuous dynamics and the Context-Free discrete dynamics of the Motion Grammar.

There are a few important advantages to the model used in the Motion Grammar. CFGs represent a balance between computational efficiency, representative power, and provable response. The fast algorithms for Context-Free *parsing* enable the robot to quickly react online without lengthy deliberative planning. CFGs are more powerful than Regular language representations while still permitting model-checking against a Regular language specification [7], allowing the system designer to tackle a broader class of problems and still prove desired response. Most robot tasks can be recursively divided into a number of simpler subtasks. The hierarchical nature of grammatical *productions* and the corresponding *parse trees* are well suited to representing this hierarchical task decomposition [2]. To our knowledge, this combination of hierarchical decomposability, Context-Free power, and formal verifiability are not found together in any prior methods for robot control.

IV. Tokenization and Reachability

Tokens are instantaneous events which drive the discrete system dynamics. In this work, we focus on a particular type of event: entry into some region of interest within the continuous state space $\mathcal{X}$. Thus, the string of tokens in Z represents an abstracted path of the system through $\mathcal{X}$, which we will use in Sect. V to prove *correctness*. Additionally, we will assume a fully observable system such that $x(t) \in \mathcal{X}$ is known for all t.

There are different types of regions in $\mathcal{X}$ that may be relevant. In [22], position and velocity regions are used to produce robot trajectories. Here, we consider the general case of any region of interest in state space. The region for an event may be an area where the underlying dynamics of the system change, such as at a contact or impact. Regions may also be areas where we want our input to the system to abruptly change, such as a mobile robot reaching a waypoint and switching to a new trajectory. A new token or event is generated when the system enters into the region.

Definition 4: The token set Z is a set of regions representing a complete partition of the state space $\mathcal{X}$. For $[\mathrm{x} \in \mathcal{R}_\mathrm{i}] \in Z$,

- $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset,\ i \neq j$, regions are nonoverlapping.
- $\bigcup_{i=1}^{|Z|} \mathcal{R}_i = \mathcal{X}$, regions cover the entire state space.

Note that it is trivial to relax this condition by splitting overlapping regions. We use the non-overlapping formulation to simplify our analysis.

A. Tokenization

Tokenization is the process of breaking up the unstructured observation into a stream of tokens or events. This is the first step our controller must take to parse the observation. Because tokenization is implemented via digital logic or software, we use a discrete-time formulation. Based on Def. 4, we define tokenization as follows,

Definition 5: A new token is generated when the system crosses the boundary between two regions. In discrete time, when $x_{k-1} \in \mathcal{R}_i \wedge x_k \in \mathcal{R}_j \wedge i \neq j$, token $[\mathrm{x} \in \mathcal{R}_\mathrm{j}]$ is appended to the input tape.

At each time step t_k, a discrete-time controller must compute which region it is in. If the region has changed since the previous step t_{k-1}, then the controller parses the token associated with the new region. One way to perform this tokenization is to express a region as bounded by codimension-1 manifolds $\mathcal{M}$ given as the level-set for some scalar function,

$$\mathcal{M} = \{x \ : \ s(x) = 0\} \tag{5}$$

Since the manifold is composed of all points where scalar $s(x) = 0$, the $\mathsf{sign}\,(s(x))$ indicates the side of the manifold, and consequently the region, of any system state.

B. Conservative Reachability with Barrier Certificates

By defining tokens as regions, we can use the continuous dynamics to predict the discrete dynamics. This will be used in Sect. VI, to transform the grammar. Observe that since tokens are regions, the set of discrete tokens which may be generated is equivalent to the set of reachable regions of continuous state. This problem has previously been addressed by others such [18] using Hamilton-Jacobi-Isaacs Partial Differential Equations (HJI PDE) to compute the backwards reachable set, and this method is indeed directly applicable to the Motion Grammar. However, it can often be very difficult to solve these HJI PDEs. If there are no known analytic solutions for the particular PDE of interest, it is often necessary to resort to numerical methods. The method of barrier certificates [20] instead considers behavior only along a specific boundary within the state space. This approach should be easier to evaluate and is directly applicable to our chosen method of tokenizing the state space.

We apply barrier certificates to the Motion Grammar using the Lie derivative along region boundaries. Consider the autonomous system dynamics given by smooth function $\dot{x} = f(x)$. Let the boundary between $\mathcal{R}_i$ and $\mathcal{R}_j$ be given by the codimension-1 manifold $\mathcal{M}$, $c = s(x)$. The normal vector to $\mathcal{M}$ at point x is $\nabla s(x)$. To determine if the system will cross $\mathcal{M}$ at point x, we relate the direction of $\nabla s(x)$ and $f(x)$ using the sign of the Lie derivative, $\mathscr{L}_f s$,

Theorem 1: Let $\mathcal{M} = \{x \ : \ s(x) = 0\}$. If $\mathscr{L}_f s(x) < 0\ \forall x \in \mathcal{M}$, the system will never cross $\mathcal{M}$. If $\exists x \in \mathcal{M},\ \mathscr{L}_f s(x) > 0$, the system *may* cross $\mathcal{M}$.

Proof: Consider some $p \in \mathcal{M}$. Then $\mathscr{L}_f s(p) = \nabla s(p) \cdot f(p) = \|\nabla s(p)\| \, \|f(p)\| \cos\theta$, where θ is the angle between $\nabla s(p)$ and $f(p)$. When $\cos\theta > 0$, the system moves off $\mathcal{M}$ in the direction of increasing $s(x)$. When $\cos\theta < 0$, the system moves off $\mathcal{M}$ in the direction of decreasing $s(x)$. Since $\mathsf{sign}\,(\cos\theta) = \mathsf{sign}\,(\mathscr{L}_f s)$ we can use $\mathsf{sign}\,(\mathscr{L}_f s)$ to test which side of $\mathcal{M}$ the system will move to from p. If there is no p for which $\mathscr{L}_f s > 0$, then the system cannot move off the manifold to cross it. If there is any p for which $\mathscr{L}_f s > 0$, then from that p, the system will move off the manifold and thus cross it. ∎

Thm. 1 thus shows whether one region is directly reachable from another based on system dynamics $\dot{x} = f(x)$. It is conservative because it only says whether a boundary crossing occurs based on the local condition. The crossing occurs when both $\mathscr{L}_f s > 0$, and the global dynamics brings x to the local neighborhood of the boundary.

We can express $\mathscr{L}_f s$ only along $\mathcal{M}$ by parameterizing $\mathcal{M}$ by some $v \in \Re^{n-1}$, where $\mathcal{X} \in \Re^n$.

$$\mathcal{M} = \{x \ : \ x = \phi(v)\} \tag{6}$$

For example, if $\mathcal{M}$ is a hyperplane, then $\phi(v) = Mv$, where M is a $n \times (n-1)$ matrix. If $\mathcal{M}$ is a hypersphere, the ϕ can be defined to transform spherical coordinate vector v to Cartesian coordinates x. Then, we consider $\mathsf{sign}\,(\mathscr{L}_f s\,[\phi\,(v)])$ to determine if the boundary may ever be crossed.

C. Reachability Example

Consider a region bounded by an ellipse centered on the origin as shown in Fig. 2.

$$M \equiv c = \frac{{x_1}^2}{{a_1}^2} + \frac{{x_2}^2}{{a_2}^2} \tag{7}$$

$$\nabla s(x) = \begin{bmatrix} \frac{x_1}{{a_1}^2} & \frac{x_2}{{a_2}^2} \end{bmatrix} \tag{8}$$

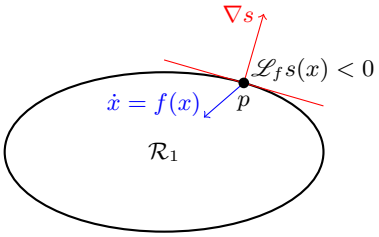


Fig. 2. A region given by $1 = \frac{x^2}{2^2} + y^2$ and tangent planes by $1 + \frac{{x_0}^2}{2^2} + {y_0}^2 = \frac{2x_0}{2^2}x + 2y_0 y$. The system evolves by $\dot{x} = -x$, indicating that it stays within $\mathcal{R}_1$ at the boundary.

1) System 1: Consider time-driven dynamics $\dot{x} = -x$. This gives us $\mathscr{L}_f s = -\frac{{x_1}^2}{{a_1}^2} - \frac{{x_2}^2}{{a_2}^2}$. Since this is always negative, the system will not cross the boundary.

2) System 2: Consider time-driven dynamics $\dot{x} = \begin{bmatrix} y - x & -x - y \end{bmatrix}^T$. This gives us $\mathscr{L}_f s = \frac{x_1 x_2}{{a_1}^2} - \frac{x_1 x_2}{{a_2}^2} - \frac{{x_1}^2}{{a_1}^2} - \frac{{x_2}^2}{{a_2}^2}$. We can use a parameterization of the manifold by v, $x = \begin{bmatrix} a_1 \cos v & a_2 \sin v \end{bmatrix}$ to rewrite $\mathscr{L}_f s = \frac{a_2}{a_1} c_v s_v - \frac{a_1}{a_2} c_v s_v - 1$. From this, we see that if $\frac{a_1}{a_2} > 1 + \sqrt{2}$ or $\frac{a_2}{a_1} > 1 + \sqrt{2}$, the Lie derivative will be positive for some values of v meaning that the system will cross the boundary from some, but not all v.

D. Additional Event Types

In addition to the region-based events of Def. 4-5 which result from the controllable continuous dynamics, we can model events resulting from uncontrollable continuous dynamics or purely-discrete dynamics as well. Such events may include faults, limit conditions, and even human actions or spoken words. All can be included as tokens in the grammar. In this paper however, we focus on the controllable region-entry events because it is by appropriately controlling these events that we can transform the grammar to achieve correctness.

V. Process to Derive Correct Grammars

To transform a *grammar which describes* the system into a *grammar that correctly controls* the system, we must consider which transformations are possible. Only certain transformations of the grammar are valid. We are particularly concerned with ensuring that our transformations maintain the property that the derived grammar ${\mathcal{G}_M}'$ is complete – that ${\mathcal{G}_M}'$ describes all *paths* that were possible in the original grammar $\mathcal{G}_M$, because $\mathcal{G}_M$ is an accurate representation of the hybrid system by definition. In this section we first define and explain the correctness property we wish to achieve. Second, we show examples of transformations that lead to complete and incomplete ${\mathcal{G}_M}'$. Third, we define completeness and present a method for determining whether a class of transformations results in complete derived grammars, ${\mathcal{G}_M}'$. Finally, in Sect. VI, we use this method to prove that certain transformations are valid for all Motion Grammars, forming a calculus that can be used to transform a Motion Grammar to a *correct* Motion Grammar.

A. Correctness

To guarantee that our system is safe, we must prove that it satisfies some property describing correct operation. Using the Motion Grammar as the model for our system, we have a natural way of expressing *correctness*. Observe from Def. 5, that the language $\mathcal{L}_{\mathcal{G}_M}$ of Motion Grammar $\mathcal{G}_M$ represents the set of all possible paths that the system may take. For $\mathcal{G}_M$ to be correct, $\mathcal{L}_{\mathcal{G}_M}$ must be within some set of allowable paths. This set of allowable paths is in fact another language $\mathcal{L}_s$ which we may specify using Regular-equivalent models such as Finite State Machines or Linear Temporal Logic [2].

Definition 6: Given model $\mathcal{G}_M$ and a language of correct operation $\mathcal{L}_s$, the language defined by the $\mathcal{G}_M$ is $\mathcal{L}_{\mathcal{G}_M}$ and $\mathsf{correct}\,\{\mathcal{G}_M, \mathcal{L}_\mathsf{s}\} \iff \mathcal{L}_{\mathcal{G}_M} \subseteq \mathcal{L}_s$

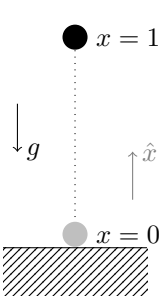
If our system is *not* correct, we must modify it to be correct. We propose an iterative derivation process to transform some initial grammar $\mathcal{G}_M$ where $\neg\mathsf{correct}\,\{\mathcal{G}_M, \mathcal{L}_\mathsf{s}\}$ into some derived grammar ${\mathcal{G}_M}^*$ such that $\mathsf{correct}\,\{{\mathcal{G}_M}^*, \mathcal{L}_\mathsf{s}\}$. At each step of this derivation process, we must ensure that any transformation $\mathcal{G}_M \rightsquigarrow {\mathcal{G}_M}'$ preserves both accuracy of the model and our ability to guarantee correctness. To determine which are allowable derivation steps, we use a *simulation* relation, $\mathcal{G}_M \preceq {\mathcal{G}_M}'$, meaning our derived grammar ${\mathcal{G}_M}'$ is able to simulate the operation of the original grammar $\mathcal{G}_M$.

B. Example of Complete and Incomplete Derivations

To illustrate the types of transformations which are and are not possible, consider the trivial system in Fig. 3(a). Here, a ball is dropped from height $x_0 = 1$. Its collision with the ground is perfectly inelastic, so it will stop as soon as it reaches $x = 0$. An initial grammar describing this system is shown in Fig. 3(b). From inspection, we can remove the expansion for $\langle \mathrm{H} \rangle$ and still have the valid grammar $\mathcal{G}'_a$ in Fig. 3(c). This is because the initial region for $\langle \mathrm{H} \rangle$, $x \geq 2$, is unreachable from the production for $\langle \mathrm{S} \rangle$. However, if we remove the production for $\langle \mathrm{G} \rangle$ as well, Fig. 3(d), then we no longer have a valid grammar. This is because $\mathcal{G}$ and $\mathcal{G}'_b$ are describing different sets of paths. Namely, $\mathcal{G}'_b$ takes a single path – tunneling to the center of the Earth – while $\mathcal{G}$ stops at $x = 0$. Thus, we see that there must be a specific relation between initial and derived grammars in order to maintain a faithful representation of the system. This specific relation is the simulation, $\mathcal{G} \preceq \mathcal{G}'$.

C. Completeness and a Simulation Lemma

A *complete* derived system model G' represents everything the initial system G could do. Slightly more formally, G'



(a) System

$$\begin{array}{lcl} \langle \mathrm{S} \rangle & \rightarrow & [\mathrm{x} = 1]\,\{\dot{x} = -mg\}\,\langle \mathrm{A} \rangle \\ \langle \mathrm{A} \rangle & \rightarrow & \langle \mathrm{G} \rangle | \langle \mathrm{H} \rangle \\ \langle \mathrm{G} \rangle & \rightarrow & [\mathrm{x} = 0]\,\{\dot{x} = 0\} \\ \langle \mathrm{H} \rangle & \rightarrow & [\mathrm{x} \geq 2]\,\{\dot{x} = -mg\} \end{array}$$

(b) Initial Grammar, $\mathcal{G}$

$$\begin{array}{lcl} \langle \mathrm{S} \rangle & \rightarrow & [\mathrm{x} = 1]\,\{\dot{x} = -mg\}\,\langle \mathrm{G} \rangle \\ \langle \mathrm{G} \rangle & \rightarrow & [\mathrm{x} = 0]\,\{\dot{x} = 0\} \end{array}$$

(c) Simulating Grammar, $\mathcal{G}'_a$

$$\langle \mathrm{S} \rangle \quad \rightarrow \quad [\mathrm{x} = 1]\,\{\dot{x} = -mg\}$$

(d) Non-Simulating Grammar, $\mathcal{G}'_b$

Fig. 3. A dropped ball with a perfectly-inelastic collision. Examples of simulating and nonsimulating grammars. $\mathcal{G} \preceq \mathcal{G}'_a$, $\mathcal{G} \not\preceq \mathcal{G}'_b$

describes the set of all *paths* that the initial system G could take. This property is given as the simulation $G \preceq G'$. The reverse may not hold. That is, G' may be able to take paths which G cannot. The concrete definition of a *path* depends on type of system we are dealing with. For discrete Transition Systems, a path is the sequence states and transitions the system takes. For continuous systems, a path is a trajectory though its state space. Simulation paths for a variety of systems are detailed in [21]. For our goal of deriving correct grammars, we will define paths and the simulation $\mathcal{G}_M \preceq_c \mathcal{G}_M{}'$ as follows,

Definition 7: Given models $\mathcal{G}_M$ and $\mathcal{G}_M{}'$ with $x, x', u, u' \in \mathcal{X}, \mathcal{X}', \mathcal{U}, \mathcal{U}'$, then $\mathcal{G}_M \preceq_c \mathcal{G}_M{}' \iff (x_0 = x_0{}' \wedge u(t) = u'(t) \implies x(t) = x'(t))$.

This relation shows that $\mathcal{G}_M$ and $\mathcal{G}_M{}'$ follow the same path provided that $\mathcal{G}_M$ is given the same initial conditions and inputs as $\mathcal{G}_M{}'$. The initial conditions are the first token of the starting nonterminal. For example, if $\langle \mathrm{A} \rangle$ begins with token $[\mathrm{x} \in \mathcal{R}]$, the initial condition of $\langle \mathrm{A} \rangle$ is $\mathcal{R}$. It is critical that $\mathcal{G}_M$ and $\mathcal{G}_M{}'$ are given the same input. The input u the our only way to influence the path of the system to make it *correct*. To shorten the notation, let $\mathcal{G}_M|_{u',x'_0}$ be Motion Grammar $\mathcal{G}_M$ subject to initial condition x'_0 and input u', and likewise for the language $\mathcal{L}_{\mathcal{G}_M|u',x'_0}$. We merge the simulation and correctness definitions as follows.

Lemma 1: $\mathcal{G}_M \preceq_c \mathcal{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x'_0 \wedge \mathsf{correct}\left\{\mathcal{G}_M{}', \mathcal{L}'_\mathsf{s}\right\} \implies \mathsf{correct}\left\{\mathcal{G}_M|_{\mathrm{x}'_0,\mathrm{u}'}, \mathcal{L}'_\mathsf{s}\right\}$, where $\mathcal{L}'_s, \mathcal{L}'_{\mathcal{G}_M}, \mathcal{L}'_{\mathcal{G}_M|u',x'_0} \subseteq Z'^*$.

Proof: From Def. 7, $\mathcal{G}_M \preceq_c \mathcal{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x'_0 \implies x(t) = x'(t)$. From Def. 5, $x(t) = x'(t) \implies \mathcal{L}'_{\mathcal{G}_M} = \mathcal{L}'_{\mathcal{G}_M|u',x'_0}$. From Def. 6 and Def. 7, $\mathcal{L}'_{\mathcal{G}_M} = \mathcal{L}'_{\mathcal{G}_M|u',x'_0} \wedge \mathsf{correct}\left\{\mathcal{G}_M, \mathcal{L}'_\mathsf{s}\right\} \implies \mathsf{correct}\left\{\mathcal{G}_M|_{\mathrm{x}'_0,\mathrm{u}'}, \mathcal{L}'_\mathsf{s}\right\}$ ■

From this, we can determine allowable derivations based on simulation $\preceq_c$, initial state x_0, and input u. Note that since derivations need not preserve the discrete token set Z, we must specify the correctness language $\mathcal{L}'_s$ over token set Z'. With Lem. 1, we can now identify a set of symbolic transformations to apply to any Motion Grammar.

VI. The Motion Grammar Calculus

To derive grammars for safe, Context-Free systems, we introduce a calculus of symbolic transformation rules for constructing a correct hybrid controller. This process begins with some initial model the hybrid system. The rules then *rewrite* the model step-by-step, always adhering to the simulation relation and Lem. 1 so that the correctness of the derived model implies correctness for our system. Thus we effectively change the system language until it satisfies our specification. Through this derivation process, we modify the behavior of the system to make it *correct*.

In each rule, we start with some grammar $\mathcal{G}_M$ and derive a grammar $\mathcal{G}_M{}'$. A rule is valid only if $\mathsf{correct}\left\{\mathcal{G}_M{}', \mathcal{L}'_\mathsf{s}\right\} \implies \mathsf{correct}\left\{\mathcal{G}_M|_{\mathrm{x}'_0,\mathrm{u}'}, \mathcal{L}'_\mathsf{s}\right\}$, which we will prove using Lem. 1. In the notation for these rules, we specify some precondition on the structure of elements of the production set P, and then specify the resulting token set Z', nonterminal set V' and production set P'.

A. Transforms

Input First, consider the very simple transformation of specifying an input u to illustrate this process. When the continuous dynamics are in the form $\dot{x} = f_0(x, u)$, we can always specify an input u.

Transform 1: Given $p = A \to \alpha f_0(x, u)\beta$, define $f(x) = f_0(x, g(x))$. Then the new production set is $P' = P - p \cup \{A \to \alpha f(x)\beta\}$.

Proof: For this transform to be allowable, it must satisfy the preconditions of Lem. 1. Namely we must have $\mathcal{G}_M \preceq_c \mathcal{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x'_0$. Using $\mathcal{G}_M{}'$ to control the system means our input is given by $u'(t)$. Since we do not change the start symbol S, the initial condition $x_0 = x'_0$. Finally, in the modified production p, $\dot{x}' = f(x) = f_0(x, g(x)) = f_0(x, u'(t))$, so $\dot{x}' = \dot{x}|_{u'}$. Thus, $x_0 = x'_0 \wedge \dot{x}' = \dot{x}|_{u'} \implies x(t) = x'(t) \implies \mathcal{G}_M \preceq_c \mathcal{G}_M{}'$. Thus, we satisfy the preconditions of Lem. 1 and can use $\mathsf{correct}\left\{\mathcal{G}_M{}', \mathcal{L}'_\mathsf{s}\right\}$ to decide $\mathsf{correct}\left\{\mathcal{G}_M, \mathcal{L}'_\mathsf{s}\right\}$. ■

Token Splitting A region represented by a token can be split into two regions, creating two new tokens. We then create new productions for these new regions.

Transform 2: Given some $\zeta_0 = [\mathrm{x} \in \mathcal{R}_0] \in Z$, create tokens $\zeta_1 = [\mathrm{x} \in \mathcal{R}_1]$ and $\zeta_2 = [\mathrm{x} \in \mathcal{R}_2]$ such that $\mathcal{R}_1 \cup \mathcal{R}_2 = \mathcal{R}_0 \wedge \mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$ and update token set $Z' = Z - \zeta_0 \cup \{\zeta_1, \zeta_2\}$. The new nonterminal set is $V' = V \cup \{A_0, A_1, A_2, A_3, A_4\}$. The new production set is $P' = P - \{(A \to \alpha_1\zeta_0\kappa\alpha_2) \in P\} \cup \{(A \to \alpha_1 A_0), (A_0 \to A_1|A_2) : (A \to \alpha_1\zeta_0\kappa\alpha_2) \in P\} \cup \{(A_1 \to \zeta_1\kappa A_3), (A_2 \to \zeta_2\kappa A_4) : (A \to \alpha_1\zeta_0\kappa\alpha_2) \in P\} \cup \{(A_3 \to A_2|\alpha_2), (A_4 \to A_1|\alpha_2) : (A \to \alpha_1\zeta_0\kappa\alpha_2) \in P\}$.

Proof: Following the form of the proof for Transform 1, we need to show that $x(t) = x'(t)$. Since we have not modified the continuous dynamics, we need only show that the discrete dynamics of $\mathcal{G}_M{}'$ permit the same paths as in $\mathcal{G}_M$. In $\mathcal{G}_M$, for some production $A \to \alpha_1\zeta_0\kappa\alpha_2$, the system may pass from the region of α_1 into $\mathcal{R}_0$ and then on to the region of α_2. When we split $\mathcal{R}_0$, there are six cases to consider: $\mathcal{R}_{\alpha_1} \to \mathcal{R}_1$, $\mathcal{R}_{\alpha_1} \to \mathcal{R}_2$, $\mathcal{R}_1 \to \mathcal{R}_{\alpha_2}$, $\mathcal{R}_2 \to \mathcal{R}_{\alpha_2}$, $\mathcal{R}_1 \to \mathcal{R}_2$, $\mathcal{R}_2 \to \mathcal{R}_1$. These cases are handled respectively by the added productions $A_0 \to A_1$, $A_0 \to A_2$, $A_3 \to \alpha_2$, $A_4 \to \alpha_2$, $A_3 \to A_2$, and $A_4 \to A_1$. Thus all paths from $\mathcal{G}_M$ and matched by $\mathcal{G}_M{}'$, so $\mathcal{G}_M \preceq_c \mathcal{G}_M{}'$. ■

Adjacency Pruning If two regions in state space are not adjacent, then the system may not pass directly between them. Thus we can eliminate productions which allow this to happen.

Transform 3: For $p_1 = A \to r_A\kappa_A B$, $B \to \beta_1| \ldots |\beta_n$, if r_A is not adjacent to $\mathcal{R}_0(\beta_n)$ WLOG, then $P' = P - p_1 \cup \{A \to r_A\kappa_A B'\} \cup \{B' \to \beta_1| \ldots |\beta_{n-1}\}$

Proof: To show $x(t) = x'(t)$, we prove by contradiction. We can say that $x(t) = x'(t)$ if and only if the removed production is unreachable, that is, the system $\mathcal{G}_M$ will never pass from r_A to $\mathcal{R}_0(\beta_n)$. Now, assume $x(t) \neq x'(t)$. Then $\mathcal{G}_M$ must pass from r_A to $\mathcal{R}_0(\beta_n)$. Since these two regions are not adjacent, this is a contradiction. Thus, we must have $x(t) = x'(t)$, so $\mathcal{G}_M \preceq_c \mathcal{G}_M{}'$. ■

Barrier Pruning The continuous dynamics f provide information that may be used to remove grammar productions. Using Thm. 1, we can show whether the system following $\dot{x} = f(x)$ may cross into any of the regions specified in the grammar.

Transform 4: For $p_1 = A \to r_A \kappa_A B$, $B \to \beta_1 | \ldots | \beta_n$, if WLOG $\mathscr{L}_f s(p) < 0$ for all p along the level set $s(x) = 0$ which borders regions r_A and $\mathcal{R}_0(\beta_n)$, then $P' = P - p_2$.

Proof: To show $x(t) = x'(t)$, we prove by contradiction. We can say that $x(t) = x'(t)$ if and only if the removed production is unreachable, that is, the system $\mathcal{G}_M$ will never pass from r_A to $\mathcal{R}_0(\beta_n)$. Now, assume $x(t) \neq x'(t)$. Then $\mathcal{G}_M$ may pass from r_A to $\mathcal{R}_0(\beta_n)$. By Thm. 1 and $\mathscr{L}_f s(p) < 0 \, \forall p \in \{x : s(x) = 0\}$, this is a contradiction. Thus, we must have $x(t) = x'(t)$, so $\mathcal{G}_M \preceq_c \mathcal{G}_M{}'$. ∎

Bounce Pruning If the system in moving from region r_1 to region r_2 will immediately reenter r_1, then we can eliminate productions showing that the system will pass through r_2 into some third region.

Transform 5: Given productions $p_1 = A \to r_1 \kappa_A B$, $p_2 = B \to r_2 \kappa_B C$, and $p_3 = C \to r_1 \kappa_B \alpha$, if $\mathscr{L}_{\kappa_B} s_{21}(x) > 0 \; \forall x \in \{x : s_{21}(x) = 0\}$, then $P' = P - p_1 - p_2 \cup \{(A \to r_1 \kappa_A B'), (B' \to r_2 \kappa_B r_1 \kappa_B \alpha)\}$

Proof: To show $x(t) = x'(t)$, we must account for the removed productions by proving the system will not pass from r_1 to r_2 to some other region in $\mathcal{R}_0(C) - r_1$. Instead, the system must immediately return to r_1 from r_2. This is given directly by Thm. 1 and the Lie derivative $\mathscr{L}_{\kappa_B} s_{21}(x) > 0 \; \forall x \in \{x : s_{21}(x) = 0\}$, indicating that under mode κ_B this system will move off $s_{12}(x) = -s_{21}(x) = 0$ in the direction of r_1. Thus, p_2 will expand nonterminal C with p_3, according to the sequence given by the additional productions in P'. ∎

B. Using the Calculus to Enforce Correctness

These rules provide important capabilities to work with hybrid models. Transform 1 allows us to specify the input to the robot to drive toward desired tokens. Transform 2 allows us to introduce new surfaces where we can discretely switch control inputs. Transform 3, Transform 4, and Transform 5 allow us to *remove* productions from the grammar. We can use this to satisfy a correctness constraint by eliminating certain bad productions causing the constraint violation. Thus, we can systematically produce a grammatical model implementing correct operation.

VII. Safe Regions and New Switching Surfaces

Using our conservative reachability test from Thm. 1 and the rewrite rules of Sect. VI, we can identify safe operating regions and consequently switching surfaces to maintain safe operation. Consider the example in Fig. 4 where there is some region with *good* and *bad* exit boundaries. Here, we wish to ensure that the system will cross the *good* manifold bounding $\mathcal{R}_0$, $\mathcal{M}_1$, and that it will not cross the other *bad* other manifold $\mathcal{M}_2$. Assume the continuous dynamics in $\mathcal{R}_0$ are $\dot{x} = f_0(x, u)$ and we have some controller $u = g(x)$, so that we can write the resulting system as the smooth function $\dot{x} = f_0(x, g(x)) = f(x)$. Then we can consider the Lie

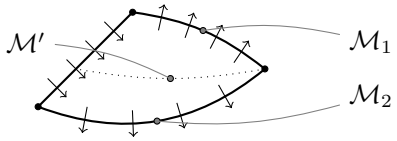


Fig. 4. Splitting the region based on $\dot{x} = f(x)$ at each boundary.

derivatives along each manifold $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathscr{L}_f s_1$ and $\mathscr{L}_f s_2$ respectively. If both $\mathscr{L}_f s_1$ and $\mathscr{L}_f s_2$ are positive, then for x close to $\mathcal{M}_1$ and $\mathcal{M}_2$, the system will cross both the safe and the unsafe boundaries. To indicate the safe subregion of $\mathcal{R}_0$, we introduce a new boundary $\mathcal{M}'$ separating $\mathcal{M}_1$ and $\mathcal{M}_2$. $\mathcal{M}'$ bounds the *region of inevitable collision*. By always staying to one side of $\mathcal{M}'$, we can be assured that by applying input function $u = g(x)$, we will not cross the unsafe boundary.

For this boundary notion to be useful during the online control of the system, we must be able to quickly test during each control cycle on which side of the boundary the system currently resides. If we express the manifold as $\mathcal{M}' = \{x : s(x) = 0\}$, then the sign of $s(x)$ will give the current side of the manifold. Thus, we must find some representation for $s(x)$ to evaluate in our control program.

We can describe the boundary between safe and unsafe regions based on the idea that boundary $\mathcal{M}'$ is composed of the family of integral curves leading to the intersection of the safe and unsafe exit surfaces. That is, for every point along $\mathcal{M}'$, the system dynamics $\dot{x} = f(x)$ will drive x to the intersection of the two exit surfaces $\mathcal{M}_1$ and $\mathcal{M}_2$. This description leads to the following two geometric properties of $\mathcal{M}' = \{x : s(x) = 0\}$. First, the gradient of $s(x)$ is orthogonal to the system vector field $f(x)$. Second, the gradient of $s(x)$ is orthogonal to the intersection $\mathcal{M}_C$ of our two exit surfaces $\mathcal{M}_1$ and $\mathcal{M}_2$.

Theorem 2: $\nabla s(x) \perp f(x)$

Proof: Consider some point p on $\mathcal{M}'$, $s(p) = 0$. Point p moves according to $\dot{p} = f(p)$, and $\frac{d}{dt} s(p) = 0$. $\frac{d}{dt} s(p) = \frac{\partial s}{\partial p}^T \frac{dp}{dt} = (\nabla s)(f) = 0$. Since the inner product of $\nabla s(x)$ and $f(x)$ is always zero, they must be orthogonal. ∎

Theorem 3: Let $\mathcal{M}_c = \left\{x : x = p(v), v \in \Re^{n-2}\right\}$, then $\nabla s(x) \perp \frac{\partial p}{\partial v_i}$.

Proof: Since $\mathcal{M}_c \subset \mathcal{M}'$, $\frac{\partial s(p(v))}{\partial v_i} = 0$. Then $\frac{\partial s(p(v))}{\partial v_i} = \frac{\partial s}{\partial p}^T \frac{\partial p}{\partial v_i} = (\nabla s)\left(\frac{\partial p}{\partial v}\right)$. ∎

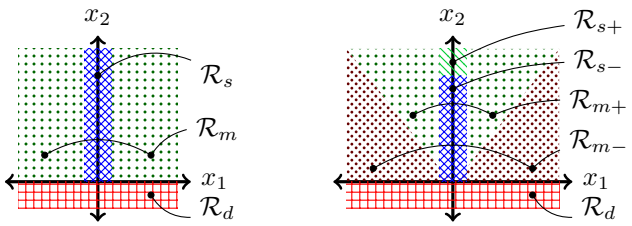
From Thm. 2 and 3, we can derive ∇s by finding the vector that satisfies the two orthogonal relations. For any vector ξ, we can find a vector ξ' orthogonal to $\psi = f(x)$ or $\psi = \frac{\partial p}{\partial v}$ using a projection.

$$\xi' = \xi - \mathsf{proj}_\psi \xi = \xi - \frac{\langle \xi, \psi \rangle}{\|\psi\|^2} \psi \tag{9}$$

Thus, we form ∇s by symbolically applying the Gram-Schmidt procedure.

$$\nabla s = \ell - \mathsf{proj}_{f(x)} \ell - \mathsf{proj}_{\frac{\partial p}{\partial v_1}} \ell - \ldots - \mathsf{proj}_{\frac{\partial p}{\partial v_{n-2}}} \ell \tag{10}$$

Equation (10) is useful when we can express ∇p in a form that does not include any v. For example, when $\mathcal{M}_c$ is linear, $p(v) = Mv$ and $\nabla p = M$. Then, with ∇s known, we can solve the following initial value problem to find $s(x)$,



(a) Initial Regions (b) Derived Regions

$$\langle \mathrm{S} \rangle \rightarrow [\mathrm{s}] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{M} \rangle \quad (11)$$
$$\langle \mathrm{M} \rangle \rightarrow [\mathrm{m}] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{N} \rangle \quad (12)$$
$$\langle \mathrm{N} \rangle \rightarrow \langle \mathrm{S} \rangle \,|\, \langle \mathrm{D} \rangle \quad (13)$$
$$\langle \mathrm{D} \rangle \rightarrow [\mathrm{d}] \{ \dot{x} = 0 \} \quad (14)$$

(c) Initial Grammar

Fig. 5. Initial grammar for 1-dimensional battery robot.

$$s_0 = g(0) \qquad \frac{\partial s}{\partial x_i} = (\nabla s)_i$$

While (10) provides an analytic form for the gradient, solving (11) is nontrivial. Approximations such as a Taylor series or Padé approximant can be directly computed from (10).

VIII. Example Derivation

We now demonstrate this derivation approach with a simple example. Consider a mobile robot moving in one dimension, x_1, with a battery, x_2, that discharges as it moves. There is a recharging station at the zero position. When the battery level falls to zero, the robot can no longer operate. The continuous state space and initial grammar are shown in Fig. 5(a). The initial grammar for this system is given in Fig. 5(c).

Because we want the robot to keep operating, its battery should never run down. This constraint is expressed in LTL:

$$G_s = \square\,(\neg\,[\mathrm{d}]) \quad (15)$$

The initial grammar does not satisfy (15). For example, the grammar generates the string [s] [m] [d], which violates the constraint. Thus, we must apply our transformations to the grammar in order to make it correct.

There are two main ideas to satisfying (15). First, the robot must not go far too from the charger, ensuring that it has enough charge to return. Second, the robot must wait in the charger to recharge. We apply these ideas in the derivation:

1) In (12), apply Transform 2 to split [m]. See new regions in Fig. 5(b).

$$\langle \mathrm{M} \rangle \rightarrow [\mathrm{m}] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{N} \rangle \quad (16)$$
$$\mid \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \quad (17)$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (18)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (19)$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2 \rangle | \langle \mathrm{N} \rangle \quad (20)$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{N} \rangle \quad (21)$$

2) Duplicate (18) and replace in (21).

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \quad (22)$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (23)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (24)$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2 \rangle | \langle \mathrm{N} \rangle \quad (25)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (26)$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}_1 \rangle \langle \mathrm{M}'_1 \rangle | \langle \mathrm{N} \rangle \quad (27)$$

3) In (24) and (26), apply Transform 1 to specify input $u = -x$.

$$\vdots$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (28)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (29)$$
$$\vdots$$

To simplify notation: $\kappa_- \equiv \left\{ \dot{x} = [-x \quad -|x|]^T \right\}$.

4) Consider the Lie derivative between [m+] and [m−] according to κ_-. $\mathcal{M}$: $-1.5x_1 + x_2 = \epsilon$, $x = [1.5v \quad v + \epsilon]^T$, $v > 0$, $\nabla s = [-1.5 \quad 1]^T$, $\mathscr{L}_{\kappa_-} s|_{\mathcal{M}} = 1.5^2 v + v + \epsilon$, always positive. In (28), (27), (29), apply Transform 5,

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \quad (30)$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (31)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (32)$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2 \rangle \langle \mathrm{M}'_2 \rangle | \langle \mathrm{N} \rangle \quad (33)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (34)$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}'_1 \rangle | \langle \mathrm{N} \rangle \quad (35)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (36)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (37)$$
$$\langle \mathrm{M}'_2 \rangle \rightarrow [\mathrm{m}-]\, \kappa_-\, [\mathrm{m}+]\, \kappa_- \langle \mathrm{M}_3 \rangle \quad (38)$$

Thus, when the system moves from [m+] to [m−], it will switch to mode κ_- to return to the charging station.

5) Apply Transform 3 and Transform 4.

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \quad (39)$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \langle \mathrm{M}'_3 \rangle \quad (40)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (41)$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}'_2 \rangle | \langle \mathrm{N} \rangle \quad (42)$$
$$\langle \mathrm{M}'_3 \rangle \rightarrow \langle \mathrm{M}'_2 \rangle | \langle \mathrm{S} \rangle \quad (43)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [u \quad -|u|]^T \right\} \langle \mathrm{M}_3 \rangle \langle \mathrm{M}'_3 \rangle \quad (44)$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}'_1 \rangle | \langle \mathrm{N} \rangle \quad (45)$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_4 \rangle \quad (46)$$
$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = [-x \quad -|x|]^T \right\} \langle \mathrm{M}_3 \rangle \quad (47)$$
$$\langle \mathrm{M}'_2 \rangle \rightarrow [\mathrm{m}-]\, \kappa_-\, [\mathrm{m}+]\, \kappa_- \langle \mathrm{M}_3 \rangle \langle \mathrm{S} \rangle \quad (48)$$

6) Switch now to (11) and split [s].

$$\langle \mathrm{S} \rangle \rightarrow [\mathrm{s}] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{M} \rangle \quad (49)$$
$$\mid S \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \quad (50)$$
$$\langle \mathrm{S}_1 \rangle \rightarrow [\mathrm{s}+] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{S}_3 \rangle \quad (51)$$
$$\langle \mathrm{S}_2 \rangle \rightarrow [\mathrm{s}-] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{S}_4 \rangle \quad (52)$$
$$\langle \mathrm{S}_3 \rangle \rightarrow \langle \mathrm{S}_2 \rangle | \langle \mathrm{M} \rangle \quad (53)$$
$$\langle \mathrm{S}_4 \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{M} \rangle \quad (54)$$

7) Give input $u = -x$.

$$\langle \mathrm{S} \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \quad (55)$$
$$\langle \mathrm{S}_1 \rangle \rightarrow [\mathrm{s}+] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{S}_3 \rangle \quad (56)$$
$$\langle \mathrm{S}_2 \rangle \rightarrow [\mathrm{s}-] \left\{ \dot{x} = [u \quad k_s]^T \right\} \langle \mathrm{S}_4 \rangle \quad (57)$$
$$\langle \mathrm{S}_3 \rangle \rightarrow \langle \mathrm{S}_2 \rangle | \langle \mathrm{M} \rangle \quad (58)$$
$$\langle \mathrm{S}_4 \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{M} \rangle \quad (59)$$

8) Consider Lie derivative between [s−] and [m] according to $\left\{\dot{x} = [-x \quad k_s]^T\right\}$:

- $\mathcal{M}$: $s(x) = x_1 = \pm\epsilon$, $x = [\pm\epsilon \quad v]^T$, $v > 0$
- $\nabla s = [\pm 1 \quad 0]^T$
- $\mathscr{L}_{\kappa_-} s|_{\mathcal{M}} = -\epsilon$ always negative

Consider Lie derivative between [s+] and [s−] according to $\left\{\dot{x} = [u \quad k_s]^T\right\}$:

- $\mathcal{M}$: $s(x) = -x_2 = -k$, $x = [v \quad -k]^T$, $|v| < \epsilon$
- $\nabla s = [\pm 0 \quad -1]^T$
- $\mathscr{L}_{\kappa_-} s|_{\mathcal{M}} = -k_s$ always negative

9) Apply Transform 4.

$$\langle \mathrm{S}\rangle \rightarrow \langle \mathrm{S}_1\rangle|\langle \mathrm{S}_2\rangle \tag{60}$$
$$\langle \mathrm{S}_1\rangle \rightarrow [\mathrm{s}+]\left\{\dot{x} = [u \quad k_s]^T\right\} \cancel{\langle \mathrm{S}_3\rangle}\langle \mathrm{M}\rangle \tag{61}$$
$$\langle \mathrm{S}_2\rangle \rightarrow [\mathrm{s}-]\left\{\dot{x} = [-x \quad k_s]^T\right\} \cancel{\langle \mathrm{S}_4\rangle}\langle \mathrm{S}_1\rangle \tag{62}$$

10) Combine $\langle \mathrm{S}\rangle$ and $\langle \mathrm{M}\rangle$.

$$\langle \mathrm{S}\rangle \rightarrow \langle \mathrm{S}_1\rangle|\langle \mathrm{S}_2\rangle \tag{63}$$
$$\langle \mathrm{S}_1\rangle \rightarrow [\mathrm{s}+]\left\{\dot{x} = [u \quad k_s]^T\right\}\langle \mathrm{M}\rangle \tag{64}$$
$$\langle \mathrm{S}_2\rangle \rightarrow [\mathrm{s}-]\left\{\dot{x} = [-x \quad k_s]^T\right\}\langle \mathrm{S}_1\rangle \tag{65}$$
$$\langle \mathrm{M}\rangle \rightarrow \langle \mathrm{M}_1\rangle|\mathrm{M}_2 \tag{66}$$
$$\langle \mathrm{M}_1\rangle \rightarrow [\mathrm{m}+]\left\{\dot{x} = [u \quad -|u|]^T\right\}\langle \mathrm{M}_3\rangle' \tag{67}$$
$$\langle \mathrm{M}_2\rangle \rightarrow [\mathrm{m}-]\,\kappa_-\langle \mathrm{M}_4\rangle\ldots \tag{68}$$

11) Apply Transform 3.

$$\ldots\langle \mathrm{M}\rangle \rightarrow \langle \mathrm{M}_1\rangle\cancel{|\langle \mathrm{M}_2\rangle}\ldots \tag{69}$$

12) Remove unreferenced and singleton productions.

$$\begin{aligned}
\langle \mathrm{S}\rangle &\rightarrow \langle \mathrm{S}_1\rangle|\langle \mathrm{S}_2\rangle\\
\langle \mathrm{S}_1\rangle &\rightarrow [\mathrm{s}+]\left\{\dot{x} = [u \quad k_s]^T\right\}\langle \mathrm{M}_1\rangle\\
\langle \mathrm{S}_2\rangle &\rightarrow [\mathrm{s}-]\left\{\dot{x} = [-x \quad k_s]^T\right\}\langle \mathrm{S}_1\rangle\\
\langle \mathrm{M}_1\rangle &\rightarrow [\mathrm{m}+]\left\{\dot{x} = [u \quad -|u|]^T\right\}\langle \mathrm{M}_3'\rangle\\
\langle \mathrm{M}_3'\rangle &\rightarrow \langle \mathrm{M}_2'\rangle|\langle \mathrm{S}\rangle\\
\langle \mathrm{M}_2'\rangle &\rightarrow [\mathrm{m}-]\,\kappa_-\,[\mathrm{m}+]\,\kappa_-\langle \mathrm{S}\rangle
\end{aligned}$$

We have derived a grammar which guarantees the robot will never discharge. Note that within the production for $\langle \mathrm{M}\rangle$, we have produced a safe operating region as given in Sect. VII.

IX. Conclusion

This paper presented a method for analyzing complex hybrid systems and deriving controllers with provably correct performance. Complicated tasks require more powerful dynamic models than are possible with Regular Languages, so we apply a Context-Free Motion Grammar to describe the system. Safety-critical systems require guarantees on performance, hence we check our system model against a specification for correct behavior. To derive the controlled system which satisfies correctness, we have introduced a calculus of transformation rules which operate on hybrid dynamics. By progressively rewriting the grammar, the resulting grammar-based controller satisfies the specification. This process allows us to correctly control systems with complex hybrid dynamics.

There are many future possibilities for this work to automate the development of robust control systems. A software implementation in the style of Computer Algebra Systems or automated theorem provers would simplify the process of deriving the correct grammar used to control the system. This approach could be integrated with existing techniques for deriving continuous-domain controllers for broad classes of physical systems. We will develop tools based on this framework to facilitate the rapid development of safe controllers for complex robotic systems.

Acknowledgments

The authors thank Magnus Egerstedt for his key insights throughout the development of the Motion Grammar.

References

[1] N. Dantam, P. Kolhe, and M. Stilman, "The motion grammar for physical human-robot games," in *ICRA*. IEEE, 2011.
[2] N. Dantam and M. Stilman, "The motion grammar: Linguistic planning and control," in *RSS*. IEEE, 2011.
[3] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.
[4] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.
[5] G. Fainekos, A. Girard, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.
[6] N. Dantam, M. Egerstedt, and M. Stilman, "Make your robot talk correctly: Deriving models of hybrid system," in *RSS Workshop on Grounding Human-Robot Dialog for Spatial Tasks*. IEEE, 2011.
[7] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, 1979.
[8] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *Analysis and Optimization of Systems*, vol. 25, no. 1, pp. 206–230, January 1987.
[9] C. Cassandras, *Discrete-Event Systems*, 2nd ed. Springer, 2008.
[10] D. Hristu-Varsakelis and W. Levine, Eds., *Handbook of Networked and Embedded Control Systems*. Birkhauser, 2005.
[11] J. Lygeros, K. Johansson, S. Simic, J. Zhang, and S. Sastry, "Dynamical properties of hybrid automata," *IEEE Trans. on Automatic Control*, vol. 48, no. 1, pp. 2–17, 2003.
[12] T. Henzinger, "The theory of hybrid automata," in *Symposium on Logic in Computer Science*. IEEE, 1996, pp. 278–292.
[13] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Hybrid Systems*, pp. 209–229, 1993.
[14] R. Brockett, "Formal languages for motion description and map making," *Robotics*, vol. 41, pp. 181–191, 1990.
[15] D. Hristu-Varsakelis, M. Egerstedt, and P. Krishnaprasad, "On the structural complexity of the motion description language mdle," in *CDC*. IEEE, 2003, pp. 3360–3365.
[16] C. Baier, J. Katoen *et al.*, *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008.
[17] G. Holtzman, *The Spin Model Checker*. Addison Wesley, Boston, MA, 2004.
[18] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Trans. on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.
[19] J. Ding, J. Gillula, H. Huang, M. Vitus, W. Zhang, C. Tomlin, J. Gillula, G. Hoffmann, H. Huang, M. Vitus *et al.*, "Toward reachability-based controller design for hybrid systems in robotics," in *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, vol. 45, 2011, pp. 2526–2536.
[20] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," *HSCC*, pp. 271–274, 2004.
[21] E. Haghverdi, P. Tabuada, and G. Pappas, "Bisimulation relations for dynamical, control, and hybrid systems," *Theoretical Computer Science*, vol. 342, no. 2-3, pp. 229–261, 2005.
[22] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.