A Task Grammar Approach to the Structure and Analysis of Robot Programs

R.Vijaykumar[†], S.Venkataraman[†], G.Dakin[†]
D.M.Lyons[‡]

COINS Technical Report 87-67

July 1987

[†]*Laboratory for Perceptual Robotics*
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

[‡]*Robotics and Flexible Automation Dept.*
Philips Laboratories
345 Scarborough Rd.
Briarcliff Manor, NY 10510

ABSTRACT

This paper describes a formal representation for the structure and context of a robot task. The principal objective of the representation is to allow the analysis of robot task descriptions for correctness. The importance of having an explicit representation of a task and its structure in addition to an object representation is emphasized. A formal-languages approach is used to express task structure. Although an informal semantics of task representation is used in this paper, we discuss the use of a formal semantics in the final section.

A version of this paper is to be published in the proceedings of the IEEE Workshop on Languages for Automation, Vienna, Austria, August 25-26th, 1987.

A Task Grammar Approach to the Structure and Analysis of Robot Programs

R.Vijaykumar[†], S.Venkataraman[†], G.Dakin[†], and D.M.Lyons[‡]

[†]*Laboratory for Perceptual Robotics*
Dept. of Computer and Information Science
Univ. of Massachusetts
Amherst, MA 01003

[‡]*Robotics and Flexible Automation Dept.*
Philips Laboratories
345 Scarborough Rd.
Briarcliff Manor, NY 10510

1. Introduction

Typically, robot programming languages are divided into *robot-level* languages, where all actions are defined in terms of manipulator movements, and *task-level* languages, where all actions are described in terms of object relationships. We argue in this paper that there is a useful third class of languages: task-level languages which have an explicit representation for the *operational* aspects of a task. Informally, a task can be defined as *a well-defined domain of interaction between the robot and the environment.* Associated with it are a *purpose*, a *method*, and a *specification of when the method remains applicable.* For example, if tracing a surface is the purpose of a task, compliant motion could be its method and contact with the surface would be the applicability condition. We define the *context of a task* to be its method along with its applicability condition. Methods of complex tasks are obtained as compositions of methods of simpler tasks, just as programs are obtained as compositions of statements, conditionals, loops, etc. The structure of a method constitutes the *structure* of the corresponding task.

In this paper we construct a formal definition for tasks and suggest a representation for the structure of tasks. The basic composition operations we introduce to express complex tasks in terms of simpler tasks are sequencing, parallel execution and conditional execution. In each case, we are interested in determining if the task will succeed or where it can fail. We use context-free grammars to represent task structure and provide operational semantics for this representation. Throughout this work the semantics of task structures is given informally. However, a formal semantics could be defined based on Lyons' $\mathcal{RS}$ (Robot

Preparation of this paper was supported in part by grant numbers DMC-8511959 and IST-8513989 from NSF (M.A.Arbib and K.Ramamritham, Principal Investigators).

Schemas) [3] model of computation. This issue is discussed in the final section.

An overview of previous work in this area sets the scene for our definition of a task. We develop a representation for task structure based on formal grammars and illustrate this development with a simple but useful example. This representation is then augmented to deal with parallel actions and conditional actions. Finally, the way in which this task structure representation suggests a hierarchical decomposition of tasks is discussed.

2. Motivation

Robot-Level programming systems are computer languages with predefined routines to control the robot actuators and sensors *explicitly*, while *task-Level* programming systems are specially defined programming languages in which actions specified *implicitly* through desired object relationships. The motivation for constructing a task-level interface to a robot is simply that it is easier for humans to specify programs in terms of the desired relationships between objects, than in terms of manipulator movements. However, goal object relationships are only part of the information a human can readily provide. A particular goal object relationship may be established by more than one set of actions. Frequently, the actions (i.e. operations) by which the goal relationship is achieved are very important, especially in fine motions. For example, the goal relationships for the insertion of a peg into a hole and placing one block on another are very similar. The *control strategy* adopted for insertion, however, is quite different from that for simple placement. The value of providing such operation information was recognized early in the development of task-level languages[2].

Our contention is that while goal object relationships by themselves do not provide adequate information for selecting an appropriate control strategy, a combination of object and operation information is adequate for this purpose. This provides the motivation behind our view of tasks as comprising of a purpose (goal object relationships), a method (control strategy) and its applicability condition.

Fine motion strategies [4] are a good example of what we would call a task control strategy and they provide a clear operational meaning (our example of insertion versus placing). In addition, they have a good formal representation. However, there is no formal representation for the way in which fine motions and free motions must be mixed, perhaps hierarchically, in a complex robot program. This is the main problem we attempt to address using our formalism for task structure. The following section introduces a formal definition of tasks and our representation for task structure.

3. The Definition of A Task

We start by reiterating what we mean by the word "task." A *task* defines a well-defined domain of interaction between the robot and the environment. A task description contains a specific *goal* condition, a *control strategy* to achieve this goal, and an *applicability condition* specifying whether or not the control strategy remains appropriate for achieving the goal. The control strategy represents one possible way to achieve this goal, and is what is primarily characteristic about any given task.

The task control strategy is a concise description of the relationship between sensing and action in a task. Since committment to a specific task control strategy eliminates possible actions and interpretations of sensory data, it is very important to ensure that the applicability condition holds throughout the execution of the control strategy. To reflect the need for continuous monitoring of this applicability condition we term this a *task invariant condition.* For example, if during an insertion task, the gripper drops the peg, then the insertion task is no longer applicable; i.e., the task invariant has become false. We now define the word *task* more formally and provide operational semantics for the components of a task:

Definition 1 *A task T is a triplet (G_T, C_T, I_T), where*

- *G_T represents the goal geometry the task is to achieve.*
- *C_T represents the control strategy of the task: a filtering of sensory input and a motor vocabulary.*
- *I_T represents the task invariant, a predicate which can be applied to the world to determine if task T is still relevant to the achievement of its goal G_T.*

Definition 2 *C_T is considered a servoprocess which operates continuously until either the logical condition G_T becomes true, in which case it is considered to terminate successfully, or until the logical condition I_T becomes false, in which case it is considered to have terminated unsuccessfully.*

On the basis of this semantics, we shall write I_T and G_T as logical conditions and C_T as a set of constraint equations in our examples. To emphasize that C_T is semantically a servoprocess, we will enclose the constraint equations in angle-brackets.

As an example of this task definition, consider a guarded move. Call the robot end-effector E and the desired contact surface C. The goal of the task is to have E *against* C, where *against* represents the geometric relationship of the end-effector in contact with the desired surface. We write this as:

$$G_T(C) : E \ \mathit{against} \ C$$

and read this as "the goal component of task T, parameterized by C, is the condition that E be against the surface given by parameter C."

The control strategy for this task is a compliant move expressed as a constraint equation [1] of the form $u = q(k\Delta(r) - f)$. The equation expresses the relationship between control signal u that drives the end effector towards C, and the difference between the sensed force f and the *compliance* force $k\Delta(r)$ caused by the error in end-effector position $\Delta(r)$. q and k are constant terms. We write this as:

$$C_T :< \ u = q(k\Delta(r) - f) \ >$$

and read this as "the control strategy component of task T is a servoprocess which continually enforces the constraint equation that the control signals that move the end-effector be given by $q(k\Delta(r) - f)$."

The invariant for this task has the responsibility of ensuring that C_T remains applicable in achieving the goal G_T. The implication here is that there exists a *path* from the end effector's present location, say r, to the contact surface C. The invariant serves to monitor the applicability of the control strategy. Depending on the sophistication of the implementation this may be quite complex, thus making the task execution proportionately more robust. For illustrative purposes, we choose to specify the invariant as saying that there exists a collision free path from r to the contact surface C. We write this as:

$$I_T : \exists \ a \ path \ from \ r \ to \ C$$

and read this as "the invariant for task T is that there exists a collision free path from the present location of the end-effector to the contact surface C"

[1]You could choose a different constraint equation to express the control strategy, and that would make it a different task.

3.1 Representation of Task Structure

The example above describes a simple task in that the control strategy is expressible as a simple constraint equation between sensed variables and controlled variables. One of the main objectives of this paper is to propose basic composition operators by which such simple (and complex) control strategies may be combined to produce control strategies of complex tasks and to suggest a suitable formal representation for the structure of such complex tasks. Let us consider a complex task T whose control strategy is the ***sequential*** composition of control strategies of simple tasks. The control strategy of this compound task can be defined as a ***context-free grammar***[5]. The language $trace_T$, defined in relation to this grammar will, in a sense to be defined, delimit all possible ways that this task can execute.

Our primary motivation for choosing context-free grammars as our representational mechanism for task structures are that: (a) sequentiality is readily represented; (b) the use of non-terminals provides a representation of grouping and abstraction; and (c) context-free grammars are well-understood as a formalism and therefore such a represenation lends itself to further analysis. The definition of a context-free grammar is:

Definition 3 *A context-free grammar G is (Σ, N, P, S), where:*

- *Σ is a finite set of terminal symbols (the alphabet),*
- *N is a finite set of non-terminal symbols,*
- *P is a finite set of productions of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$.*
- *$S \in N$ is the start symbol.*

$L(G)$, the language defined by G, is the subset of Σ^* each element of which is obtainable by repeated applications of productions in P to the start symbol S.

Our alphabet will be a set of primitive ***sensory predicates*** and ***motor actions***. Sensory predicates will include task invariants and goals. Motor actions will include the control strategy components of tasks. For example, if the control strategy of task T, C_T, occurs in some string, we consider that to mean that the servoprocess which obeys the constraint equations specified by C_T is started.

Now that we have introduced the notion of formal languages, we can provide a succinct reformulation of the operational semantics of Definition 2 of simple tasks, by defining a set $trace_t$ of possible outcomes of a task t.

Definition 4 *For a simple task* $t = (G_t, C_t, I_t)$, $trace_t$ *is defined as the language over the alphabet* $\{G_t, C_t, I_t, \neg\}$ *given by* $trace_t = \{C_t, C_tG_t, C_t\neg I_t\}$.

We define a *task grammar* as a formal language representation of the control strategy component of a task.

Definition 5 *A task grammar is a context-free grammar (Definition 3) in which the alphabet is composed of the goal condition, control strategy and invariant condition of some number of subtasks.*

We now have to extend our semantics of Definition 2 to tell us how to execute a control strategy which is defined as a task grammar. We provide an informal description below: If C_T is defined by a grammar T_g, the productions are applied to the start symbol in a left-most derivation (i.e., the left-most non-terminal is rewritten first) until a string is derived which contains a terminal control strategy symbol C_t. This control strategy is executed, and on successful termination of C_t, the suffix of C_t is processed left to right; if a terminal control strategy appears before a non-terminal, the control strategy is executed and the procedure is repeated with its suffix; if a non-terminal is found then left-most derivation is applied until a terminal control strategy appears. This process is repeated until a string of terminals comprising only sensory predicates is left. This informal description is formalized by defining the set $trace_T$ of possible execution traces of a complex task T whose structure is given by the grammar T_g.

Definition 6 *Let* T *denote a complex task whose control structure is given by the task grammar* T_g, *where* $T_g = (\Sigma_T, N_T, P_T, S_T)$. *Let* $\circ$ *denote the operation of concatenation of languages. For any string* α *of terminals and non-terminals of the grammar* T_g, *define the set* $trace_\alpha$ *recursively as follows:*

1. *if* $\alpha = w$, *where* w *is a string of terminal symbols consisting purely of sensory predicates,* $trace_\alpha = \{w\}$.

2. *if* $\alpha = wC_t\beta$, *where* w *is a string of terminal symbols consisting purely of sensory predicates, and* C_t *is a terminal control strategy, then* $trace_\alpha = (\{w\} \circ \{C_t, C_t\neg I_t\}) \cup (\{wC_tG_t\} \circ trace_\beta)$.

3. if $\alpha = wA\gamma$, where w is a string of terminal symbols consisting purely of sensory predicates, A is a non-terminal, and the set of producions in P_T that rewrite A are given by
$A \rightarrow \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$, then $trace_\alpha = \{w\} \circ (trace_{\beta_1} \cup trace_{\beta_2} \cdots \cup trace_{\beta_n}) \circ trace_\gamma$.

$trace_T$ is then defined as $trace_T = trace_{S_T}$. $trace_T$ is thus a language over $\Sigma_T \cup \{\neg\}$.

Definitions 4 and 6 describe the computational procedures by which the language $trace_T$ can be derived given the grammar corresponding to task T.

Two previous uses of formal languages in robotics are due to Albus[1] and Saridis[6]. Both of these have been aimed at structuring and programming hierarchical control systems. Our use of formal languages is to express in a concise form what possible behaviors can result from a given task. We shall, however, take advantage of their work in decomposition to examine how abstract tasks can be rephrased at less abstract levels. Both Saridis and Albus use formal languages as a useful programming tool, whereas our use of formal languages is primarily analytic.

3.2 Example

We shall explore the use of a grammar to represent a complex control strategy with an illustrative example (figure 1). Suppose task T is composed of three sequential motion control strategies for a robot end-effector — P: a position controlled move to some location (x, y, z); followed by Q: a compliant move in the z direction until contact is made with a surface; followed by R: a compliant move in the x direction until contact is established with a surface during which contact is maintained in the z direction. We will describe the components of these tasks, and then formulate the control strategy of the compound task, T.

$P = (G_P, C_P, I_P)$ is the positioning task. Let r denote the position of the end-effector; $r_c = (x_c, y_c, z_c)$ denotes the current position and $r_g = (x_g, y_g, z_g)$, the goal position. Let $PC(\Delta r)$ denotes a *position controller* based on the error in r.

$$\begin{array}{ll} G_P(x,y,z) & : r_c = r_g \\ C_P & :< u = PC(\Delta r) > \\ I_P & : (f_x = 0)\ AND\ (f_y = 0)\ AND\ (f_z = 0) \end{array}$$

The control strategy component here is a servoprocess which calculates control signals u to move the end effector based on the error between the desired and the current position.

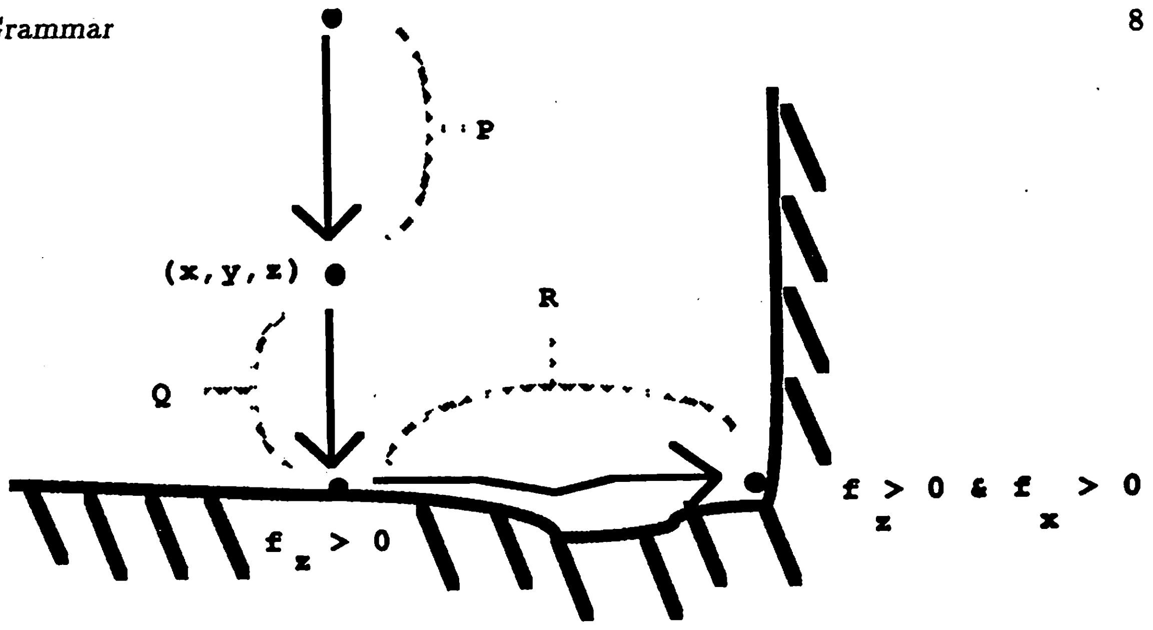

Figure 1: A Simple Aggregate Control Strategy

$Q = (G_Q, C_Q, I_Q)$ is the compliant move, along the z axis, to a surface. The terminology is identical to the example in Section 3 except that all forces are subscripted with z to denote they are forces along the z-axis (i.e., E is the end-effector, u_z is the z direction control signal, f_z is sensed z force, and k_z, q_z are constants).

$$\begin{array}{ll} G_Q(C) & : E \textit{ against } C \\ C_Q & :< \ u_z = q_z(k_z\Delta(z) - f_z) \ > \\ I_Q & : (f_x = 0) \ AND \ (f_y = 0) \end{array}$$

Note that contact (in the z direction) with the wall is represented through the goal component and how we achieve contact in this direction is represented through the control strategy. Note also that the difference between the invariance for P and Q is that Q has no invariance specification in the z direction because that is the direction in which we perform the compliant move through Q's control strategy.

$R = (G_R, C_R, I_R)$ is the final compliant motion (representing a sliding action) and is the most complex task so far. Let D be the upright surface.

$$\begin{array}{ll} G_R(C, D) & : (E \textit{ against } C) \ AND \ (E \textit{ against } D) \\ C_R & :< \ u_z = q_z(k_z\Delta(z) - f_z), \ u_x = q_x(k_x\Delta(x) - f_x) \ > \\ I_R & : f_y = 0 \end{array}$$

Now how are these three strategies aggregated to represent T? The goal component of task T is given as

$$G_T(C,D) : (E\ against\ C)\ AND\ (E\ against\ D)$$

The task invariant component for T would be that the sensed force in the y direction is zero throughout the sequence of motions representing T.

The compound task T has a control strategy component defined by the grammar T_g given below:

$$\begin{aligned} T_g &= (\Sigma_T, N_T, P_T, S_T) \\ \Sigma_T &= \{C_P, C_Q, C_R, G_P, G_Q, G_R, I_P, I_Q, I_R\} \\ N_T &= \{t1, t2, t3\} \\ S_T &= \{t1\} \\ P_T &= \{t1 \rightarrow C_P t2 \\ & \quad\; t2 \rightarrow C_Q t3 \\ & \quad\; t3 \rightarrow C_R\} \end{aligned} \tag{1}$$

The alphabet of the grammar consists of the control strategies, goal assertions, and invariants of the three component tasks. The productions implement the sequencing of the three tasks. The language $trace_T$ obtained by Definition 6 corresponding to the task grammar T_g is given by

$$\begin{aligned} trace_T = \{ & C_P, C_P \neg I_P, \\ & C_P G_P C_Q, C_P G_P C_Q \neg I_Q, \\ & C_P G_P C_Q G_Q C_R, C_P G_P C_Q G_Q C_R \neg I_R, \\ & C_P G_P C_Q G_Q C_R G_R\}. \end{aligned} \tag{2}$$

$trace_T$ consists of all possible outcomes of the compound task T, including non-terminations and partial completions terminated by errors. The crucial point is that we now have a representation of the interactions of the three control strategies that is open to formal analysis.

3.3 Parallel Actions

A robot programming formalism should allow for representation of parallel actions in order to adequately express the versatility of robot actions. Production grammars such as the task grammar presented in this paper produce words comprising fixed sequences of terminal symbols. Unless constructs for concurrency are built within the task grammar, all sensory events and motor actions generated by the grammar will be processed and performed in a strictly sequential order. Although most tasks call for some degree of sequential ordering of their motor and sensory components, many perceptions and actions may, or should, be sensed or executed in parallel. For example, preshaping of the hand is performed in parallel with the motion of the arm.

To represent parallel actions, we extend our string notation. A string of symbols such as $S_1S_2S_3S_4S_5$ is called a sequential string of symbols. A string of symbols separated by commas and enclosed in paranthesis such as (S_1, S_2, S_3), is called a parallel string. Mixed sequential and parallel strings are allowed. We formalize the syntactic construction of mixed sequential and parallel strings in *Backus-Naur Form (BNF)* as given below[2]:

$$\begin{aligned} <mixed\_string> \; &::= \; <mixed\_string> <mixed\_string> | \\ & \quad\;\; <parallel\_string> | \; symbol \\ <parallel\_string> \; &::= \; (<list\_of\_mixed\_strings>) \\ <list\_of\_mixed\_strings> \; &::= \; <mixed\_string>, <list\_of\_mixed\_strings> | \\ & \quad\;\; <mixed\_string>, <mixed\_string> \end{aligned} \tag{3}$$

We are now able to express the control strategy of a compound task as sequential and parallel compositions of simpler tasks. For example, $S_1S_2(S_3, S_4)S_5S_6$ expresses the intent that operation S_2 follows S_1, which is then followed by operations S_3 and S_4 executed in parallel. On completion of S_3 and S_4, operations S_5 and S_6 are to be executed sequentially in that order. Based on this informal description of the execution semantics of the parallel construct, we define below a procedure to compute the execution traces of a parallel control strategy. The definition below is an addendum to Definition 6 to include parallel constructs.

Definition 7 *Let T denote a complex task whose control structure is given by the task grammar T_g, where $T_g = (\Sigma_T, N_T, P_T, S_T)$. Let $\circ$ denote the operation of concatenation of languages. Given a string s of symbols, let Π_s denote the set of all strings obtained by permuting the symbols in s (eg., If $s = abc$, then $\Pi_s = \{abc, acb, bac, bca, cab, cba\}$). For any string α of terminals and non-terminals of the grammar T_g,*

> *if $\alpha = w(\beta_1, \beta_2, \cdots, \beta_n)\gamma$, where w is a string of terminal symbols consisting purely of sensory predicates, then the set $trace_\alpha$ of possible execution traces of the control strategy given by α is defined as $trace_\alpha = \{w\} \circ (\bigcup_{s \in \delta} \Pi_s) \circ trace_\gamma$, where $\delta = trace_{\beta_1} \circ trace_{\beta_2} \circ \cdots \circ trace_{\beta_n}$.*

[2]Left and right parantheses and comma are now added to the set of terminal symbols to allow this syntactic construction for representing parallel strings

3.4 Conditional Actions

Another desirable feature in a notation for task control strategies is the ability to express conditional execution. A course of action may be appropriate only when certain conditions are detected in the environment. We use $\diamond$ to denote conditional execution. For example, $T_1 \rightarrow (\sigma \diamond T_0)$ is to be interpreted as: evaluate the condition expressed by σ; if σ evaluates to ***true***, then execute the control strategy of the task represented by T_0. Conditional execution constructs can be mixed with sequential and parallel constructs to express complex task control strategies. We extend the BNF description of mixed strings in equation 3 to reflect this.

$$\begin{aligned}
<\mathit{mixed\_string}> \; &::= \; <\mathit{mixed\_string}> <\mathit{mixed\_string}>| \\
&\quad\; <\mathit{parallel\_string}>| \; <\mathit{conditional\_string}>| \\
&\quad\; \mathit{symbol} \\
<\mathit{conditional\_string}> \; &::= \; (\mathit{predicate} \diamond <\mathit{mixed\_string}>)
\end{aligned}$$

Based on the description of execution semantics of conditional constructs given above, we provide an extension to Definition 6 to deal with task grammars that include conditional constructs. Like Definition 7, the following definition should also be treated as an addendum to Definition 6.

Definition 8 *Let T denote a complex task whose control structure is given by the task grammar T_g, where $T_g = (\Sigma_T, N_T, P_T, S_T)$. For any string α of terminals and non-terminals of the grammar T_g,*

> *if $\alpha = w(\sigma \diamond \beta)\gamma$, where w is a string of symbols consisting purely of sensory predicates, and σ denotes a condition, then the set $trace_\alpha$ of possible execution traces of the control strategy given by α is defined as $trace_\alpha = \{w\} \circ (\{\neg\sigma\} \cup (\{\sigma\} \circ trace_\beta)) \circ trace_\gamma$.*

3.5 Task Hierarchies

Our task definition does not help in decisions about what domains of interaction between objects should be embodied as tasks, and what such tasks form an atomic set. That is, they do not help in deciding how abstract tasks (high-level tasks) decompose into other tasks, or in deciding on a complete and robust set of primitive (lowest level) tasks. The question that immediately comes to mind is: when is a set of primitive tasks sound and complete with respect to assembly operations? A set of primitive tasks is *complete* if it can produce all the fine and free motions necessary for any assembly. A set of primitive tasks is *sound* if only valid free and fine motions can be produced by combining these tasks.

It is clear that it is hard to use completeness and soundness in any formal sense. However, we can use them in an informal sense. It is common to divide robot motion into *free motion* (motion in free space) and *fine motion* (motion which incurs reactive forces, including guarded moves). The first requirement that we wish to impose on the system is that if we construct a hierarchy of tasks, *each level of the hierarchy should have a natural concept of free and fine motion.* In this case, soundness presents a criterion for *the interactions between successive levels in the hierarchy.* Using this we can define an natural set of hierarchical levels, similar to [8].

At the lowest level fine and free motion are defined between the robot and *its* environment (this is the traditional domain in which to view fine and free motion). At the next level, fine and free motion are defined between an object and *its* environment (i.e., the robot is holding the object, its scope of attention is limited to just the object its holding). At the highest level of abstraction, fine and free motion are defined between arbitrary objects (i.e., the scope of attention is the whole assembly, and is independent of robot structure). We call these three levels of abstraction, the *Immediate, Tactical* and *Strategic* level, respectively. For example, a fine-motion of a tool grasped in a dextrous hand against an object in the workspace can be seen at a lower level as a fine-motion between the hand and the tool. The informal soundness constraint tells us that a fine motion between two objects *must transform into a set of fine motions between the hand and a single object.* It also tells us that there should be no loss of information in the interaction between various levels. For example, the task invariant associated with a task at one level should be inherited by its subtasks at lower levels.

One of the strengths of this approach to constructing a hierarchy is that the levels are clearly separate, and the transformations between levels are well-defined. Note that this hierarchy embodies the object-level, manipulator-level division discussed earlier, but has more generality since it can represent more than just geometric information. At each level of abstraction, the control strategy component of a task is modelled as a formal language.

4. Discussion

We have introduced a formal representation of a robot *task* and the *structure of a task* in order to express and analyze complex tasks. The representation allows us to analyze non-termination of a task in addition to other error cases. Our analysis is preliminary as yet on all these issues, but it shows much promise.

We feel that the research outlined in this paper motivates the development of task-level languages which have an explicit representation for a task description. There has been some research along exactly these lines in the literature, e.g. Taylor's procedure skeletons[2,7]. However, full use has not been made of the advantages of task context. In particular the literature is weak in the formal representation of task structure. One promising work from the task grammar perspective is Lyons' $\mathcal{RS}$ model [3]. This is a formal model of computation specifically designed for sensory-based robot control. $\mathcal{RS}$ provides a set of well-defined computational entities with which to structure robot task plans. In [3], a Temporal logic was used to analyze tasks for correctness and safety. A formal, $\mathcal{RS}$ semantics could be defined for task grammars in place of the trace semantics (Definition 4,6, and its extensions 7 and 8). This would have the advantages of making the task grammar concept more concise, and also providing a less complicated analysis tool for $\mathcal{RS}$ programs. However, the representational abilities of the two systems are yet to be compared.

Although this work is still in a preliminary stage, we feel the examples presented here show its promise. Much work remains to be done. Specifically, we are now looking into Recursive Transition Networks (RTNs) [5] and their extensions, Augmented Transition Networks [9] as alternate representation mechanisms for task structure. Although RTNs are equivalent to context-free grammars, the notion of task hierarchies and nesting of task grammars suggest RTNs as a more natural representation. Augmented Transition Networks have the additional feature that choice of rewrite rules can be based on evaluation of specified conditions, which is appealing given that we have conditional constructs. However, determining the viability of using these representations need further consideration.

Acknowledgements.

The authors would like to thank Michael A. Arbib of USC, and Roy Featherstone of Philips Laboratories for many challenging comments and suggestions on the task grammar concept.

REFERENCES

[1] Albus, J., MacLean, C., Barbera, A., and Fitzgerald, M., "Hierarchical Control for Robots in an Automated Factory," *Proceedings, 13th ISIR*, Chicago, Ill., Apr., 1983, pp.13.29–13.43.

[2] Lozano-Perez, T., and Brooks, R., "An Approach to Automatic Robot Programming," *AI Memo 842*, MIT, Cambridge, MA, April, 1985.

[3] Lyons, D.M., "$\mathcal{RS}$: A Formal Model of Distributed Computation for Sensory-Based Robot Control" *Ph.D. Dissertation, COINS Technical Report* # 86-43, University of Massachusetts at Amherst, Amherst, MA 01003, 1986.

[4] Mason, M., "Compliance and Force Control for Computer Controlled Manipulators" *IEEE Trans. SMC* SMC-11, No.6, June 1981, pp418-432.

[5] Moll, R., Arbib, M., Kfoury, A., *An Introduction to Formal Language Theory* Springer-Verlag 1987.

[6] Saridis, G.N., "Intelligent Robotic Control" *IEEE Trans. on Automatic Control* Vol AC-28, No.5, May 1983, pp547-557.

[7] Taylor, R., "A Synthesis of Manipulator Control Programs from Task-Level Specifications," *Technical Report* STAN-CS-76-560, Department of Computer Science, Stanford University, Stanford, CA, Jul., 1976.

[8] Vijaykumar, R., and Arbib, M.A., "Problem Decomposition for Assembly Planning" *Proceedings IEEE R&A*, Raleigh, North Carolina, 1987.

[9] Woods, W.A., "Transition Network Grammars for Natural Language Analysis" *Communications of the ACM*, Vol. 13, No. 10, October 1970, pp591-606.