The Motion Grammar

Linguistic Perception, Planning, and Control

Neil Dantam
ntd@gatech.edu

Mike Stilman
mstilman@cc.gatech.edu

Center for Robotics and Intelligent Machines
Georgia Institute of Technology
801 Atlantic Dr., Atlanta, GA 30332

ABSTRACT

We present the Motion Grammar: a novel unified representation for task decomposition, perception, planning, and hybrid control that provides a computationally tractable way to control robots in uncertain environments with guarantees on completeness and correctness. The grammar represents a policy for the task which is parsed in real-time based on perceptual input. Branches of the syntax tree form the levels of a hierarchical decomposition, and the individual robot sensor readings are given by tokens. We implement this approach in the interactive game of Yamakuzushi on a physical robot resulting in a system that repeatably competes with a human opponent in sustained game-play for matches up to six minutes.

Categories and Subject Descriptors

F.4.2 [**Grammars and Other Rewriting Systems**]: Context Free; F.4.3 [**Formal Languages**]: Context Free; I.2.8 [**Problem Solving, Control Methods, Search**]: Plan execution, formation, and generation; I.2.9 [**Robotics**]: Manipulators

General Terms

Theory, Reliability

Keywords

Robotics, Manipulation, Formal Methods, Grammars, Control

1. INTRODUCTION

As robots come into increasing contact with humans, it is absolutely vital to *prove* that these potentially dangerous machines are *safe and reliable*. Furthermore, applying robots to increasingly complicated and varied tasks requires a tractable way to generate desired behaviors. Typical reactive strategies do not provide a way to prove how the system will respond during complicated tasks with uncertain outcomes [11, 10]. Existing deliberative planners often simplify the control system or have prohibitive computational cost [5]. By representing perception and control of robotic systems using Context-Free Grammars, our *Motion Grammar* enables robots to handle uncertainty in the outcomes of control actions through online parsing. Furthermore, the grammar makes it possible to prove that the system will respond to all possible situations.

Imagine a robot searching for earthquake survivors. This robot must carefully remove pieces of rubble while ensuring that the larger structure does not collapse. It must also coordinate its efforts with other robots and human rescuers. We consider a simplified version of this scenario in the two-player game of Yamakuzushi. While the game is adversarial, the robot and human collaborate to safely disassemble a pile of Shogi pieces. The game contains many of the challenging elements of physical human-robot interaction. For instance, both the robot and human make careful contact with pieces without causing them to fall. The robot slides pieces without losing contact. It also observes human actions and handles dangerous conditions. The key to all these interactions is that the robot must handle *uncertainty*. Our Motion Grammar guarantees that the robot responds to the complete range of uncertain outcomes online.

Our proposed linguistic method for computation in robotic systems yields three critical properties: hierarchical specification, immediate response to uncertainty and provable completeness. 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. Existing work has applied grammars to problems in perception [1, 2, 3, 4]. Our Motion Grammar seamlessly integrates this approach with robotic control. Finally, Context-Free grammars represent a balance between power, computational cost and provability. This allows the system designer to tackle a broader class of problems while simultaneously using practical computational resources and proving desired response.

This paper discusses related work, the theory behind the Motion Grammar, and presents our experimental validation using the game of Yamakuzushi. Sect. 2 explores related methods in planning and control, other uses of grammars related to robotics, and the use of Finite Automata in robotic systems. Sect. 3 formally defines the Motion Grammar and explain the requirements for applying context-free grammars to robotic systems. Sect. 5 describes our application of the Motion Grammar to the interactive game Yamakuzushi. Sect. 6 analyzes provability and properties of the Motion Grammar. Finally, Sect. 7 introduces some of the numerous possible extensions to the Motion Grammar approach.

2. RELATED WORK

Existing methods for planning and policy generation typically trade off efficiency for analytical properties such as completness. For instance, classical planners such as those based on STRIPS, Prolog decision makers, and modern PDDL symbolic reasoning methods [5] can guarantee completeness by reducing planning to theorem proving. However, the problem of inference in proposi-

tional and first order logic is NP-complete [5]. Furthermore, semantic methods do not explicitly address continuous domains. Partially Observable Markov Decision Processes can explicitly represent domains with uncertainty. However, POMDP solvers are also NP-complete [6]. The Motion Grammar provides a natural representation for hybrid continuous-discrete systems and by employing Context-Free languages, guarantees worst case $O(n^3)$ runtime [7].

There is a large body of literature on grammars from the Linguistic and Computer Science communities, with a number applications related to robotics focusing on image processing. Fu did some early work in syntactic pattern recognition [1]. Han uses attribute graph grammars to parse images of indoor scenes by describing the relationships of planes in the scene according to production rules [2]. Koutsourakis uses grammars for single view reconstruction by modelling the basic shapes in architectural styles and their relations using syntactic rules [3]. Toshev uses grammars to recognize buildings in 3D point clouds [4] by syntactically modelling the points as planes and volumes. B. Stilman's Linguistic Geometry applies a syntactic approach to deliberative planning and search in adversarial games [8]. These works show that grammars are useful beyond their traditional role in the Linguistic, Theoretical, and Programming Language communities. Our approach applies grammars to online control of robotic systems.

Some widely fielded efficient control architectures for robot manipulators are behavior-based methods introduced by Brooks [9] and Arkin [10]. Experimental validation of such methods poses a challenge for large correlated state spaces, such as physical human-robot manipulation, where a complete finite set of evaluations requires a discretization of state space that is exponential in the number of objects and robot DOF. Traditionally, the primary validation of behavioral problem decomposition has been experimental since it was thought that formal methods have unacceptable computational requirements [11]. Yet, recent work shows the feasibility and importance of formal analysis [12, 13, 14, 7]. We argue that the common applications of behavior-based robotics including military and personal service require formal verification. It is essential to *guarantee* that robots will not cause accidental injury or damage. [10] formalizes behavior semantics with schema theory, yet it does not ensure completeness. [15] proposes an Ethical Architecture to assist with correctness in the military domain. The Motion Grammar provides a more general solution to complete operation through domain-independent specification of robot response.

There exist a number of alternative formal methods for robot control. Control of Discrete Event Systems was pioneered by [16] and is detailed in [17]. [18] describes the Computation and Control Language (CCL), a type-safe domain specific language. However, because this language is Turing-complete, Rice's Theorem prevents proving nontrivial properties for arbitrary programs in the CCL [19]. [20] solves graph grammars as constraint satisfaction problems to direct many-agent systems in a planar environment. In contrast, our approach focuses on systems with fewer agents that have more correlated dimensions and uncertain action outcomes. Hybrid Automata [17] switch between continuous-domain controllers in a Finite State Machine. Our approach uses a Context-Free grammar. Stability of hybrid systems is analyzed by [**?**, **?**]. We provide alternative language-theoretic approaches to completeness and correctness in Sect. 6.1 and 6.2 allowing for more complicated performance specifications. [12, 13, 14] use linear temporal logic to formally describe uncertain multi-agent mobile robotics scenarios. Such research discretizes the two dimensional environment of mobile robots, requiring an exponential number of states in the degrees of freedom (DOF). Our manipulation tasks involve a 7-DOF robot interacting with 40 movable objects. Currently, discretization is not computationally feasible. The Motion Grammar avoids discretization through continuous domain semantics and discrete events that represent the task.

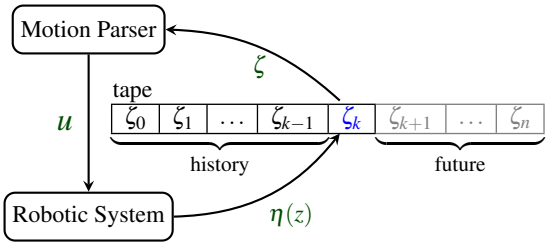


Figure 1: Operation of the Motion Parser.

The Motion Description Language (MDL) introduced by Brockett [21] is a hybrid control representation that describes robot operation by switching through a sequence of continuous-valued controllers. Applications of MDL require the strict assumption that each controller in the string will take the robot to a known state appropriate for the next controller. Real systems must contend with uncertainty in the outcomes of their actions, particularly with regard to discrete conditions and events. We address this challenge in the Motion Grammar through online parsing.

Numerous other authors have built upon the Motion Description Language. The Maneuver Automaton by Frazzoli [22] is able to control a simulated model helicopter, demonstrating the capability of this approach to handle systems with complicated dynamics; we proved a detailed comparison with the Motion Grammar in Sect. 6.4.2. Egerstedt uses MDL to describe plan complexity in terms of the length of plan string Σ [23], and provides another application with MDLp for the autonomous control of puppets [24]. These previous works, however, do not address the challenge of uncertainty in many physical systems, and the plan string Σ will be invalidated if the system falls into an unexpected state. The MDLe [25], works towards formalizing robot behaviors focusing on nonholonomic mobile robots, and allows a basic response to uncertainty through replanning. In contrast, our research extends the linguistic approach to manipulation and addresses online uncertainty by providing a complete *policy*.

When considering the Motion Grammar and MDL(e), it is important to observe the distinction between the language of *specification* and the language of *system*. The *specification language* is the set of strings used to generate the control program for the robot. Strings expressed in this language are parsed off-line prior to execution of the program. *The system language is the set of strings generated by the robot*. This language is parsed online by the control program. In Discrete Event System theory, the system language is described by the *generator* automaton [17]. Hristu-Varsakelis [26] provides an improved formalization of MDLe and proves that the MDL(e) *specification language* is Context-Free. We show that our Motion Grammar provides a more computationally powerful representation for the *system language* in Sect. 6.4.1.

3. THE MOTION GRAMMAR

The Motion Grammar represents the operation of a robotic system by a Context-Free language. The grammar is used to generate the *Motion Parser* which drives the robot as shown in Fig. 1.

Definition 1. The Motion Grammar, $\mathscr{G}_M$, is a tuple $\mathscr{G}_M = (\mathscr{Z}, Z, \mathscr{U}, \eta, V, P, K, S)$:

$\mathscr{Z}$ space of robot sensor readings

Z set of tokens representing robot state

$\mathscr{U}$ space of robot commands

$\eta : \mathscr{Z} \mapsto Z$ tokenizing function

V set of nonterminals

P set of productions

K set of semantic rules, each associated with one and only one production.

$S \in V$ starting nonterminal

Definition 2. The Motion Parser is a program that recognizes the language specified by the Motion Grammar and executes the corresponding semantic rules for each production.

Consider the illustration in Fig. 1. The output of the robot z is discretized into a stream of tokens ζ for the parser to read. Based on the sequence of tokens seen so far, the parser decides upon a control action u to send to the robot. The token type ζ is used to pick the correct production to expand at that particular step, and the semantic rule for that production uses the continuous value z to generate the command u. Thus, the Motion Grammar represents the language of robot sensor readings. The Motion Parser for that grammar is an *transducer* that translates the language of sensor readings into the language of controllers or actuator commands.

3.1 Motion Grammar Semantics

Semantics in the Motion Grammar are defined using attributes, which are parameters associated with each token and nonterminal. The attributes of tokens are continuous-valued variables such as sensor readings. In some production $A \to X_0 \ldots X_n$, the associated semantic rule $k \in K$ calculates the values to assign to the attributes of A and each X_i. These calculated values are functions of other attributes in the production, and there are certain restrictions on the allowable dependencies between attributes as detailed in Sect. 6.3.2. Ultimately, the parser reaches a semantic rule to calculate the robot's input $u \in \mathscr{U}$, and sends this value to the robot.

3.2 Languages, Systems, and Specifications

The robot and the controller are both mappings between the command and sensor spaces. They are *transducers* [19, p124].

Definition 3. The *robot* is a transducer f mapping from input command $u \in \mathscr{U}$ to sensor reading $z \in \mathscr{Z}$ given internal state (q_f, x_f) where q_f is a discrete valued vector and $x_f \in \Re^m$.

Definition 4. The *controller* is a transducer g mapping from sensor reading $z \in \mathscr{Z}$ to command $u \in \mathscr{U}$ with internal state (q_g, x_g) where q_g is a discrete valued vector and $x_g \in \Re^n$.

The Motion Grammar represents a language that is produced by the robot and consumed by the controller. The physical robotic system, coupled with the tokenizer η produces a string of tokenized sensor readings. The Motion Parser is the controller which uses semantic rules, K, to determine continuous commands. Both discrete and continuous information are passed between the robot and the controller. Discrete events are generated by η, becoming part of the string. The continuous portion of controller state x_g comes from sensor readings $z \in \mathscr{Z}$ or semantic rules K and is stored by the parser in the attributes of tokens and nonterminals. In general, the robot and controller may be of any language class. Since we use a Context-Free language over Z in the Motion Grammar, q_g can have infinite dimension due to the stack of a pushdown-automaton.

In linguistic control approaches, a critical distinction must be made between the language of the *system* and the language of the *specification*. The system is the physical entity with which we are concerned: the controller and the robot. The specification is the description of how the controller and robot respond; it is a set of mathematical symbols on paper or in a computer program. Both the system and the specification can be described by formal languages.

Definition 5. The *System Language*, L_g, is the set of strings generated by the robot and parsed by the controller during operation.

Definition 6. The *Specification Language*, L_s, is the set of strings that may describe the operation of the controller and robot.

These languages are related. The specification language is parsed offline to generate the control program. The system language is parsed online *by the control program*. The Motion Grammar is a specification language that describes a Context-Free *system*.

We emphasize that the Motion Grammar is not a Domain Specific Language or Robot Programming Language [27, p339] but rather the direct application of linguistic theory to robot control. The language described by the Motion Grammar is that of the robotic system itself. Our notation for this grammar, as presented in the figures, is a Syntax-Directed Definition (SDD) in Backus-Naur Form [28, p304]. Nonterminals are represented between angle brackets $\langle\rangle$, and tokens are represented between floor brackets $\lfloor\rfloor$.

4. GRAMMARS FOR ROBOTIC SYSTEMS

A Motion Grammar for any given task is developed based on the task specification and the robot hardware to be used. The spaces $\mathscr{U}$ and $\mathscr{Z}$ are the inputs and sensors that the robot possesses. The token set Z should be designed as the collection of events, timeouts, and discrete state that may occur during task execution. The system designer must create the tokenizing function η to map from $\mathscr{Z}$ to Z. Then, the nonterminals V and productions P can be created by hierarchically decomposing the task into progressively simpler subtasks until finally bottoming out in a continuously valued control-loop. Once the productions have been created, the semantic rules K for each production can be developed. These rules will perform some calculations on the attributes of the tokens and nonterminals in the production until the bottom of the control loops where the calculated command is sent to the robot. Finally, the starting nonterminal S is selected from V as the top level of the hierarchical decomposition and the grammar is complete.

An example Motion Grammar for an arbitrary task is given in Fig. 2. In this grammar, the overall task $\langle\text{Start}\rangle$ may be achieved either by consecutively performing $\langle\text{Task}_1\rangle$ and $\langle\text{Task}_2\rangle$ or by performing $\langle\text{Task}_3\rangle$. $\langle\text{Task}_1\rangle$ requires $\lfloor\text{event}_1\rfloor$ to be valid and then it goes to the control loop nonterminal $\langle g_1\rangle$. The control loop nonterminal $\langle g_1\rangle$ can expand to $(\lfloor 1\rfloor)$ indicating that the terminating condition for the loop has been achieved, or it can expand to $(\lfloor 0\rfloor \langle\kappa_1\rangle \langle g_1\rangle)$ which represents a single iteration of the loop. The token $\lfloor 0\rfloor$ indicates to the parser that it must continue the loop. The nonterminal $\langle\kappa_1\rangle$ is used for its semantic rule which will calculate and send the appropriate command to the robot. Finally, the nonterminal $\langle g_1\rangle$ appears causing the production to recurse on itself. Once $\langle\text{Task}_1\rangle$ is complete, the parser can move on to expanding $\langle\text{Task}_2\rangle$ which requires either $\lfloor\text{event}_2\rfloor$ to be valid, or it will execute $\langle\kappa_2\rangle$. Alternatively to beginning with $\langle\text{Task}_1\rangle$, the parser may expand $\langle\text{Task}_3\rangle$ if $\lfloor\text{event}_3\rfloor$ is initially valid. In this case, it will continue on to expand $\langle g_3\rangle$ instead of $\langle g_1\rangle$. Either way, the starting nonterminal $\langle\text{Start}\rangle$ will be satisfied and the overall task achieved.

4.1 Tokenizing

While the token set Z may be regarded as opaque symbols in some alphabet, it is more convenient to view the token as a tuple of discrete variables $(z_1, \ldots, z_n)$ and Z as the Cartesian product over the tuple elements, $z_1 \times \ldots \times z_n$. Each of these variables z_i would represents the discretization of a particular sensor on the robot. This is useful because a given production may only be related to a subset of the total sensors. For example, the force sensor and encoders may be the only relevant sensors when moving an arm

$$
\begin{aligned}
\langle \text{Start} \rangle &\rightarrow \langle \text{Task}_1 \rangle \langle \text{Task}_2 \rangle \mid \langle \text{Task}_3 \rangle \\
\langle \text{Task}_1 \rangle &\rightarrow \lfloor \text{event}_1 \rfloor \langle g_1 \rangle \\
\langle \text{Task}_2 \rangle &\rightarrow \lfloor \text{event}_2 \rfloor \mid \langle \kappa_2 \rangle \\
\langle \text{Task}_3 \rangle &\rightarrow \lfloor \text{event}_3 \rfloor \langle g_3 \rangle \\
\langle g_1 \rangle &\rightarrow \lfloor 1 \rfloor \mid \lfloor 0 \rfloor \langle \kappa_1 \rangle \langle g_1 \rangle \\
\langle g_3 \rangle &\rightarrow \lfloor 1 \rfloor \mid \lfloor 0 \rfloor \langle \kappa_3 \rangle \langle g_3 \rangle
\end{aligned}
$$

Figure 2: Sample generic Motion Grammar.

```
1: procedure A
2:     Choose a production for A, A → X_1 ... X_n
3:     for i = 1, n do
4:         if nonterminal? X_i then call X_i
5:         else if X_i = η(z(t)) then continue
6:         else error
7:         end if
8:     end for
9:     Execute semantic rule for A → X_1 ... X_n
10: end procedure
```

Figure 3: Procedure for $\langle \text{A} \rangle$ in a recursive parser for $\mathscr{G}_M$

to contact an object; the camera is not used for the contact motion. The parser is thus able to expand upon a given production for any value of non-relevant variables. Besides making the productions easier to express, this division is also efficient because it eliminates the need to process data from sensors that are not needed to make the current decision. Discretizing robot sensors into a tuple of discrete values provides an efficient set of tokens for the parser.

4.2 Parsing

Once the Motion Grammar for the task is developed, it must be transformed into the Motion Parser. For our proof-of-concept application, we used a hand-written *recursive descent parser*, an approach also employed by GCC [29]. A recursive descent parser is written as a set of mutually-recursive procedures, one for each nonterminal in the grammar. An example of one of these procedures is shown in Fig. 3, based on [28, p219]. Each procedure will fully expand its nonterminals via a top-down, left-to-right derivation. This approach is a good match for the Motion Grammar's top-down task decomposition and its left-to-right temporal progression.

5. HUMAN-ROBOT GAME APPLICATION

We implemented and evaluated the performance of the Motion Grammar on the Japanese game Yamakuzushi [1] (yama). This game is similar to Jenga. In yama, a mountain of Shogi pieces is randomly piled in the middle of a table as shown in Fig. 4. Each of the two players tries to clear the pieces from the table. Each player is only allowed to use one finger to move pieces. If the player causes the pieces to make a sound, it becomes the other players turn. The winner is the player who removes the most pieces.

In our implementation, a human plays against the Schunk LWA3 7-DOF robot arm. This robot has a 6-axis force-torque sensor at the wrist that we used for force control. A Mesa Swiss Ranger allowed to robot to locate the shogi pieces, and a microphone allowed it to detect sounds indicating turn loss. We used a Kalman filter on the force-torque sensor and both a median filter and a Kalman filter on the Swiss Ranger. The robot used a speaker and text-to-speech program to communicate with its human opponent. The lowest levels of our grammatical controller operated at 1kHz.

[1] Videos online at `http://golems.org/node/1224`

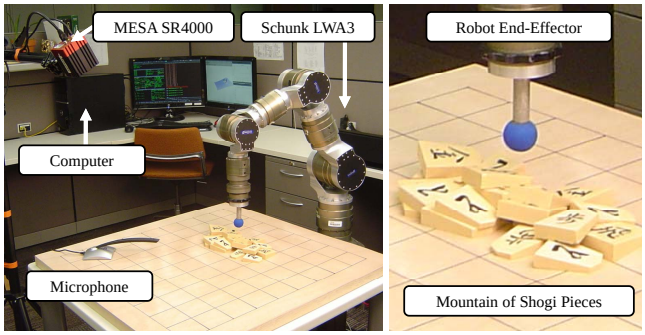


Figure 4: Our experimental environment for linguistic physical human-robot games of Yamakuzushi.

Token	Description
$\lfloor \alpha \leq \text{t} < \beta \rfloor$	Within time Range
$\lfloor$contact$\rfloor$	E.E. touching piece
$\lfloor$no contact$\rfloor$	not touching piece
$\lfloor$destination$\rfloor$	at traj. end
$\lfloor$human piece$\rfloor$	removed by human
$\lfloor$robot piece$\rfloor$	removed by robot
$\lfloor$clear$\rfloor$	board cleared
$\lfloor$sound$\rfloor$	noise removing piece
$\lfloor$quiet$\rfloor$	no noise made
$\lfloor$in space$\rfloor$	human in workspace
$\lfloor \neg$in space$\rfloor$	not in workspace
$\lfloor$point$\rfloor$	element of point cloud

(a) Tokens

Attr.	Description
Sensor Driven	
t	Current Time
x	Act. Robot/Point Pos.
f	Act. E.E. Force
Inherited/Synthesized	
t_α	Duration or Timeout
x_r	Ref. Robot Pos.
$\dot{x}_r$	Ref. Robot Vel.
x_0	Traj. Start Pos.
x_n	Traj. End Pos.
f_r	Ref. E.E. Force

(b) Attributes

Figure 5: Tokens, Attributes for the Yama Motion Grammar

5.1 Tokens and Attributes

The tokens and attributes used by the yama grammar encode the hybrid system dynamics. Both are summarized in Fig. 5. The tokenizer η produces each token ζ by thresholding the current sensor reading. The Motion Parser uses tokens to determine syntactically correct expansions of productions according to the grammar.

While tokens encode discrete events, attributes represent the continuous system state. Some attributes are obtained directly from the sensors: x from encoders, f for the force-torque sensor, t from the clock. The rest are derived according to semantic rules. Attributes are passed between the tokens and nonterminals in the grammar to implement the continuous domain semantics. Hybrid control is achieved by combining the discrete decisions of the Motion Parser with continuous functions defined by the semantics.

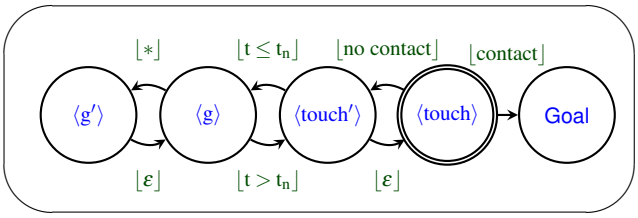
5.2 Semantic Rules

Semantic rules are procedures that are executed when the parser expands a production. In yama, these rules assign updated sensor readings to attributes, maintain previously computed attributes, determine new targets for the controller and send control commands. Attribute maintenance is achieved through *synthesis* in child productions and *inheritance* from the parent production being expanded and from the left-siblings of each nonterminal. In this paper, we give one key example of robot control through semantic rules.

The SDD presented in Fig. 6 illustrates the semantics of trapezoidal velocity profiles. Expanding $\langle \text{g}' \rangle$ yields distinct semantic rules depending on t, the time attribute of the current nonterminal. For each stage of the trajectory we target distinct reference positions and velocities. The semantic rule for $\langle \text{g} \rangle$ defines the control output as a target velocity, u, based on the references provided by expanding $\langle \text{g}' \rangle$ and the current force attribute. This demonstrates how the continuous domain control of physical systems can be encoded in the semantics of a discrete grammar.

Production			Semantic Rules
$\langle g \rangle$	$\rightarrow$	$\langle g' \rangle$	$u = \dot{x}_r - K_p(x - x_r) - K_f(f - f_r)$
$\langle g' \rangle$	$\rightarrow$	$\lfloor t < 0 \rfloor$	$x_r = 0, \dot{x}_r = 0$
$\langle g' \rangle$	$\rightarrow$	$\lfloor 0 \leq t < t_1 \rfloor$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t^2, \dot{x}_r = t\ddot{x}_m$
$\langle g' \rangle$	$\rightarrow$	$\lfloor t_1 \leq t < t_2 \rfloor$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t_1^2 + \dot{x}_m(t - t_1), \dot{x}_r = \dot{x}_m$
$\langle g' \rangle$	$\rightarrow$	$\lfloor t_2 \leq t < t_n \rfloor$	$x_r = x_n - \frac{1}{2}\ddot{x}_m(t_n - t)^2, \dot{x}_r = \dot{x}_m + \ddot{x}_m(t_2 - t)$
$\langle g' \rangle$	$\rightarrow$	$\lfloor t_n < t \rfloor$	$x_r = 0, \dot{x}_r = 0$

Figure 6: Syntax-Directed Definition that encodes impedance control over trapezoidal velocity profiles.



(a) Finite State Machine Representation

$$
\begin{aligned}
\langle \text{touch} \rangle &\rightarrow \lfloor \text{contact} \rfloor \\
&\mid \lfloor \text{no contact} \rfloor \langle \text{touch}' \rangle \langle \text{touch} \rangle \\
\langle \text{touch}' \rangle &\rightarrow \lfloor t > t_n \rfloor \mid \lfloor t \leq t_n \rfloor \langle g \rangle \langle \text{touch}' \rangle
\end{aligned}
$$

(b) Equivalent Motion Grammar Fragment

Figure 7: Illustration of control for piece touching.

5.3 Touching Pieces

The Finite State Machine in Fig. 7(a) could be used to make the robot touch a shogi piece. This state machine is equivalent to the right regular grammar in Fig. 7(b). In the grammar, the tokenizing function η applies a threshold to the force-torque sensors and produces $\lfloor \text{contact} \rfloor$ if the end-effector forces exceed the threshold or $\lfloor \text{no contact} \rfloor$ otherwise. To expand the $\langle \text{touch} \rangle$ nonterminal, the parser consumes a $\lfloor \text{contact} \rfloor$ and returns, or it consumes a $\lfloor \text{no contact} \rfloor$, moves down a small increment using the trapezoidal velocity profile in $\langle \text{touch}' \rangle$ and $\langle g \rangle$, and recurses on $\langle \text{touch} \rangle$. This behavior is mirrored by the state transitions in Fig. 7(a).

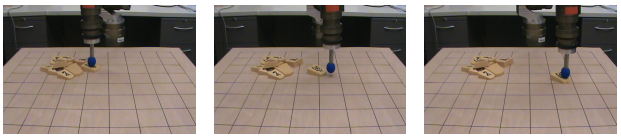
We implemented this grammatical controller for touching shogi pieces on the LWA3 and compared it to a pure continuous-domain impedance controller. Due to the large physical constants of the LWA3, we implemented our impedance controller on top of a velocity controller. This approach has the potential for oscillation, especially when gains are large, yet even under these circumstances, the grammatical controller achieved superior performance. The impedance controller in Fig. 8(a) with an appropriate gain is able to make contact with the piece, but it does suffer from some oscillation and overshoot. An impedance controller with high gains in Fig. 8(b) has severe oscillation and very poor performance. The grammatical controller in Fig. 8(c) has both less overshoot and less oscillation than the purely continuous impedance controller. Additionally, we also observed the grammatical controller to be much more robust to sensing errors. If we estimated the height of a piece incorrectly, the impedance controller would often completely fail to make contact due to limited ability to increase gains; however, the grammatical controller would still be able to find the piece.

5.4 Sliding and Reacquiring Lost Pieces

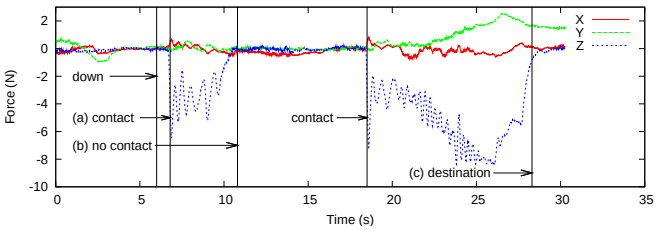
The grammar in Fig. 9 describes how the robot slides pieces and how it can reacquire pieces it has lost. This grammar again uses the trapezoidal velocity profile $\langle g \rangle$ and $\lfloor \text{contact} \rfloor$ and $\lfloor \text{no contact} \rfloor$ is for the grammar in Fig. 7(b). The tokenizer η supplies $\lfloor \text{destination} \rfloor$

$$
\begin{aligned}
\langle \text{slide} \rangle &\rightarrow \lfloor \text{contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
&\mid \lfloor \text{no contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
&\mid \lfloor \text{destination} \rfloor \\
&\mid \langle \text{reacquire} \rangle \lfloor \text{contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
\langle \text{reacquire} \rangle &\rightarrow \lfloor \text{no contact} \rfloor \langle \text{touch} \rangle
\end{aligned}
$$

Figure 9: Grammar fragment to reacquire lost pieces.



(a) contact (b) no contact (c) destination



(d) Forces at the robot end-effector during grammar execution.

Figure 10: Images and data gathered by applying the sliding grammar in Fig. 9 when contact is lost.

when robot has moved the piece to the desired location. If the robot momentarily loses contact with the piece, it will continue expanding $\langle \text{slide} \rangle$; however, when the contact loss is long enough for the robot to move past the piece, it will have to backtrack to the last contact position to reacquire it. This action is performed by $\langle \text{reacquire} \rangle$. Following this grammar allows the robot to move pieces across the table and recover any pieces that it loses.

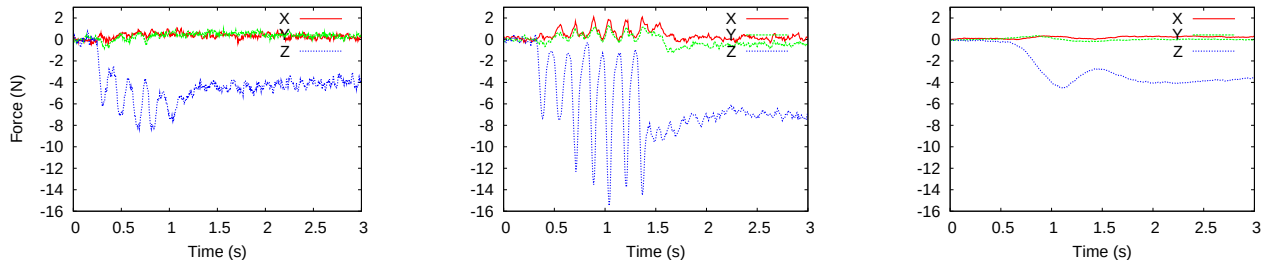
Our implementation of this grammar on the LWA3 slides pieces and recovers them after any contact loss. As the robot moves through the sequence in Fig. 10, it uses the end-effector forces shown in Fig. 10(d) to make its decisions regarding piece contact. At 6s, the robot begins moving down to touch the piece. It acquires the pieces at 6.8s, Fig. 10(a) and begins sliding. At 10.8s, Fig. 10(b), it loses contact with the piece. Recognizing this, the robot backtracks, and again makes contact with the piece at 18.5s. It then continues sliding the piece, reaching the destination at 28.3s, Fig. 10(c).

5.5 Selecting Target Pieces

The grammar in Fig. 5.5 describes how the robot chooses a target piece to move. The Swiss Ranger provides a point cloud from the stack of pieces, Fig. 11(a). From this point cloud, a set of planes is progressively built based on the distance between the test point and the plane and on the angle between the plane normal and the normal of a plane in the region of the test point. Each of these planes is a potential target for the robot. The precedence of these planes is based on height above the ground, a clear path to the edge of the table, and whether the piece may be supporting stacked neighboring pieces. The parser will select the highest precedence plane as the target to move, Fig. 11(b).

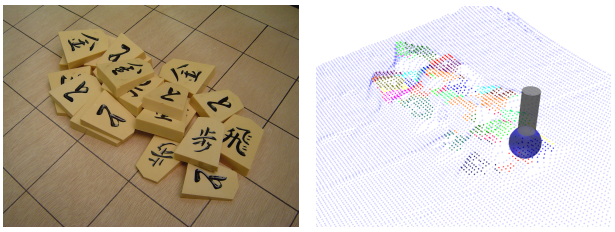
5.6 Deciding the Winner

An example of a Context-Free system language is deciding the winner of the game. The grammar fragment for this task is shown in Fig. 12. This grammar will count the number of pieces removed by



(a) Impedance (b) High Gain Impedance (c) Grammatical

Figure 8: Piece Touching with Impedance and Discrete Control Strategies



(a) Pieces (b) Planes

$$
\begin{aligned}
\langle \text{act} \rangle &\rightarrow \langle \text{target} \rangle \langle \text{touch} \rangle \langle \text{slide} \rangle \\
\langle \text{target} \rangle &\rightarrow \langle \text{plane} \rangle_0 \mid \ldots \mid \langle \text{plane} \rangle_n \\
\langle \text{plane}_i \rangle &\rightarrow \lfloor \text{point} \rfloor_j \mid \lfloor \text{point} \rfloor_j \langle \text{plane}_i \rangle
\end{aligned}
$$

(c) Motion Grammar Fragment

Figure 11: Deciding target piece and direction.

$$
\begin{aligned}
\langle \text{winner} \rangle &\rightarrow \langle \text{draw} \rangle \mid \langle \text{robot} \rangle \mid \langle \text{human} \rangle \\
\langle \text{draw} \rangle &\rightarrow \varepsilon \\
&\mid \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \\
&\mid \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \\
\langle \text{robot} \rangle &\rightarrow \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \\
&\mid \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{robot} \rangle \\
\langle \text{human} \rangle &\rightarrow \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \\
&\mid \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{human} \rangle
\end{aligned}
$$

Figure 12: Grammar fragment to decide winner

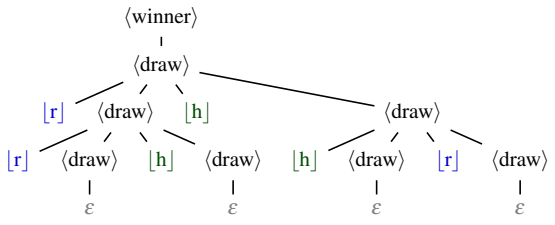


Figure 13: Parse tree for winner decision problem, $\lfloor \text{h} \rfloor \equiv \lfloor \text{human piece} \rfloor$, $\lfloor \text{r} \rfloor \equiv \lfloor \text{robot piece} \rfloor$

$$
\begin{aligned}
\langle \text{game} \rangle &\rightarrow \langle \text{robot turn} \rangle \lfloor \text{clear} \rfloor \langle \text{winner} \rangle \\
&\mid \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \lfloor \text{clear} \rfloor \langle \text{winner} \rangle \\
&\mid \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \langle \text{game} \rangle \\
\langle \text{robot turn} \rangle &\rightarrow \langle \text{act} \rangle \lfloor \text{quiet} \rfloor \langle \text{robot turn} \rangle \\
&\mid \langle \text{act} \rangle \lfloor \text{sound} \rfloor \\
&\mid \lfloor \text{clear} \rfloor \\
\langle \text{human turn} \rangle &\rightarrow \langle \text{wait sound} \rangle \langle \text{wait safe} \rangle \\
\langle \text{wait sound} \rangle &\rightarrow \lfloor \text{sound} \rfloor \\
&\mid \lfloor \text{clear} \rfloor \\
&\mid \lfloor \text{quiet} \rfloor \langle \text{wait sound} \rangle \\
\langle \text{wait safe} \rangle &\rightarrow \lfloor \text{in space} \rfloor \langle \text{wait safe} \rangle \\
&\mid \lfloor \neg \text{in space} \rfloor
\end{aligned}
$$

Figure 14: Complete Yama Grammar. This is the remaining set of productions used in the game.

the human and the robot. The ⟨draw⟩ nonterminal serves to match up a piece removed by the human and a piece removed by the robot. The ⟨robot⟩ and ⟨human⟩ nonterminals consume the extra tokens for pieces removed by the robot or the human, indicating that player is the winner. An example parse tree for a draw condition is given in Fig. 13. This parse tree demonstrates how each ⟨draw⟩ matches one ⌊r⌋ and one ⌊h⌋ token which requires a Context-Free language [19, p125]. This solution to counting problem for deciding the winner demonstrates the advantage of using a Context-Free model for the Motion Grammar.

5.7 Complete Game

The remaining productions to implement a full game of Yamakuzushi are given in Fig. 14. A game consists of alternating robot and human turns until the board is clear. During the ⟨robot turn⟩, it will repeatedly ⟨act⟩ to remove pieces until it causes ⌊sound⌋ by making a noise exceeding the preset threshold or until it clears the board. During the ⟨human turn⟩, the robot will simply wait until it detects a ⌊sound⌋ or sees that the board has been cleared. After the human makes a ⌊sound⌋, the robot will wait until the human is out of the workspace before it begins its turn.

6. ANALYSIS

In this section, we investigate several analytical properties of the Motion Grammar. Sect. 6.1 demonstrates grammar Completeness ensuring that the robot will respond to all situations. Sect. 6.2 presents one approach to proving Correctness, ensuring that the grammatically-controlled system satisfies a target constraint. Next, we explain the necessary properties of the Motion Grammar that arise from the dynamic, temporal constraints of online pars-

ing as opposed to the static inputs given to traditional compilers. Finally, Sect. 6.4 compares the Motion Grammar and two related approaches, MDLe and the Maneuver Automaton.

One key element of our analysis is that Motion Grammars use a Context-Free System Model. This representation allows for an appropriate balance between power of the computational model and provability of the resulting system. Regular languages are a simpler representation whose response can be just as easily proven, but they are very limited in what they can represent. Context-Sensitive Languages are somewhat more powerful than Context-Free, but the Context-Sensitive decision problem is PSPACE-Complete. Recursively-enumerable languages are more powerful, but by Rice's Theorem, any nontrivial property of a Turing Machine is unprovable [19, p188]. Since Context-Free languages maintain provable properties and can be parsed in polynomial time, they are an appropriate representation for robotic systems that must operate in real-time and whose behavior should be provably complete.

6.1 Completeness

By formalizing the hybrid control problem as a grammar that recognizes the language of the robotic system, we can precisely determine what situations we are able to handle. This allows us to guarantee that the robot will *never give up*.

Definition 7. A Motion Grammar $\mathscr{G}_M$ is *complete*, $\mathsf{complete}\{\mathscr{G}_M\}$, if and only if it will direct the robot to take some action in all possible circumstances.

PROPOSITION 1. *Let $L_{\mathscr{G}_M}$ be the set of all strings accepted by $\mathscr{G}_M$. Let L_g be the set of all strings produced by the robotic system. Then* $\mathsf{complete}\{\mathscr{G}_M\} \iff L_g \subseteq L_{\mathscr{G}_M}$

PROOF. Suppose that $\mathsf{complete}\{\mathscr{G}_M\} \wedge \exists \ell \in L_g,\ \ell \notin L_{\mathscr{G}_M}$. Because $\ell \notin L_{\mathscr{G}_M}$, the grammar will give a syntax error if it encounters ℓ. If the robot produces ℓ, the grammar could not direct the robot and thus $\neg\mathsf{complete}\{\mathscr{G}_M\}$. This is a contradiction. Now suppose that $\neg\mathsf{complete}\{\mathscr{G}_M\} \wedge \forall \ell \in L_g, \ell \in L_{\mathscr{G}_M}$. Since $\mathscr{G}_M$ accepts every $\ell \in L_g$, there is no possibility that the grammar couldn't handle. So $\mathsf{complete}\{\mathscr{G}_M\}$. This is a contradiction. □

Now we give one method for proving completeness of a Motion Grammar. Though the general subset relation between two Context-Free languages is undecidable [19, p203], we can take advantage of the constraints imposed by continuous-domain dynamics to decide $\mathsf{complete}\{\mathscr{G}_M\}$. This approach combines the First and Follow sets used in compiler theory [28, p220] and the Invariant Set Theorem from Lyapunov theory [30, p68] to show that the system will always remain in a region that is defined by the grammar. For this purpose, we use a variation on the conventional $\mathsf{follow}(A)$ which is based on unions and instead base our $\mathsf{xfollow}(A)$ on the intersections of tokens that may follow A.

Definition 8. $\mathsf{xfollow}(A)$ is the intersection of all tokens that may follow nonterminal A in the grammar. Specifically, $\mathsf{xfollow}(S)$ = \$ where \$ indicates the end of the input string and S is not a child in any production. For all $A \neq S$, $\mathsf{xfollow}(A)$ is the intersection of $\mathsf{first}(\beta)$ for all productions $B \to \alpha A \beta$ and of $\mathsf{xfollow}(B)$ for all productions $B \to \alpha A$.

LEMMA 2. *If a top-down parser recieves token ζ after expanding some production $A \to X_1 \ldots X_n$ and $\zeta \in \mathsf{xfollow}(A)$, then ζ will not generate a syntax error.*

PROOF. Since $\zeta \in \mathsf{xfollow}(A)$, all parent productions of A must be able to accept ζ following their expansion of A. Thus ζ cannot cause a syntax error. □

THEOREM 3. *For Motion Grammar $\mathscr{G}_M$ in Chomsky normal form with productions $p = (A \to \chi) \in P$, let*

- *S be the starting nonterminal*
- *$\mathscr{X} = \Re^n$ be the continuous, finite-dimensional, state of the robot and $x \in \mathscr{X}$*
- *$h_p(x) : \mathscr{X} \mapsto \mathscr{Z}$*
- *$\mathscr{R}_0(p) = \{x \mid \eta(h_p(x)) \in \mathsf{first}(\chi)\}$*
- *$\mathscr{R}_1(p) = \{x \mid \eta(h_p(x)) \in \mathsf{xfollow}(A)\}$*
- *ζ_0 be the first token received by the Motion Parser.*

If, $\zeta_0 \in \mathsf{first}(S)$ and for all $p \in P$, $\mathscr{R}_0(p) \subseteq \mathscr{R}_1(p)$ and $\mathscr{R}_0(p)$ or $\mathscr{R}_1(p)$ is an Invariant Set, then $\mathsf{complete}\{\mathscr{G}_M\}$.

PROOF. We prove this theorem inductively. For the inductive case, the Motion Parser expanding some $p \in P$ must start with $x \in \mathscr{R}_0(p)$. If $\mathscr{R}_0(p) \subseteq \mathscr{R}_1(p)$ and $\mathscr{R}_0(p)$ or $\mathscr{R}_1(p)$ is an Invariant Set, then at the end of expanding p, the system will have $x \in \mathscr{R}_1(p)$. This will generate a subsequent token $\zeta \in \mathsf{xfollow}(A)$ and Lemma 2 shows that this will not cause a syntax error. For the base case, the Motion Parser is given ζ_0 as the first token. Since ζ_0 is in $\mathsf{first}(S)$, it will not generate a syntax error. □

Theorem 3 shows us that a system is fully represented by a Motion Grammar. If our Motion Grammar is not complete, we can fix this by modifying semantic rules K to create the invariant set property or by creating more productions in P to respond to the additional cases. When the grammar is complete, it guarantees a response to all situations and additionally lets us use the discrete *syntax* to represent the system. We use the syntax to guarantee that the system is correct.

6.2 Correctness

In addition to ensuring that the robot takes *some* action for all circumstances, it is also important to evaluate the correctness of the action. We define the correctness of a language specified by the Motion Grammar, $L_{\mathscr{G}_M}$, by relating it to a *constraint language*, L_r. While $L_{\mathscr{G}_M}$ for a given problem integrates all problem subtasks, as shown in Sect. 5, the constraint language targets correctness with respect to a specific criterion. Criteria can be formulated for general tasks including: safe operation, target acquisition and the maintenance of desirable system attributes. By judiciously choosing the complexity of these languages, we can evaluate whether or not all strings generated during execution are also part of language L_r.

Definition 9. A Motion Grammar $\mathscr{G}_M$ is correct with respect to some constraint language L_r when all strings in the language of $\mathscr{G}_M$ are also in L_r: $\mathsf{correct}\{\mathscr{G}_M, L_r\} \iff L_{\mathscr{G}_M} \subseteq L_r$.

The question of $\mathsf{correct}\{\mathscr{G}_M, L_r\}$ is only decidable for certain language classes of $L_{\mathscr{G}_M}$ and L_r. Hence, the formal guarantee on correctness is restricted to a limited range of complexity for both systems and constraints. We prove decidability and undecidability for combinations of Regular, Deterministic Context-Free, and Context-Free Languages.

LEMMA 4. *Let $\mathscr{L}_R$ be the Regular set, $\mathscr{L}_D$ be the Deterministic Context-Free set, and $\mathscr{L}_C$ be the Context-Free set. $R \in \mathscr{L}_R$, $D \in \mathscr{L}_D$, and $C \in \mathscr{L}_C$. Then,*

1. *$C \subseteq C'$ is undecidable. [19, p203]*
2. *$R \subseteq C$ is undecidable. [19, p203]*
3. *$C \subseteq R$ is decidable. [19, p204]*
4. *$R \subseteq D$ is decidable. [19, p246]*
5. *$D \subseteq D'$ is undecidable. [19, p247]*

COROLLARY 5. *Based on $\mathscr{L}_R \subset \mathscr{L}_D \subset \mathscr{L}_C$, the results from [19] extend to the following statements on decidability:*

1. *$D \subseteq R$ and $R \subseteq R$ are decidable.*
2. *$D \subseteq C$ undecidable.*
3. *$C \subseteq D$ is undecidable.*

Combining these facts about language classes, the system designer can determine which types of languages can be used to define both the grammars for specific problems and general constraints.

THEOREM 6. *The decidability of* $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ *for Regular, Deterministic Context-Free, and Context-Free Languages is specified by Fig. 15.*

	$L_r \in \mathscr{L}_R$	$L_r \in \mathscr{L}_D$	$L_r \in \mathscr{L}_C$
$L_{\mathscr{G}_M} \in \mathscr{L}_R$	yes	yes	no
$L_{\mathscr{G}_M} \in \mathscr{L}_D$	yes	no	no
$L_{\mathscr{G}_M} \in \mathscr{L}_C$	yes	no	no

Figure 15: Decidablity of $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ **by language class.**

PROOF. Each entry in Fig. 15 combines a result from Lemma 4 or Corollary 5 with Definition 9. The algorithms that perform each subset evaluation and therefore the evaluation of $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ are given in [19]. □

Theorem 6 ensures that we can prove the correctness of a Motion Grammar with regard to any constraint languages in the permitted classes. We are limited to Regular constraint languages except in the case of a Regular system language which allows a Deterministic Context-Free constraint. Regular constraint languages may be specified as Finite Automata, Regular Grammars, or Regular Expressions since all are equivalent. Furthermore, since we can determine the complement of any Regular or Deterministic Context-Free language, we can specify constraints in terms of events that should *never* happen, $L_{\mathscr{G}_M} \subseteq \bar{L}_e$. Both positive and negative constraints allow existing algorithms to guarantee that a Motion Grammar-based system is safe and reliable.

6.3 Time and Semantics

Next we study the properties of the Motion Grammar that arise from the online parsing of the system language. While a translating parser such as a compiler is typically given its input as a file, a Motion Parser must act token-by-token continually driving the system. This temporal constraint means the Motion Grammar must decide using only the single token of lookahead derived from the current sensor reading. It cannot lookahead to future tokens that have not happened, and it cannot backtrack to undo actions already taken. This restriction affects the type of parser we may use and the allowable ordering of attribute semantics.

6.3.1 Selecting Productions and Semantic Rules

First we compare the Motion Grammar to the LL(1) class of grammars. LL(1) grammars can be parsed by recursively descending through productions, picking the next production to expand using only a single token of lookahead and without backtracking [28, p222]. While we could satisfy the Motion Grammar's temporal constraint by restricting ourselves to an LL(1) grammar, we can actually relax that restriction slightly. The real problem is not that the Motion Parser must immediately know which production it is expanding, but that it must immediately provide some command to the robot. Thus the parser may use additional lookahead, but only if all productions it is deciding between have *identical semantic rules*. This way, the parser can immediately execute the semantic rule, and use some additional lookahead to figure which production it is really expanding. We describe this property as *Semantically LL(1)*. This is important because a pure LL(1) grammar must unambiguous; however, by requiring LL(1) only in semantics, we are also able to use certain ambiguous grammars.

Definition 10. A grammar is Semantically LL(1) if for all strings in its language, the correct semantic rule to execute can be determined using a single token of lookahead and without backtracking.

THEOREM 7. *A Motion Grammar must be Semantically LL(1).*

PROOF. The Motion Parser derived from the Motion Grammar, $\mathscr{G}_M$, must be able to immediately provide the system with an input command $u \in \mathscr{U}$ in response to each token, and it cannot change the value of commands already sent. Suppose that the $\mathscr{G}_M$ was not Semantically LL(1). This would mean it could use multiple tokens of lookahead or backtrack before deciding on a semantic rule to calculate u. Since u must be known before more tokens are accepted and previous u values cannot be changed, this a contradiction. Thus $\mathscr{G}_M$ must be Semantically LL(1). □

When designing our Motion Grammar, we must ensure that the correct semantic rule can be selected without any additional lookahead. For productions where this is not the case, we must either rework the grammar or instruct the parser as to the appropriate precedence levels so that it can resolve the ambiguity.

6.3.2 Attribute Inheritance and Synthesis

Now we consider the structure of the attribute semantics in the Motion Grammar. In our SDD, the attributes of some given nonterminal will be calculated from the attributes of other tokens and nonterminals; this introduces a dependency graph into the syntax tree. We must ensure that the dependency graph has no cycles or we will not be able to evaluate the SDD [28, p310]. The temporal nature of the Motion Grammar constrains the attribute dependencies even further; during parsing, we only have access to information from the past because the future has not happened yet. Attributes can be described as either *synthesized* or *inherited* based on their dependencies. Synthesized attributes depend on the children of the nonterminal while inherited attributes depend on the nonterminal's parent, siblings, and other attributes of the nonterminal itself. The temporal constraint of the Motion Grammar corresponds to a particular class of SDDs called *L-attributed definitions* for the left-to-right dependency chain. A nonterminal X in an L-attributed definitions may only have attributes that are synthesized, or inherited with dependencies on inherited attributes of X's parent, attributes of X's siblings that precede it in the production, or on X itself in ways that do not result in a cycle [28, p313].

THEOREM 8. *A Motion Grammar must have L-attributed semantics.*

PROOF. We must determine the attributes in a single pass because parsing is online, so the past cannot be changed, and the future is unknown. Let the inherited attributes of nonterminal V be $V.h$, and let its synthesized attributes be $V.s$. For all productions $p = A \to X_1 X_2 \ldots X_n$, consider the attributes of X_i. While expanding X_i, $A.h$ are known. All X_j, $j < i$ in this production have already been expanded because they represent past action, so $X_j.h$ and $X_j.s$ are also known. However, X_k, $k > i$ represent future actions, so $X_k.h$ and $X_k.s$ are unknown. This also means that $A.s$ is unknown because its value may depend on $X_k.h$ and $X_k.s$. Consequently, $X_i.h$ may only depend on $A.h$, $X_j.h$, and $X_j.s$. $X_i.s$ may depend on from its children because they will be known after X_i has been expanded. These constraints on attributes synthesis and inheritance correspond to L-attributed definitions. □

6.4 Relationship with Existing Methods

The Motion Grammar builds on a number of advances in linguistic control. This section relates our approach to two related methods: MDLe and Maneuver Automata.

procedure MDLE-TO-FA(Σ, B', U)
  for $i = 0, n$ **do**
    Create state $s_i \in S$
  end for
  for all $\xi_j \in B'$ **do**
    Create tokens $\lfloor \xi_j = 0 \rfloor \in E$ and $\lfloor \xi_j = 1 \rfloor \in E$
  end for
  for $i = 0, n$ **do**
    if $\sigma_i = \xi_j \in B'$ **then**
      Create transition $\left(s_i \xrightarrow{\xi_j = 0} s_k \right) \in d$,
      where σ_k is the $\lfloor (\rfloor$ matching the $\lfloor) \rfloor$ following σ_i
      for all $\sigma_p = \xi_r \in B'$ nested within $\lfloor (\rfloor_k$ and $\lfloor) \rfloor_{i+1}$ **do**
        Create transition $\left(s_i \xrightarrow{\xi_j = 1} s_{i+1} \right) \in d$
      end for
    else
      Create transition $\left(s_i \xrightarrow{\varepsilon} s_{i+1} \right) \in d$.
    end if
  end for
end procedure

Figure 16: Convert MDLe String to Finite Automaton

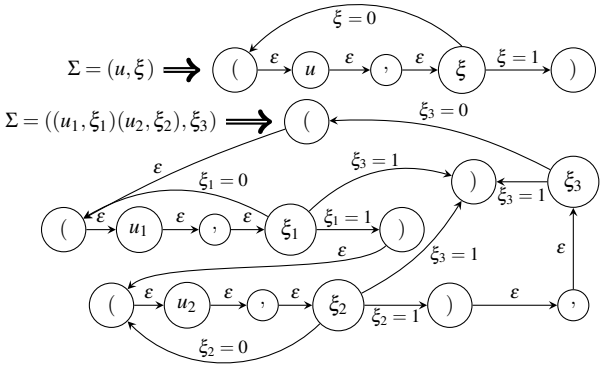


Figure 17: Example Transform: MDLe to Finite Automata

6.4.1 MDLe

The MDLe is a Specification Language with a Context-Free grammar [26]. Each string in the MDLe represents some control program. Each of those control programs can parse only a Regular Language. This is in contrast to the Motion Grammar which, describes the *System Language* for a *Context-Free System*. Each string in MDLe represents a control program that recognizes a regular system language.

THEOREM 9. *The System Language recognized by an MDLe string is Regular.*

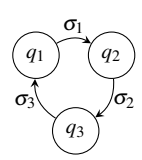
PROOF. Given that an MDLe controller is represented by a string in the MDLe language, we prove that the resulting System Language is regular by providing an algorithm to transform any MDLe string, Σ, into a Finite Automaton, $A = (S, E, d)$ that accepts the System Language L_g. MDLe string Σ is composed of tokens $\lfloor (\rfloor$, $\lfloor) \rfloor$, $\lfloor , \rfloor$, $u \in U$, and $\xi \in B'$. The algorithm in Fig. 16 creates the automaton A corresponding to Σ.

The resulting Finite Automaton encodes the evaluation rules for the MDLe string. Since we can transform Σ to a Finite Automaton, Σ must recognize a Regular System Language. □

Two examples of this conversion procedure are shown in Fig. 17, one simple case and one more complicated case. Unlike the transformation to Hybrid Automata in [26], we do not restrict repeated controllers in Σ to a single state in our system language Finite Automata. Notice also that there is ambiguity in the case of simultaneously active interrupt functions. [26] specifies that this is resolved via precedence among the different interrupts.

1: **procedure** G_o-TO-$G_n(G_o)$
2: **for all** $\langle q_i \rangle$ in G_o **do**
3: Create a production $\langle q'_i \rangle \rightarrow \kappa_{q_i} \langle q'_i \rangle$ in G_n
  where κ_{q_i} is a semantic rule for the controller
  to keep the system in the trim state.
4: **end for**
5: **for all** tokens $\lfloor \sigma_i \rfloor$ in G_o **do**
6: Create production $\langle \sigma'_i \rangle \rightarrow \lfloor \xi_i = 1 \rfloor$
  where ξ_i is an interrupt for the maneuver
7: Create production $\langle \sigma'_i \rangle \rightarrow \lfloor \xi_i = 0 \rfloor \kappa_{\sigma_i} \langle \sigma'_i \rangle$ in G_n
  where κ_{σ_i} is the semantic rule for the controller
8: **end for**
9: **for all** productions $\langle q_i \rangle \rightarrow \lfloor \sigma_j \rfloor \langle q_k \rangle$ in G_o **do**
10: create a production $\langle q'_i \rangle \rightarrow \langle \sigma'_j \rangle \langle q'_k \rangle$ in G_n.
11: **end for**
12: **end procedure**

Figure 18: Maneuver Automaton Conversion



$$\begin{aligned} \langle q_1 \rangle &\rightarrow \lfloor \sigma_1 \rfloor \langle q_2 \rangle \\ \langle q_2 \rangle &\rightarrow \lfloor \sigma_2 \rfloor \langle q_3 \rangle \\ \langle q_3 \rangle &\rightarrow \lfloor \sigma_3 \rfloor \langle q_1 \rangle \end{aligned}$$

(a) Offline Grammar

$$\begin{aligned} \langle q'_1 \rangle &\rightarrow \kappa_{q_1} \langle q'_1 \rangle \mid \langle \sigma'_1 \rangle \langle q'_2 \rangle \\ \langle \sigma'_1 \rangle &\rightarrow \lfloor \xi_1 = 0 \rfloor \\ &\mid \lfloor \xi_1 = 1 \rfloor \kappa_{\sigma_1} \langle \sigma'_1 \rangle \\ \langle q'_2 \rangle &\rightarrow \kappa_{q_2} \langle q'_1 \rangle \mid \langle \sigma'_2 \rangle \langle q'_3 \rangle \\ \langle \sigma'_2 \rangle &\rightarrow \lfloor \xi_2 = 0 \rfloor \\ &\mid \lfloor \xi_2 = 1 \rfloor \kappa_{\sigma_2} \langle \sigma'_2 \rangle \\ \langle q'_3 \rangle &\rightarrow \kappa_{q_3} \langle q'_1 \rangle \mid \langle \sigma'_3 \rangle \langle q'_1 \rangle \\ \langle \sigma'_3 \rangle &\rightarrow \lfloor \xi_3 = 0 \rfloor \\ &\mid \lfloor \xi_3 = 1 \rfloor \kappa_{\sigma_3} \langle \sigma'_3 \rangle \end{aligned}$$

(b) Online Grammar

Figure 19: Maneuver Automaton → Online Grammar.

COROLLARY 10. *Every MDLe string can be translated to a Motion Grammar.*

PROOF. The Motion Grammar is a Context-Free grammar for the System Language, and we can translate every MDLe string to a Finite Automaton accepting the System Language. Finite Automata are equivalent to Regular Grammars. Regular Grammars are a subset of Context-Free Grammars. □

From Corollary 10, we also observe that the Motion Grammar can control a broader class of systems than the MDLe. MDLe controllers can only accept regular languages while the Motion Grammar can accept Context-Free languages. Because Regular languages are a subset of Context-Free languages, the Motion Grammar can describe systems that the MDLe cannot.

6.4.2 Maneuver Automata

There are some important similarities between the Maneuver Automaton and the Motion Grammar. The Maneuver Automaton represents a hybrid system moving between a set of *trim* trajectories $q \in Q$ using a motion library of *maneuvers* $\sigma \in \Sigma$. This system is represented as a Finite Automaton with states Q and tokens Σ. It is possible to transform this representation into a grammar suitable for online control of the system. An example of this process is shown in Fig. 19. First, the Maneuver Automaton, Fig. 19(a) is rewritten as a Regular Grammar, G_o in Fig. 19(a), with one production of the form $\langle q_i \rangle \rightarrow \lfloor \sigma_j \rfloor \langle q_k \rangle$ to indicate each transition in the automaton. We then transform this offline grammar into an online grammar G_n according the algorithm in Fig. 18.

We also note that an arbitrary Maneuver Automaton cannot be directly transformed into a Motion Grammar. The Maneuver Automaton does not include information about how long to hold in trim states q or when to begin maneuvers σ. Thus, it does not represent a policy and it can be transformed only to an ambiguous grammar for the system language. Since the resulting system

language grammar is not Semantically LL(1), Theorem 7 indicates that it cannot be a Motion Grammar.

Even though we cannot directly transform a Maneuver Automaton to a Motion Grammar, this transformation is possible by providing some additional information necessary for LL(1) Semantics. Establishing precedence levels between the ambiguous productions or extending the system representation to include tokens, such as timeouts for coasting times, indicating when to begin maneuvers are possible ways to resolve the ambiguity. By augmenting the Maneuver Automaton with the additional information to achieve a policy, we can then derive a corresponding Motion Grammar.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented a novel approach to planning, perception, and control using grammars. Our Motion Grammar demonstrates that you can have both formal guarantees and computationally efficient performance, crucial properties in the development of robots that are safe and reliable. We showed how the Motion Grammar can be used to prove a system will be robust to uncertainty, responding to every possible situation, and we described how to develop such a complete grammar. We showed how to provide guarantees on the correct operation of the system. We also explained some particular constraints that arise in applying grammars to time-based physical systems. Finally, we have demonstrated the efficacy of this approach by developing a physical robotic system to play the game Yamakuzushi against a human opponent.

This work opens the way for many extensions to enhance the guarantees, expressiveness, and power the system through further use and refinement of our Motion Grammar. A parser generator could be developed to automatically produce a Motion Parser from the Motion Grammar, handling the Motion Grammar's deep recursion and LL(1) Semantics. Applying type theory could provide for stricter definitions and guarantees. There are also restricted classes of Context-Sensitive languages that can be efficiently parsed if the Context-Free model for the Motion Grammar should be insufficiently powerful for some problem [31]. The hierarchical structure exposed in the grammar could also be combined with MDP learning algorithms that exploit this structure [32]. We will continue exploring these approaches to improve the capabilities and guarantees of the resulting system.

8. ACKNOWLEDGMENTS

The authors thank Magnus Egerstedt for his insight throughout the development of our work on the Motion Grammar.

9. REFERENCES

[1] K. Fu. *Syntactic Pattern Recognition and Applications*. Prentice Hall, 1981.

[2] F. Han and S.C. Zhu. Bottom-up/top-down image parsing by attribute graph grammar. *International Conference on Computer Vision*, 2, 2005.

[3] P. Koutsourakis, L. Simon, O. Teboul, G. Tziritas, and N. Paragios. Single View Reconstruction Using Shape Grammars for Urban Environments. *ICCV*, 2009.

[4] A. Toshev, P. Mordohai, and B. Taskar. Detecting and Parsing Architecture at City Scale from Range Data. *International Conference on Computer Vision and Pattern Recognition*, 2010.

[5] S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2nd edition, 2002.

[6] M.L. Littman. *Algorithms for sequential decision making*. PhD thesis, Brown University, 1996.

[7] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[8] B. Stilman. *Linguistic geometry: from search to construction*. Kluwer Academic Publishers, 2000.

[9] R. Brooks. A robust layered control system for a mobile robot. *IEEE journal of robotics and automation*, 2(1):14–23, 1986.

[10] R.C. Arkin. *Behavior-Based Robotics*. MIT press, 1999.

[11] R.T. Pack. *IMA: The Intelligent Machine Architecture*. PhD thesis, Vanderbilt University, 2003.

[12] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid controllers for path planning: a temporal logic approach. *Proceedings of the 2005 IEEE Conference on Decision and Control*, 2005.

[13] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[14] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.

[15] R.C. Arkin. Governing lethal behavior: embedding ethics in a hybrid deliberative/reactive robot architecture. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 121–128. ACM, 2008.

[16] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *Analysis and Optimization of Systems*, 25(1):206–230, January 1987.

[17] C.G. Cassandras. *Discrete-Event Systems*. Springer, 2nd edition, 2008.

[18] E. Klavins. A language for modeling and programming cooperative control systems. In *IEEE international conference on robotics and automation*, volume 4, pages 3403–3410. IEEE; 1999, 2004.

[19] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-wesley Reading, MA, 1979.

[20] E. Klavins, R. Ghrist, D. Lipsky, E.E. Departjment, and WA Seattle. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962, 2006.

[21] RW Brockett. Formal Languages for Motion Description and Map Making. *Robotics*, 41:181–191, 1990.

[22] E. Frazzoli, MA Dahleh, and E. Feron. Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, 2005.

[23] M. Egerstedt. Motion Description Languages for Multi-Modal Control in Robotics. *Control Problems in Robotics*, pages 74–90, 2002.

[24] M. Egerstedt, T. Murphey, and J. Ludwig. Motion Programs for Puppet Choreography and Control. *Hybrid Systems: Computation and Control*, pages 190–202, 2007.

[25] V. Manikonda, PS Krishnaprasad, and J. Hendler. Languages, Behaviors, Hybrid Architectures and Motion Control. *Mathematical Control Theory*, pages 199–226, 1998.

[26] D. Hristu-Varsakelis, M. Egerstedt, and PS Krishnaprasad. On the structural complexity of the motion description language MDLe. In *42nd IEEE Conference on Decision and Control, 2003. Proceedings*, pages 3360–3365, 2003.

[27] J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson, 3rd edition, 2005.

[28] A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, & Tools*. Pearson, 2nd edition, 2007.

[29] July 2010. http://gcc.gnu.org/gcc-3.4/changes.html.

[30] J.J.E. Slotine, W. Li, et al. *Applied nonlinear control*. Prentice Hall Englewood Cliffs, NJ, 1991.

[31] A.K. Joshi, K. Vijay-Shanker, and D. Weir. The convergence of mildly context-sensitive grammar formalisms. *Foundational issues in natural language processing*, pages 31–81, 1991.

[32] T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artifcial Intelligence Research*, 13:227–303, 2000.