IEEE International Conference on Robotics and Automation, May 2010

The Motion Grammar for Physical Human-Robot Games

Neil Dantam, Pushkar Kolhe, and Mike Stilman

***Abstract*— We introduce the Motion Grammar, a powerful new representation for robot decision making, and validate its properties through the successful implementation of a *physical* human-robot game. The Motion Grammar is a formal tool for task decomposition and hybrid control in the presence of significant online uncertainty. In this paper, we describe the Motion Grammar, introduce some of the formal guarantees it can provide, and represent the entire game of human-robot chess through a single formal language. This language includes game-play, safe handling of human motion, uncertainty in piece positions, misplaced and collapsed pieces. We demonstrate the simple and effective language formulation through experiments on a 14-DOF manipulator interacting with 32 objects (chess pieces) and an unpredictable human adversary.**

***Index Terms*— Manipulation, Hybrid Control, Formal Methods, Planning**

I. Introduction

Future robots will be required to respond safely and completely to physical interaction with humans in complex environments ranging from hospital assistance to search-and-rescue operations. Presently, researchers use a number of distinct methods including collision-free motion planning and compliant control in order to design strategies for specific applications. Since it is often infeasible to experimentally validate a sufficient space of the possible states the robot may encounter, it is important to *prove* that the robot follows a policy that handles all states that may occur. The *Motion Grammar* is an efficient representation for complete strategies that combine existing methods, resulting in provably complete robot manipulation in uncertain environments.

In order to demonstrate the function and effectiveness of the Motion Grammar, as well as our method of grammar construction, we apply our representation to the complex domain of *physical human-robot chess*. Typically, chess is thought of as a discrete strategy game. In reality, it is also a complex continuous manipulation challenge that requires the robot to handle numerous sources of uncertainty. For instance, human adversaries may incorrectly displace pieces, pieces may collapse sideways, and most importantly humans may endanger themselves by entering the workspace of the robot. Using the Motion Grammar, it is simple to hierarchically specify response strategies to each of these potential threats and achieve a complete global policy.

This paper is organized as follows. Sect. II places the Motion Grammar in the context of existing work on formal methods for robot control. Sect. III details the proposed representation and describes its key characteristics. Sect. IV demonstrates its expressive power through the domain of human-robot chess. We show that the complete system was able to play chess even with a highly uncooperative adversary. Sect. V presents the experimental results of our implementation in a variety of circumstances. Sect. VI summarizes this work and describes future directions of research.

The authors are with the Center for Robotics and Intelligent Machines (RIM) at the Georgia Institute of Technology, Atlanta, Georgia, 30332, USA. Email: ntd@gatech.edu, pushkar@cc.gatech.edu, mstilman@cc.gatech.edu.

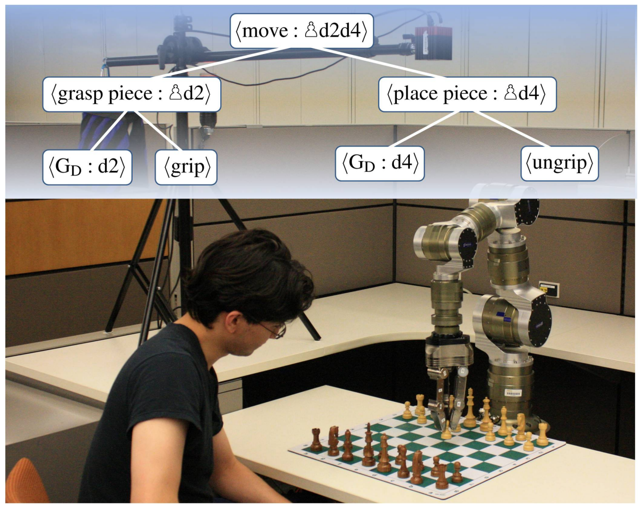


Fig. 1. Our experimental setup for human-robot chess and a partial parse-tree indicating the robot's plan to perform a chess move.

II. Related Work

The Motion Grammar [1] is a linguistic approach to robot control with significant benefits over existing techniques. Our method provides immediate response to uncertainty and efficiency in the context of many degrees of freedom. Formal guarantees on parsing algorithms enable the robot to react without lengthy deliberative planning. Furthermore, the structure imposed by the grammar yields rigorous guarantees that controllers will respond to all contingencies. The hierarchical nature of grammatical *productions* and corresponding *parse trees* allows robot tasks to be recursively divided into a number of simpler subtasks. To our knowledge, these benefits are not found together in any prior methods for robot control.

The most prominent efficient control architectures for robot manipulators are behavior-based methods introduced by Brooks [2] and Arkin [3]. However, the primary approach to validation has been experimental due to infeasible computation time requirements of earlier formal methods [4]. Instead, [3] formalizes behavior semantics with schema theory, yet it does not ensure completeness. [5] proposes an Ethical Architecture to assist with correctness in the military domain. The Motion Grammar ensures complete operation through a domain-independent specification of robot response.

Common techniques for planning and policy generation trade efficiency for analytical properties such as completeness. Classical planners guarantee completeness by reducing planning to theorem proving; however, inference in first order logic is NP-complete [6]. Furthermore, symbolic methods do not explicitly address continuous domains. Partially Observable Markov Decision Processes generate solutions for uncertain environments; however, they also pose NP-complete problems [7]. The Motion Grammar employs Context-Free languages, guaranteeing $O(n^3)$ runtime [1, 8]. Motion Grammars yield both provable completeness and efficiency.

Alternative *formal methods* make similar compromises. [9] describes the Computation and Control Language (CCL), a Turing-Complete language for robot control; however, Rice's Theorem prevents proving nontrivial properties for arbitrary programs in the CCL [10]. [11] solves graph grammars for many simple agents. While any application of the Motion Grammar (Sect. IV) is domain specific, the method (Sect. III) is domain independent. [12–14] use linear temporal logic to formally describe uncertain multi-agent robotics by discretizing the 2D environment. This requires a number of states that is exponential in the degrees of freedom (DOF). For our manipulation task with a 14-DOF robot and 32 movable objects, discretization is not computationally feasible. The Motion Grammar avoids discretization through continuous domain semantics and discrete events that represent the task.

The Motion Grammar improves upon the online response of other hybrid control approaches. Hybrid [15] and Maneuver [16] Automata switch continuous controllers in a Finite State Machine. In contrast, the Motion Grammar uses Context-Free Grammars, which allow the controller to keep a memory of prior actions, build models, and improve decisions. The Motion Description Language (MDL) [17–19] describes robot operation with strings of symbols each representing continuous-valued controllers. These strings are generated off-line. Instead, the Motion Grammar parses a string of tokenized sensor readings online. The associated parse tree evolves to represent the history of system execution. MDLe [20] also handles reactive planning; however, as shown in [21], it is not strictly more expressive than Hybrid Automata. More detail on the expressive power of the Motion Grammar in relation to other methods is discussed in [1].

In the context of safe human-robot interaction, [22] demonstrates safe response of a knife-wielding robot based on collision detection when a human enters the workspace. Other approaches to safe physical interaction between humans and robots surveyed by [23] and [24] suggests specific methods for different types of safety. The Motion Grammar builds on such methods by providing both task-level guarantees and a common structure for existing techniques.

In general, grammars are a common formal tool in Linguistics and Computer Science. A number of studies related to robotics focus on image processing. These include Fu's syntactic pattern recognition [25], Han's attribute graph grammars for relationships between planes in a scene [26] and syntactic visual modeling methods by Koutsourakis [27] and Toshev [28]. B. Stilman's Linguistic Geometry applies

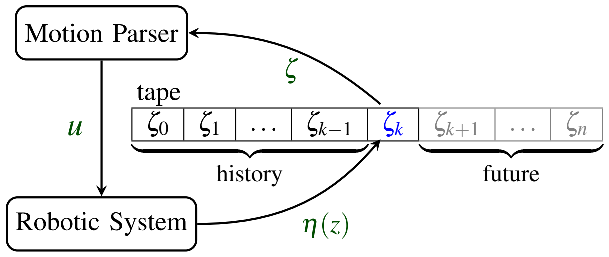


Fig. 2. Operation of the Motion Parser.

grammars to deliberative planning in adversarial games [29]. These works do not address robot control, which is the focus of our Motion Grammar.

Our experimental domain of robot chess has previous implementations. [30] describes a specially designed robot arm and board. [31] developed a robot chess player using a specialized analytical inverse kinematics. Instead of focusing on chess play, we use the context of the physical human-robot game to demonstrate the Motion Grammar. We present a general solution implemented on a general-purpose robot arm using general kinematics approaches. Furthermore, we provide features and safeties beyond game-play and manipulation.

III. The Motion Grammar

The Motion Grammar represents the operation of a robotic system through a Context-Free Grammar. This grammar is used to generate the *Motion Parser* which drives the robot. This method is illustrated in Fig. 2. The output of the system z is discretized by function $\eta(z)$ into a stream of tokens ζ for the parser to read. Based on the sequence of tokens seen so far, the parser picks the correct production to expand at each step. The semantic rule for that production uses the history of continuous z values to generate the command u. The Motion Grammar and Motion Parser are defined in Fig. 3. The Motion Grammar represents the language of robot sensor readings. *The Motion Parser is an interpreter that translates the language of sensor readings into the language of controllers or actuator commands.*

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
η	tokenizing function, $\eta : \mathscr{Z} \mapsto Z$
V	set of nonterminals
P	set of productions
K	set of semantic rules, each associated with one and only one production.
$S \in V$	start variable

The Motion Parser

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

Fig. 3. Definition of the Motion Grammar.

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 sensor readings. In some production $A \to X_0 \dots X_n$, 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. Ultimately, the parser reaches a semantic rule to calculate the robot's input $u \in \mathscr{U}$, and sends this value to the robot.

We emphasize that the Motion Grammar is not a Domain Specific Language or Robot Programming Language [32] but rather the direct application of linguistic theory to the problem of 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 in Backus-Naur Form augmented with semantic rules [33]. Nonterminals are represented between angle brackets $\langle\rangle$, tokens are represented between floor brackets $\lfloor\rfloor$, and semantic rules are represented between curly braces $\{\}$.

1) General Application to 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. After the productions, the semantic rules K for each production are created to perform calculations over 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 start variable S is selected from V as the top level of the hierarchical decomposition, and the grammar is completed.

2) Benefits of The Motion Grammar: We use a Context-Free model for the Motion Grammar because it represents 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 representation. Context-Sensitive Languages are somewhat more powerful than Context-Free, but the Context-Sensitive decision problem is PSPACE-Complete. Recursively-enumerable languages are very powerful, but by Rice's Theorem, any nontrivial property of a Turing Machine is unprovable [10]. Because 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 provable.

3) Completeness: The formulation of the controller as a grammar allows us to guarantee that the robot will respond to all situations. By enumerating the nonterminals and the productions for each nonterminal, one can determine which tokens would cause a syntax error in expanding any nonterminal. If there exists a possible syntax error in expanding nonterminal $v \in V$ and the robotic system is actually capable of producing the offending string σ during expansion of v, we must extend the grammar to handle this additional case. This is done by adding a production for v to handle σ. In turn, this may require the addition of further nonterminals and productions. We propose this iterative strategy as a novel method for ensuring that a robot will *never get stuck*.

IV. Grammars for Human-Robot Chess

A. Experimental Setup

We performed our experiments using a Schunk LWA3 7-DOF robot arm with a Schunk SDH 7-DOF, 3-fingered hand as shown in Fig. 1. A wrist mounted 6-axis force-torque sensor and finger-tip pressure distribution sensors provided force control feedback. The robot manipulated pieces in a standard chess set, and a Mesa SwissRanger 4000 mounted overhead allowed it to locate the individual pieces. Domain-specific planning of chess moves was done with the Crafty chess engine [34]. The perception, motion planning, and control software was implemented primarily in C/C++ and Common Lisp using message-passing IPC via shared memory and TCP running on Ubuntu Linux 10.04. The lowest-levels of our grammatical controller operate at a 1kHz rate.

TABLE I

Chess Grammar Tokens

Sensor Tokens		
Token	$\eta(z)$	**Description**
$\lfloor 0 \rfloor$	$t < t_1 \vee \lVert\mathbf{x} - \mathbf{x}_1\rVert > \varepsilon_x \vee \lVert\dot{\mathbf{q}}\rVert > \varepsilon_{\dot{q}}$	Not at Traj. End
$\lfloor 1 \rfloor$	$\neg \lfloor 0 \rfloor$	At Traj. End
$\lfloor$limit$\rfloor$	$\lVert\mathbf{F}\rVert > F_{\max}$	Force Limit
$\lfloor$grasped$\rfloor$	$\int \rho dA > \varepsilon_{\int \rho}$	Pressure sum limit
$\lfloor$ungrasped$\rfloor$	$\neg \lfloor$grasped$\rfloor$	Pressure sum limit
Chessboard Tokens		
Token		**Description**
$\lfloor$set$\rfloor$		board is properly set
$\lfloor$moved$\rfloor$		opponent has completed move
$\lfloor$checkmate$\rfloor$		checkmate on board
$\lfloor$resign$\rfloor$		a player has resigned
$\lfloor$draw$\rfloor$		players have agreed to draw
$\lfloor$cycle(x)$\rfloor$		x is in a cycle of visited during
Perception Tokens		
Token	$\eta(z)$	**Description**
$\lfloor$obstacle$\rfloor$	$w(\mathbf{C}) < w_k$	Robot workspace occupied
$\lfloor$occupied(x)$\rfloor$	$w(x) > w_m in$	Piece is present in x
$\lfloor$clear(x)$\rfloor$	$\neg \lfloor$occupied(x)$\rfloor$	No piece in x
$\lfloor$fallen(x)$\rfloor$	height$(x) < h_{\min}$	Piece is fallen
$\lfloor$offset(x)$\rfloor$	mean$(x) -$ pos$(x) > \varepsilon$	Piece is not centered
$\lfloor$moved$\rfloor$	$C_r \neq C_c$	Boardstate is different
$\lfloor$misplaced(x)$\rfloor$	$C_r(x) \neq C_c(x)$	Piece is missing

1) Tokenizing: The tokens in the Motion Grammar for Chess are based on both the sensor readings and chessboard state. A summary of token types is given in Table I. Position thresholds, velocity thresholds, and timeouts indicate when the robot has reached the end of a trajectory. Force thresholds and position thresholds indicate when the robot is in a safe operating range.

$$
\begin{array}{rcll}
\langle \mathrm{G} \rangle & \rightarrow & \langle \mathrm{G_D} \rangle \mid \langle \mathrm{G_L} \rangle & (1) \\
\langle \mathrm{G_D} \rangle & \rightarrow & \lfloor 1 \rfloor \mid \langle \kappa \rangle \langle \mathrm{G_D} \rangle & (2) \\
\langle \mathrm{G_L} \rangle & \rightarrow & \lfloor \mathrm{limit} \rfloor \mid \langle \kappa \rangle \langle \mathrm{G_L} \rangle & (3) \\
\langle \kappa \rangle & \rightarrow & \lfloor 0 \rfloor \ \{ \dot{\mathbf{q}} = \mathbf{J}^* \left(\dot{\mathbf{x}} - \mathbf{K}_p \left(\mathbf{x} - \mathbf{x}_r \right) - \mathbf{K}_f \left(\mathbf{F} - \mathbf{F}_r \right) \right) \} & (4)
\end{array}
$$

Fig. 4. Grammar fragment for guarded moves

```
⟨G⟩()
switch
    case ⌊0⌋ :
        call⟨κ⟩
        return call⟨G⟩
    case ⌊1⌋ :
        return ⟨G_D⟩
    case ⌊limit⌋ :
        return ⟨G_L⟩
```

(a) Nonterminal $\langle \mathrm{G} \rangle$

```
⟨κ⟩()
$\mathbf{x}_e = \mathbf{x} - \mathbf{x}_r$
$\mathbf{F}_e = \mathbf{F} - \mathbf{F}_r$
$\dot{\mathbf{x}}_r = \dot{\mathbf{x}} - \mathbf{K}_p \mathbf{x}_e - \mathbf{K}_f \mathbf{F}_e$
$\dot{\mathbf{q}} = \mathbf{J}^* \mathbf{x}_r$
```

(b) Nonterminal $\langle \kappa \rangle$

Fig. 5. Parser for Cartesian Trajectory Control

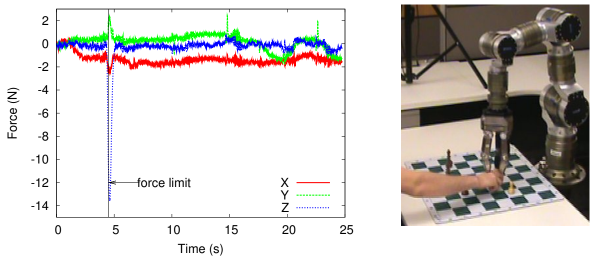
2) *Parsing:* Once the Motion Grammar for the task is developed, it must be transformed into the Motion Parser. For our chess application, we used a hand-written recursive descent parser, an approach also employed by GCC [35]. A recursive descent parser is written as a set of mutually-recursive procedures, one for each nonterminal in the grammar. 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.

B. Example: Trajectory Control

To move the LWA3, we used Cartesian space parabolic blends [32] with a damped least squares Jacobian inverse $\mathbf{J}^*$ [36]. To prevent damage to equipment and injury to individuals, we implemented guarded moves based on feedback from the wrist Force-Torque sensor. The entire controller is expressed in the grammar in Fig. 4.

In Fig. 4, line (1) indicates the controller nonterminal $\langle \mathrm{G} \rangle$ can expand to a nonterminal that reaches its destination $\langle \mathrm{G_D} \rangle$ or a nonterminal that reaches a limit $\langle \mathrm{G_L} \rangle$. Line (2) indicates that $\langle \mathrm{G_D} \rangle$ can either expand to a token $\lfloor 1 \rfloor$ indicating the destination is reached or to nonterminal $\langle \kappa \rangle$ followed by recursing on itself. Line (3) indicates that $\langle \mathrm{G_L} \rangle$ can either expand to a token $\lfloor \mathrm{limit} \rfloor$ indicating a force limit or to nonterminal $\langle \kappa \rangle$ followed by recursing on itself. Line (4) indicates that nonterminal $\langle \kappa \rangle$ expands to token $\lfloor 0 \rfloor$ indicating the destination is not yet reached and then it executes the semantic rule to calculate the desired joint velocities using the Jacobian damped inverse $\mathbf{J}^*$ from a feed-forward velocity $\dot{\mathbf{x}}$ and linear feedback terms for position $\mathbf{x}$ and force $\mathbf{F}$.

This grammar fragment is implemented using the procedures in Fig. 5. Procedure $\langle \mathrm{G} \rangle$ implements productions (1) to (3). Procedure $\langle \kappa \rangle$ implements the semantic rule in production (4).



(a) Forces (b) Contact

Fig. 6. Grammatical guarded moves safely protecting the human player.

$$
\begin{array}{rcl}
\langle \mathrm{recover} : \mathbf{x}, \mathrm{z} \rangle & \rightarrow & \langle \mathrm{G_D} : \mathbf{x} \rangle \langle \mathrm{pinch} \rangle \langle \mathrm{G_D} : \mathbf{x} + \mathrm{h(z)} \hat{k}, \frac{\pi}{6} \rangle \langle \mathrm{release} \rangle \\
\langle \mathrm{pinch} \rangle & \rightarrow & \lfloor \mathrm{grasped} \rfloor \mid \lfloor \mathrm{ungrasped} \rfloor \langle \mathrm{pinch} \rangle
\end{array}
$$

Fig. 7. Grammar fragment for recovering fallen pieces

V. Experimental Results

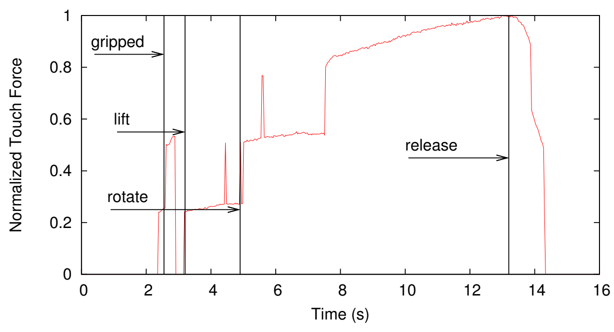
A. Guarded Moves

Our implementation of guarded moves using the Motion Grammar allows the human and robot to safely operate in the same workspace. When the parser detects a force limit, it stops and backs off, preventing damage or injury. The plot in Fig. 6(a) shows the forces encountered by the robot in this situation. The large spike at 4.7s occurs when the robot's end-effector makes contact with the human's hand pictured in Fig. 6(b). Production (3) of our grammar *guarantees* that when this situation occurs, the robot will stop. The robot can then safely reattempt its move after the human removes his hand from the piece.

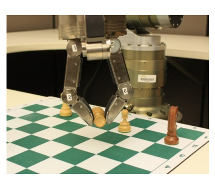
This example shows the importance of fast online control that is possible using the Motion Grammar. The robot must respond immediately to the dangerous situation of impact with the human. The polynomial runtime performance of Context-Free parsers means that the grammatical controller can respond quickly enough, and the syntax of the grammar guarantees that the robot will stop moving.

B. Fallen Pieces

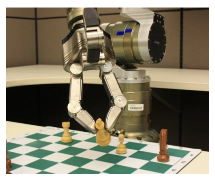
The grammar to set fallen pieces upright has a fairly simple structure but builds upon the previous grammars to perform a more complicated task, demonstrating the advantages of a hierarchical decomposition for manipulation. This grammar is shown in Fig. 7, and Fig. 8 shows a plot of the finger tip forces and pictures for this process. The production $\langle \mathrm{recover} : \mathbf{x}, \mathrm{z} \rangle$ will pick up fallen piece z at location $\mathbf{x}$. The nonterminal $\langle \mathrm{G_D} : \mathbf{x} \rangle$ moves the arm to location $\mathbf{x}$. The production $\langle \mathrm{pinch} \rangle$ will grasp the piece by squeezing tighter until the fingertip pressure sensors indicate a sufficient force. The production $\langle \mathrm{G_D} : \mathbf{x} + \mathrm{h(z)} \hat{k}, \frac{\pi}{6} \rangle$ will lift the piece sufficiently high above the ground and rotate it so that it can be replaced upright. Finally the nonterminal $\langle \mathrm{release} \rangle$ will release the grasp on the piece setting it upright.



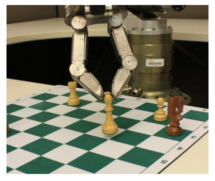
(a) Touch Force: Knight



(b) Grasped, Rook



(c) Rotated, Queen



(d) Finished, Bishop

Fig. 8. Robot recovering fallen pieces

$\langle$reset board$\rangle$	$\rightarrow$	$\lfloor$set$\rfloor$ | $\lfloor$misplaced$(x)\rfloor$ $\langle$reset : x, home$(x)\rangle$
$\langle$reset : $x_0, x_1\rangle$	$\rightarrow$	$\lfloor$clear$(x_1)\rfloor$ $\langle$move : $x_0, x_1\rangle$
	|	$\lfloor$occupied$(x_1)\rfloor$ $\langle$reset : x_1, home$(x_1)\rangle\langle$move : $x_0, x_1\rangle$
	|	$\lfloor$cycle$(x_1)\rfloor$ $\langle$move : x_1, rand()$\rangle$

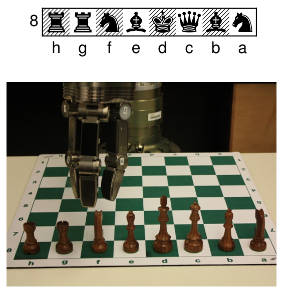
Fig. 9. Grammar fragment to reset chessboard

C. Board Resetting

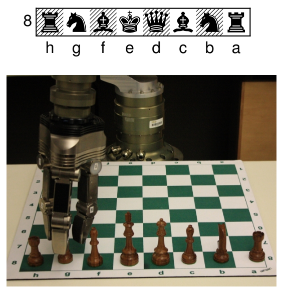
The problem of resetting the chess board presents an interesting grammatical structure. If the home square of some piece is occupied, that square must first be cleared before the piece can be reset. Additionally, if a cycle is discovered among the home squares of several pieces, the cycle must be broken before any piece can be properly placed. The grammatical productions to perform these actions a given in Fig. 9.

An example of this problem is shown in Fig. 10(a) where all of Blacks's Row 8 pieces have been shifted right by one square. The parse tree for this example is shown in Fig. 10(c), rooted at ⟨reset board⟩. As the robot recurses through the grammar in Fig. 9, chaining an additional ⟨reset⟩ for each occupied cell, it eventually discovers that a cycle exists between the pieces to move. To break the cycle, one piece, ♘c1, is moved to a random free square, χ. With the cycle broken, all the other pieces can be moved to their home squares. Finally, ♘χ can be moved back to its home square. This sequence of board state tokens and ⟨move⟩ actions can be seen by tracing the leaves of the parse tree, shown also beginning from PLAN in Fig. 10(c).

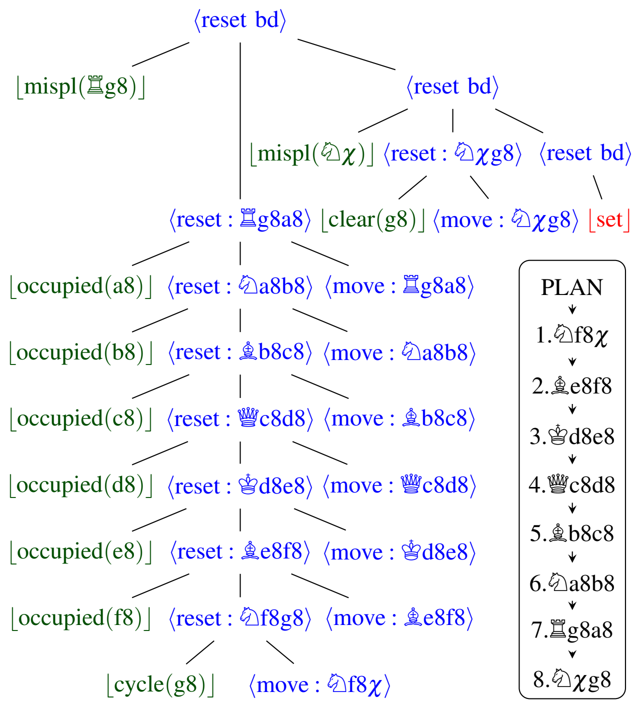
Observe that as the parser searches through the chain of pieces that occupy each other's home squares, it is effectively building up a stack of the moves to make. This demonstrates the benefits of the increased power of Context Free Languages over the Regular languages commonly used in other hybrid control systems. Regular languages, equivalent to finite state machines, lack the power to represent this arbitrary depth search.



(a) Board position - Initial



(b) Board position - Final



(c) Motion grammar parse tree and plan for resetting the board.

Fig. 10. Example of board resetting

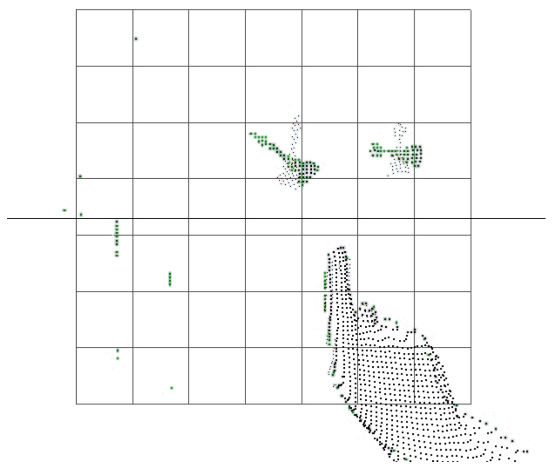
Claim 1: Let n be the number of misplaced pieces on the board. The grammar in Fig. 9 will reset the board with at most $1.5n$ moves.

Proof: Every misplaced piece not in a cycle takes one move to reset to its proper square. Every cycle causes one additional move in order to break the cycle. A cycle requires two or more pieces, so there can be at most $0.5n$ cycles. Thus one move for every piece and one move for $0.5n$ cycles give a maximum of $1.5n$ moves. ■

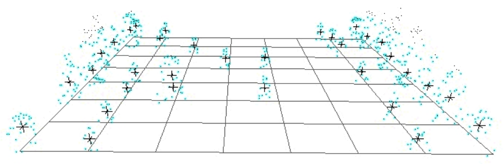
D. Perception and Board Tokens

To play the game of chess, we integrated our controller with the Crafty [34] chess engine. The Crafty *boardstate* serves as the model of the position of the chessboard. The MESA SR4000 point cloud is discretized by clustering to generate the tokens given in Table I. We use a finite moving average filter over the point cloud to remove sensor noise.

Obstacles are found in the point cloud $\mathbf{C}$ by a weighting function $w(\mathbf{C})$ which finds out if the workspace above the



(a) Detecting obstacle (Black points are obstacles. Red/Green points indicate the orientation of each fallen piece.)



(b) Finding offsets for all pieces

Fig. 11. Perception with point cloud is discretized into tokens.

chessboard is occupied or not. An example of an obstacle is shown in Fig. 11(a). We give the following attributes to each cluster in the point cloud: the weight of the cluster, the height of the cluster from the chessboard, the maximum area occupied by a cross section parallel to the chessboard, and the mean of the cluster. Here, the weight is denoted by $w(.)$, and it counts the number of points in that cluster. The height of the cluster is the highest point in the cluster. The maximum area occupied by the chess piece is expressed as a ratio of its width and length. If the ratio is above a certain threshold, we can easily conclude that a chess piece is fallen. The longest side of the chess piece gives its orientation. The mean point gives the center of the chess piece. Fig. 11 shows these attributes in the point cloud.

If an obstacle is found, it is denoted by $\lfloor\text{obstacle}\rfloor$. Nearest Neighbor over the entire chessboard determines all squares x with $\lfloor\text{occupied(x)}\rfloor$. If a piece is not placed exactly in the center of the square, an offset is computed and denoted by $\lfloor\text{offset(x)}\rfloor$. The boardstate retrieved from perception is termed C_r and the one from the Crafty engine is C_c. C_r is with C_c reported by Crafty to find whether a move has been made. If a move has been made, then $\lfloor\text{clear(x)}\rfloor$ and $\lfloor\text{misplaced(x)}\rfloor$ are determined. Our perception algorithm also finds out the height, orientation, and the area occupied by a horizontal cross-section for each piece. Using this and a recursive nearest neighbor algorithm for clustering, we can find all $\lfloor\text{fallen(x)}\rfloor$ as shown in Table I.

$$
\begin{array}{rcl}
\langle\text{game}\rangle & \rightarrow & \langle\text{act}\rangle\langle\text{end}\rangle \mid \langle\text{act}\rangle\langle\text{game}'\rangle \\
\langle\text{game}'\rangle & \rightarrow & \langle\text{wait}\rangle\langle\text{end}\rangle \mid \langle\text{wait}\rangle\langle\text{game}\rangle \\
\langle\text{end}\rangle & \rightarrow & \lfloor\text{checkmate}\rfloor \mid \lfloor\text{resign}\rfloor \mid \lfloor\text{draw}\rfloor \\
\langle\text{act}\rangle & \rightarrow & \langle\text{fix}\rangle\langle\text{turn}\rangle\langle\text{fix}\rangle \\
\langle\text{fix}\rangle & \rightarrow & \langle\text{end}\rangle \mid \lfloor\text{fallen}:\text{x},\text{z}\rfloor\,\langle\text{recover}:\text{x},\text{z}\rangle\langle\text{fix}\rangle \mid \varepsilon \\
\langle\text{turn}\rangle & \rightarrow & \langle\text{move}:\text{x}_0,\text{x}_1\rangle \mid \langle\text{capture}:\text{x}_0,\text{x}_1\rangle \\
 & \mid & \langle\text{castle}\rangle \mid \langle\text{castle queen}\rangle \mid \langle\text{en passent}\rangle \\
 & \mid & \langle\text{resign}\rangle \mid \langle\text{draw}\rangle \\
\langle\text{wait}\rangle & \rightarrow & \lfloor\text{moved}\rfloor \mid \langle\text{wait}\rangle \\
\langle\text{move}:\text{x}_0,\text{x}_1\rangle & \rightarrow & \langle\text{grasp piece}:\text{x}_0\rangle\langle\text{place piece}:\text{x}_1\rangle \\
\langle\text{grasp piece}:\text{x}\rangle & \rightarrow & \langle\text{G}_\text{L}:\text{x}\rangle\langle\text{grasp piece}:\text{x}\rangle \mid \langle\text{G}_\text{D}:\text{x}\rangle\langle\text{grip}\rangle \\
\langle\text{place piece}:\text{x}\rangle & \rightarrow & \langle\text{G}_\text{L}:\text{x}\rangle\langle\text{place piece}:\text{x}\rangle \mid \langle\text{G}_\text{D}:\text{x}\rangle\langle\text{ungrip}\rangle \\
\langle\text{grip}\rangle & \rightarrow & \lfloor\text{grasped}\rfloor \mid \lfloor\text{ungrasped}\rfloor\,\langle\text{grip}\rangle \\
\langle\text{capture}:\text{x}_0,\text{x}_1\rangle & \rightarrow & \langle\text{take}:\text{x}_1\rangle\langle\text{move}:\text{x}_0,\text{x}_1\rangle \\
\langle\text{take}:\text{x}\rangle & \rightarrow & \langle\text{move}:\text{x},\text{offboard}\rangle \\
\langle\text{castle}\rangle & \rightarrow & \langle\text{move}:\text{♔e1g1}\rangle\langle\text{♖h1f1}\rangle \\
\langle\text{castle queen}\rangle & \rightarrow & \langle\text{move}:\text{♔e1c1}\rangle\langle\text{♖a1d1}\rangle \\
\langle\text{en passent}:\text{x}\rangle & \rightarrow & \langle\text{take}:\text{x}-1\rangle\langle\text{move}:\text{♙x}\rangle \\
\langle\text{resign}\rangle & \rightarrow & \langle\text{G}_\text{L}:\text{♔}+1\rangle\langle\text{resign}\rangle \mid \langle\text{G}_\text{D}:\text{♔}+1\rangle\langle\text{resign}'\rangle \\
\langle\text{resign}'\rangle & \rightarrow & \langle\text{G}_\text{L}:\text{♔}-1\rangle\langle\text{resign}'\rangle \mid \langle\text{G}_\text{D}:\text{♔}-1\rangle
\end{array}
$$

Fig. 12. Grammar Productions for Chess Game

E. Full Game

The entire motion planning and control policy is specified in the grammar in Fig. 12. This grammar describes the game, $\langle\text{game}\rangle$, as consisting of an alternating sequence of the robot moving, $\langle\text{act}\rangle$, followed by the human moving, $\langle\text{wait}\rangle$, until the game has ended, $\langle\text{end}\rangle$, via checkmate, resignation, or draw. When it is the robot's turn, it will correct any fallen pieces, $\langle\text{fix}\rangle$, make its move, and then again correct any pieces that may have fallen while it was making the move. Making a move, $\langle\text{turn}\rangle$, can be either a simple move between squares, a capture, a castle, en passent, or a draw or resignation. A simple piece move, $\langle\text{move}\rangle$, requires first grasping the piece, then placing it on the correct square. To grasp the piece, the robot will move its hand around then piece then tighten its grip, $\langle\text{grip}\rangle$, until there is sufficient pressure registered on the touch sensors. To capture a piece, the robot will remove the captured piece from the board, $\langle\text{take}\rangle$, and then move the capturing piece onto that square. A $\langle\text{castle}\rangle$ requires the robot to move both the rook and the king. For $\langle\text{en passent}\rangle$, the robot will $\langle\text{take}\rangle$ the captured pawn and then move its own pawn to the destination square. Finally, to resign – indicating a failure in chess strategy, not motion planning – the robot moves its end-effector through the square occupied by the king, knocking it over. By following the rules of this grammar, our system will play chess with the human opponent.

VI. Conclusions and Future Work

In this paper, we presented a novel approach for planning and control using grammars. We have demonstrated the capabilities of our Motion Grammar for the manipulation of chess pieces in interactive human-robot gameplay representing the entire controller with 18 tokens, 26 nonterminals, and 51 productions. The Motion Grammar provides safe and reliable performance with low computational overhead. This same approach can be used for both the realtime control in Fig. 6 and fast deliberative planning in Fig. 10.

We will continue applying the Motion Grammar to new systems, particularly those requiring hybrid control approaches and those where guarantees on performance and reliability are critical. Additionally, we will extend the underlying theoretical basis of the Motion Grammar to increase its flexibility and strengthen the guarantees it can provide, and we will develop tools to simplify and automate the construction of grammars and parsers.

Acknowledgments

The impetus for our experiments in the chess domain comes from work led by Hyun-Soo Yi on Golem Chesster which took 2nd place at the 2010 AAAI Robot Chess Challenge.

References

[1] N. Dantam and M. Stilman, "The motion grammar: Linguistic perception, planning, and control," College of Computing, Georgia Institute of Technology, Tech. Rep. GT-GOLEM-2010-001, 2010.

[2] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal of robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

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

[4] R. Pack, "IMA: The intelligent machine architecture," Ph.D. dissertation, Vanderbilt University, 2003.

[5] Arkin, R.C., "Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture," in *ACM/IEEE International Conf. on Human Robot Interaction*. ACM, 2008, pp. 121–128.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A modern Approach*, 2nd ed. Prentice Hall, 2002.

[7] M. Littman, "Algorithms for sequential decision making," Ph.D. dissertation, Brown University, 1996.

[8] J. Earley, "An efficient context-free parsing algorithm," *Communications of the ACM*, vol. 13, no. 2, pp. 94–102, 1970.

[9] E. Klavins, "A language for modeling and programming cooperative control systems," in *IEEE Intl Conf. on Robotics and Automation*, vol. 4. IEEE; 1999, 2004, pp. 3403–3410.

[10] M. Sipser, *Introduction to the Theory of Computation*. Intl. Thomson Publishing, 1996.

[11] E. Klavins, R. Ghrist, D. Lipsky, E. Departjment, and W. Seattle, "A grammatical approach to self-organizing robotic systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 949–962, 2006.

[12] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Hybrid controllers for path planning: a temporal logic approach," in *IEEE Conf. on Decision and Control*, 2005.

[13] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 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*, vol. 26, no. 1, pp. 48–61, 2010.

[15] C. Cassandras, *Discrete-Event Systems*, 2nd ed. Springer, 2008.

[16] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[17] R. Brockett, "Formal languages for motion description and map making," *Robotics*, vol. 41, pp. 181–191, 1990.

[18] M. Egerstedt, "Motion description languages for multi-modal control in robotics," *Control Problems in Robotics*, pp. 74–90, 2002.

[19] M. Egerstedt, T. Murphey, and J. Ludwig, "Motion programs for puppet choreography and control," in *Hybrid Systems: Computation and Control*. Springer, 2007, pp. 190–202.

[20] V. Manikonda, P. Krishnaprasad, and J. Hendler, "Languages, behaviors, hybrid architectures and motion control," *Mathematical Control Theory*, pp. 199–226, 1998.

[21] D. Hristu-Varsakelis, M. Egerstedt, and P. Krishnaprasad, "On the structural complexity of the motion description language mdle," in *IEEE Conf. on Decision and Control*, 2003, pp. 3360–3365.

[22] A. De Luca, A. Albu-Schaffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the DLR-III lightweight manipulator arm," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2006, pp. 1623–1630.

[23] A. De Santis, B. Siciliano, A. De Luca, and A. Bicchi, "An atlas of physical human-robot interaction," *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253–270, 2008.

[24] M. Giuliani, C. Lenz, T. Müller, M. Rickert, and A. Knoll, "Design principles for safety in human-robot interaction," *Intl. Journal of Social Robotics*, pp. 1–22, 2010.

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

[26] F. Han and S. Zhu, "Bottom-up/top-down image parsing by attribute graph grammar," in *Intl. Conf. on Computer Vision*, vol. 2, 2005.

[27] P. Koutsourakis, L. Simon, O. Teboul, G. Tziritas, and N. Paragios, "Single View Reconstruction Using Shape Grammars for Urban Environments," in *Intl. Conf. on Computer Vision*, 2009.

[28] A. Toshev, P. Mordohai, and B. Taskar, "Detecting and parsing architecture at city scale from range data," in *Intl. Conf. on Computer Vision and Pattern Recognition*, 2010.

[29] B. Stilman, *Linguistic Geometry: From Search to Construction*. Kluwer Academic Publishers, 2000.

[30] L. Jones, A. Howden, M. Knighton, A. Sims, D. Kittinger, and R. Hollander, "Robot computer chess game," Aug. 1983, US Patent 4,398,720.

[31] D. Urting and Y. Berbers, "Marineblue: A low-cost chess robot," *Robotics and Applications*, pp. 76–81, June 2003.

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

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

[34] R. Hyatt, "CRAFTY–Chess Program," *ftp://ftp.cis.uab.edu/pub/hyatt*, 1996.

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

[36] Y. Nakamura and H. Hanafusa, "Inverse kinematics solutions with singularity robustness for robot manipulator control," *Journal of Dynamic Systems, Measurement, and Control*, no. 108, pp. 163–171, 1986.