On the Knowledge Requirements of Tasks

Ronen I. Brafman
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada V6T 1Z4
brafman@cs.ubc.ca

Joseph Y. Halpern
Computer Science Department
Cornell University
Ithaca, NY 14853-7501
halpern@cs.cornell.edu

Yoav Shoham
Computer Science Department
Stanford University
Stanford, CA 94305
shoham@cs.stanford.edu

Abstract

In order to successfully perform a task, a situated system requires some information about its domain. If we can understand what information the system requires, we may be able to equip it with more suitable sensors or make better use of the information available to it. These considerations have motivated roboticists to examine the issue of sensor design, and in particular, the minimal information required to perform a task. We show here that reasoning in terms of what the robot knows and needs to know to perform a task is a useful approach for analyzing these issues. We extend the formal framework for reasoning about knowledge, already used in AI and distributed computing, by developing a set of basic concepts and tools for modeling and analyzing the knowledge requirements of tasks. We investigate properties of the resulting framework, and show how it can be applied to robotics tasks.

1 Introduction

The notion of computational complexity has had a profound effect on the development of computer science. While imperfect, our ability to classify different computational problems in terms of their complexity allows us to understand inherent difficulties in solving such problems. Thus, when a problem can be solved or approximately solved in polynomial time, we can concentrate on improving algorithms for its solution. Conversely, when a problem is shown to be a member of (what is believed to be) a more difficult class such as the class of NP-complete problems, we know that we must look for heuristics and simplifying assumptions when confronting this problem.

Some areas of robotics have benefited from advances in computational complexity. This is true primarily of certain stylized robotics problems, such as variants of robot motion-planning (e.g., [Can89, RS85, Rei79]). However, the area of robotics as a whole still lacks the analog of a Turing machine, a formal device that faithfully quantifies the difficulty of a robotic task or the capabilities of a robot.[1] The reason for this is that usually space and time complexity are not the dominating factors in a robotic task. Rather, issues such as the sloppiness of controllers, the imprecision of sensors, and the need for communication between spatially separated components assume major importance. This suggests that a good model for robotics should revolve around the notions of information and uncertainty. Similar points have also been made by Erdmann [Erd94] and Donald [Don94].

We propose a formal framework to capture these notions, closely based on that of [FHMV95], which makes use of a formal notion of *knowledge*. We believe that reasoning in terms of knowledge can form the basis for a general model of informational aspects of robots and robotic tasks. In our framework, robotic tasks can be characterized in terms of the knowledge required to perform them, and robots can be characterized in terms of the knowledge they can acquire. We can therefore assess the ability of a particular robot to perform a task by comparing its knowledge capabilities to the knowledge requirements of the task. This is reminiscent of the use of knowledge in distributed systems to characterize the information needed to perform tasks such as coordinated attack [HM90].

In the coordinated attack problem and other problems of coordination and agreement, it turns out that *common knowledge*—the state where everyone knows that everyone knows that everyone knows ...—plays a crucial role. It is, in a precise sense, a necessary and sufficient condition for coordination and agreement [FHMV95, HM90]. Moreover, the knowledge before common knowledge is attained is irrelevant; all that matters is that common knowledge is eventually attained. In a certain important class of tasks that we consider here, which we refer to as *manipulation tasks*, we can say even more. The goal in a manipulation task is to move an object from some initial configuration to a goal configuration. These are the types of tasks discussed in the motion-planning literature [Erd94, Lat91]. In a manipulation task, we can typically find a set of propositional formulas such that, if the agent knows one of these formulas at every step, then the task can be performed and, moreover, (if the agent has appropriate sensors) it is possible for the agent always to know one of these conditions. Intuitively, each of these tests identifies a set of configurations for which a particular transition exists which would reduce the distance, according to some distance measure, of the system's configuration from the goal. This is essentially the approach taken by Erdmann [Erd94]. As we shall see, thinking in terms of knowledge gives us a high-level tool to clarify what is going on. We illustrate this point by applying our ideas to a maze-searching example originally analyzed by Blum and Kozen [BK78]; see Section 4.

To provide intuition, throughout this paper we will anchor the formal development in the following example. Although simple, the example embodies two important ingredients—imprecise sensing, and the need to coordinate the actions of spatially distributed actuators.

Example 1.1: Two horizontal, perpendicular, one-dimensional robotic arms must coordinate as follows. The first arm must push a hot object lengthwise across the table until the

[1] This observation was made by John Mitchell.

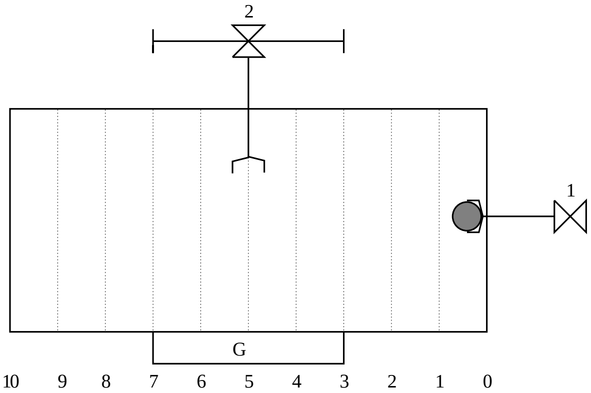

Figure 1: The two-arm system

second arm is able to push it sideways so that it falls into a cooling bin. The length of the table is marked in feet, from 0 through 10 (for simplicity we ignore the vertical coordinate). The object is initially placed at position 0 on the table. The second arm is able to push the object if it is anywhere in the region [3,7].[2] The second arm cannot hit the object while it is pushed by the first arm, since this will cause the mechanism to jam; on the other hand, the object cannot remain motionless for more than an instant or it will burn a hole into the table. Thus, the second arm must move precisely when the first one stops. This setup is illustrated in Figure 1. We consider two variants of the problem:

1.1a. The arms share a controller. The controller has access to a sensor reporting the position of the object with error no greater than 1, i.e., if the object's current location is q then the reading can be anywhere in $[q-1, q+1]$.

1.1b. Same as 1.1a, except the error bound is 4 rather than 1.

It is not hard to see that in case 1.1b, there is no protocol that performs the task, whereas in cases 1.1a, there is. For example, a centralized protocol that deals with 1.1a is the following (where r is the current reading):

if $r \leq 4$ **then** Move(arm$_1$) **else** Move(arm$_2$).

∎

Example 1.1 illustrates the need to analyze several basic issues, such as how much information the controller needs in order to perform this task and what information each controller is capable of obtaining. These are the types of issues we consider in this paper.

This example should make apparent that, unlike the planning perspective taken by the work of Moore [Moo85] and Morgenstern [?] on knowledge, actions, and plans, we take a design perspective. It is not our goal to provide knowledge representation tools for an agent that reasons about its knowledge during the course of its planning activities. Rather, we provide a set of concepts that can aid in the process of designing a situated system

[2]We use the $[a, b]$ notation to denote the interval of natural numbers between a and b, including a and b. Thus, $[3, 7] = \{3, 4, 5, 6, 7\}$.

that can perform some given set of tasks. To use the analogy of computational complexity, we are not considering the task of building agents that must figure out how to solve a particular computational problem; rather, we attempt to provide tools by which a designer could characterize the resources needed by a program that solves this problem. Although both problems are related at some abstract level, different models, assumptions, concepts, and languages are appropriate in each case. We shall have more to say on this issue in our discussion of related work (Section 5).

The rest of this paper is organized as follows: In the next section, we describe the view we take in modeling tasks, agents, and their information requirements. In Section 3, we use the concept of knowledge to define measures of information requirements of tasks and information capabilities of agents, and we show some relations that exist among these measures. In Section 4, we continue with this development, supplying a number of additional tools, such as control variables and learning. We illustrate these tools with the problem of maze searching. We discuss related work in Section 5, and conclude in Section 6 with some directions for further work.

2 The Basic Model

In this section, we describe a basic model of an agent embedded in an environment in which it must act. We start with an overview of our perspective and our aims, an understanding of which will help the reader understand the development of this paper and our technical choices in the rest of this paper. We then formalize these ideas.

2.1 An Overview of our Approach

To investigate issues such as the information complexity of tasks and information-attaining capabilities of agents, we must first make the notion of a task more precise. Tasks are defined in some context; for example, the task of getting a robot from point A to point B is defined in the context of some physical environment in which the robot's motions take place; the task of rearranging the furniture in a room is defined in the context of some room description, some description of the furniture, their initial positions, and their desired final positions. More abstractly, a task is defined in the context of some set of possible configurations of the system of interest, called its *configuration space*. A typical task might involve taking a system from some initial configuration to some goal configuration, while making sure that the system's configuration always satisfies certain conditions. Such a task can be described abstractly in terms of sets of acceptable sequences of configurations, or in the continuous case, acceptable functions from $[0, \infty)$ to configurations. For example, the task of getting from an initial configuration c_0 to some goal configuration c_f can be defined as the set of sequences of configurations in which c_0 appears first and c_f appears last. Or, using infinite sequences (as we do in this paper), this task corresponds to the set of sequences which start with c_0 and stabilize at c_f from some point on.

So far, we have said nothing about how the task is to be performed, that is, how we get from the initial to the final configuration. We abstract away from this issue here, and simply assume that there is a fixed set of changes, or *transitions*, that an agent can effect. Our goal is to understand what information an agent capable of these transitions needs in order to perform its task. This, in turn, affects the design of the agent's information-gathering capabilities, such as its sensors and communication channels.

To summarize, we are given (1) a set of possible configurations for a system, (2) a set of sequences of configurations defining the task, and (3) a set of allowed transitions defining the changes that can be made to the system's configuration at each point in time. We must supply the agent with the information necessary to perform its task, and a program that uses this information appropriately. As we shall see, using a formal notion of knowledge, we can analyze the information needs of a task, and provide guidelines for the design of the agent's sensory apparatus at an abstract, yet useful level.

The view developed here was strongly influenced by three sources. Erdmann's discussion of abstract sensors [Erd94], which explicitly examines the issue of sensor design, led us to consider many of the issues discussed in this paper and motivated our choice of semantics. Donald's work on information invariants [Don94], which provides a framework for comparing and evaluating sensor systems, led us to examine the ideas of task and sensor complexity. Finally, the framework for knowledge in multi-agent systems developed in [FHMV95] provides a natural tool for capturing and formalizing these ideas, especially given its past use in establishing lower bounds on message transmission and other resources required for performing tasks in distributed systems (see, for example, [HM90, CM86]). Indeed, Erdmann's semantics of abstract sensors leads naturally to the concept of knowledge. Our major contribution is the formalization and further development of these ideas in the context of robotics. We discuss the connection between our work and these other papers in more detail in Section 5.

In the remainder of this section, we present enough background to make te technical development in the paper self contained.

2.2 The Model

Our formal model is based on the notion of *system*, as defined in [FHMV95], with modifications appropriate for our context. We start by defining the space in which agents act and a set of possible transitions on that space. We shall confine ourselves to discrete domains. While a knowledge-level analysis can be carried out in continuous domains, (e.g., see [BLMS]), the technical issues raised by continuous domains would needlessly complicate this exposition.

Definition 2.1: Let $\mathcal{E}$ be the *environment's* set of states, also referred to as the *configuration space*. The set Λ of *transitions* over $\mathcal{E}$ consists of functions from $\mathcal{E}$ to $2^{\mathcal{E}} \setminus \emptyset$. We assume that the identity mapping Id is contained in Λ.

A task is simply a set of a sequences of configurations, which we call *C-histories*. One can view the C-histories defining a task as the set of desirable behaviors.

Definition 2.2: A *C-history* C (over $\mathcal{E}$) is an infinite sequence of configurations in $\mathcal{E}$. We use $\mathsf{C}(n)$ $(n \geq 0)$ to denote the n^{th} element of this sequence. A *task* (over $\mathcal{E}$) is a set of C-histories (over $\mathcal{E}$).

An agent is defined in the context of a fixed environment $\mathcal{E}$ and a set Λ of possible transitions. The state of the environment, or the *configuration*, describes the state of the external world, i.e., all the relevant aspects of the world not belonging to the robot's internal state. This is the world which the agent is to manipulate. The agent itself has a set of local states and actions, where each action transforms the local state of the agent and the external state of the world. These actions need not be deterministic, and it is possible for an action to change only the local state of the agent, as, in fact, is the case for sensing actions. We make two requirements. The first is that the effects of an action on the environment depend only on the current state of the environment. This guarantees that the local state of the agent represents only its internal state. The second is that the effects an action can have on the environment conform to the set of possible transitions. Hence, agents defined have the same abilities to transform the state of the external world (environment); they differ only in the structure of their local states (that is, in the information that they have) and in the effect of actions on these local states. Although the actions "implement" a fixed set of transitions, actions implementing the same transition may have different effects on each agent's local state. We note that although we often refer to an agent as a "robot," there is no requirement that its sensors and effectors make for a contiguous piece of equipment or that they are otherwise related to one another.

As the discussion above suggests, we assume that agents have *local states*. The *global state* of the system is a pair consisting of the configuration and the agent's local state.

Definition 2.3: If $\mathcal{E}$ is the set of configurations and L is the set of local states, then $\mathcal{G} = \mathcal{E} \times L$ is the set of *global states* (based on $\mathcal{E}$ and L). The projections of a global state $g = (c, l) \in \mathcal{G}$ to $\mathcal{E}$ and L are defined as $\mathsf{proj}_{config}((c,l)) \stackrel{\text{def}}{=} c$ and $\mathsf{proj}_{local}((c,l)) \stackrel{\text{def}}{=} l$. Projections of sets and sequences of global states are similarly defined, e.g., $\mathsf{proj}_x(S) \stackrel{\text{def}}{=} \cup_{s \in S} \mathsf{proj}_x(s)$.

Definition 2.4: An *agent $\mathcal{A}$ situated in* $(\mathcal{E}, \Lambda)$ is a pair $(L, \mathit{Actions})$, where L is the set of local states of the agent, and $\mathit{Actions}$ is a set of functions from $\mathcal{G} = \mathcal{E} \times L$ to $2^{\mathcal{G}} \setminus \emptyset$ satisfying the following conditions:

1. For all $a \in \mathit{Actions}$, $c \in \mathcal{E}$, and $l, l' \in L$, we have $\mathsf{proj}_{config}(a(c,l)) = \mathsf{proj}_{config}(a(c,l'))$.

2. For $a \in \mathit{Actions}$, let τ_a be the transition defined by $\tau_a(c) \stackrel{\text{def}}{=} \mathsf{proj}_{config}(a(c,l))$ for some $l \in L$.[3] Then for all $a \in \mathit{Actions}$, we must have $\tau_a \in \Lambda$. Moreover, for all $\tau \in \Lambda$, there exists some $a \in \mathit{Actions}$ such that $\tau = \tau_a$.

We call τ_a the transition *induced by $a \in \mathit{Actions}$*.

¿From now on, we assume we are working with a *fixed* configuration space $\mathcal{E}$ and set Λ of possible transitions. All agents discussed will be situated in $(\mathcal{E}, \Lambda)$. Therefore, all agents we discuss have the same physical capabilities but may differ in their information-attaining capabilities.

[3]The choice of the local state in the definition of τ_a is inconsequential because for all $l, l' \in L$ we require that $\mathsf{proj}_{config}(a(c,l)) = \mathsf{proj}_{config}(a(c,l'))$.

Example 2.5: The configuration space for example 1.1a consists of all possible positions of the hot object: $\mathcal{E} = [0,10] \times \{\mathit{Table}, \mathit{Bin}\}$.[4] The set Λ of possible transitions, consists of Move(arm_1), Move(arm_2), and the identity mapping, where Move(arm_1) transforms (q,x) to $(q+1,x)$ when $x = \mathit{Table}$ and $q \leq 9$, while Move(arm_2) transforms (q,x) to (q,Bin). Otherwise, these transitions do not change the configuration. We model the controller 1.1a as the agent $\mathcal{A}_{1a} = (L_{1a}, \mathit{Actions}_{1a})$, where $L_{1a} = [0,10]$, so that the controller's local state consists of its position reading, and $\mathit{Actions}_{1a} = \{\mathrm{Move}_1, \mathrm{Move}_2\}$, where

$$\mathrm{Move}_1((q,x),r) = \begin{cases} \{((q',x),r') : |q'-r'| \leq 1,\ q' = q+1 \text{ if } q \leq 9,\ q' = q \text{ if } q > 9\} & \text{if } x = \mathit{Table} \\ \{((q,x),r)\} & \text{if } x = \mathit{Bin} \end{cases}$$
$$\mathrm{Move}_2((q,x),r) = \{((q,\mathit{Bin}),r') : |q-r'| \leq 1\} \text{ (where } x \in \{\mathit{Table}, \mathit{Bin}\}).$$

Finally, let Task_{rob} consist of all trajectories that lead us from the initial configuration $(0, \mathit{Table})$ to one of the goal configurations $[3,7] \times \{\mathit{Bin}\}$. ∎

Global states provide us with an instantaneous description of a system. To characterize the system, we need to consider how the global state changes over time. The following definitions are taken from [FHMV95]

Definition 2.6: A *run* is a function r from N (the natural numbers) to the set of global states $\mathcal{G}$. A run r is *consistent* with respect to agent $\mathcal{A} = (L, \mathit{Actions})$ if for every $n \in N$, it is the case that $r(n+1) \in a(r(n))$ for some $a \in \mathit{Actions}$. A *system* is a set of runs.

According to this definition, we are identifying a system with its possible behaviors. Typically, systems are generated by protocols.

Definition 2.7: A *protocol* for an agent $\mathcal{A} = (L, \mathit{Actions})$ is a function $\mathcal{P} : L \rightarrow 2^{\mathit{Actions}} \setminus \emptyset$. A run r is an *execution* of a protocol $\mathcal{P}$ if for every $n \in N$ it is the case that $r(n+1) \in \mathcal{P}(\mathsf{proj}_{local}(r(n)))(r(n))$. If $I \subseteq \mathcal{G}$ is a set of (initial) global states, then the system $\mathcal{R}[I, \mathcal{A}]$ consists of every run r consistent with respect to $\mathcal{A}$ such that $r(0) \in I$. If $\mathcal{P}$ is a protocol for $\mathcal{A}$, then system $\mathcal{R}[I, \mathcal{A}, \mathcal{P}]$ consists of every execution r of $\mathcal{P}$ by $\mathcal{A}$ such that $r(0) \in I$.

A protocol describes the agent's program, allowing for non-deterministic behavior whenever more than one action is assigned at a local state. Its executions are the set of runs in which the agent's action at each point is consistent with the assignment of the protocol.

Finally, we say that an agent can perform a task if it has a protocol all of whose executions are in the task.

Definition 2.8: A protocol $\mathcal{P}$ for agent $\mathcal{A}$ *performs Task* from I if $\mathsf{proj}_{config}(\mathcal{R}[I, \mathcal{A}, \mathcal{P}]) \subseteq \mathit{Task}$. An agent *can perform Task* from I if it has a protocol that performs *Task* from I.

Example 2.9: Consider Example 1.1a, and let $I = \{((0, \mathit{Table}), 0)\}$. The system $\mathcal{I}[I, \mathcal{A}]$ consists of all runs starting in I in which the object is moved forward for some number of steps (possibly 0) and is eventually moved to the cooling bin. At all points, the local state indicates the current position with an error no greater than 1. In addition, this system

[4] Recall that $[0,10]$ denotes all natural numbers between 0 and 10.

contains all runs in which the object reaches position (10,Table) and remains there forever, with similar constraints on the local state.

Let protocol $\mathcal{P}$ assign the action Move$_1$ when the controller's local state is in $[0,3] \cup [7,10]$ and Move$_2$ when its local state is in $[4,6]$. The system $\mathcal{R}[I, \mathcal{A}, \mathcal{P}]$ consists of all runs in which the object moves forward until a reading in $[4,6]$ occurs for the *first* time. Given the above restrictions on sensing error, this could be anywhere in $[3,5]$. At this point the object is pushed to the cooling bin. Since all such runs are in $Task_{rob}$, $\mathcal{P}$ performs $Task_{rob}$. ▮

2.3 A Language for Reasoning about Knowledge

Having set up a model, we would like to have a formal language that will allow us to express properties of particular systems. *Epistemic logic*, first introduced by Hintikka [Hin62], provides a particularly suitable tool for this purpose. We start with set Φ of primitive propositions. We can think of these primitive propositions as statements like "the robot is at position 2" or "the temperature is high". The language $\mathcal{L}$ contains Φ, and is closed under the standard boolean connectives and the knowledge operator K. Thus, if α_1 and α_2 are formulas, then so are $\alpha_1 \wedge \alpha_2$, $\neg\alpha$, and $K\alpha$. We want to assign truth values to formulas in $\mathcal{L}$ at *points* in some system $\mathcal{R}$, where a point is a pair (r, m), consisting of a run r and a time m. To do this, we first need a way of deciding when the primitive propositions in Φ are true. Given a set $\mathcal{G}$ of global states, an *interpretation function* π *over* $\mathcal{G}$ assigns to each proposition $p \in \Phi$ a truth value at each global state in $\mathcal{G}$. A pair $\mathcal{I} = (\mathcal{R}, \pi)$ consisting of a system $\mathcal{R}$ of runs over set $\mathcal{G}$ and an interpretation π over $\mathcal{G}$ is called an *interpreted system*. In an interpreted system, we can define the semantics of propositional formulas in a straightforward way. Intuitively, a formula of the form $K\alpha$ is true at a point (r, m) if φ is true at all points (r', m') that the agent cannot distinguish from (r, m). An agent cannot distinguish two points if it has the same local state in both. These intuitions are formalized in the following definition (where $\mathcal{I}, r, m \models \alpha$ is read "α is true at the point (r, m) in the interpreted system $\mathcal{I}$"):

- $\mathcal{I}, r, m \models p$ for $p \in \Phi$ if $\pi(r(m), p) = \texttt{true}$;
- $\mathcal{I}, r, m \models \neg\alpha$ if $\mathcal{I}, r, m \not\models \alpha$;
- $\mathcal{I}, r, m \models \alpha \wedge \beta$ if $\mathcal{I}, r, m \models \alpha$ and $\mathcal{I}, r, m \models \beta$.
- $\mathcal{I}, r, m \models K\alpha$ if $\mathcal{I}, r', m' \models \alpha$ for all points (r', m') in $\mathcal{R}$ such that $\mathsf{proj}_{local}(r(m)) = \mathsf{proj}_{local}(r'(m'))$.

It is easy to check that whether $\mathcal{I}, r, m \models \alpha$ depends only on the global state $r(m)$; that is, if $r'(m') = r(m)$, then for all formulas α, we have $\mathcal{I}, r, m \models \alpha$ iff $\mathcal{I}, r', m' \models \alpha$. Thus, if s is a global state, we often abuse notation and write $\mathcal{I}, s \models \alpha$. (We remark that this would not be true if we used a richer language that included explicit temporal operators.) Moreover, whether a formula of the form $K\alpha$ is true depends only on the agent's local state. Thus, we further abuse notation and write $\mathcal{I}, l \models K_i\alpha$, if l is a local state. This notation emphasizes the fact that the knowledge operator allows us to express correlations between the local state of the agent and the external world. Finally, we assume for the rest of this paper that the interpretation function depends only on the configuration component of the global state. That is, for all propositions p, if the system's configuration in s and s' are the same, then $\pi(s,p) = \pi(s',p)$. This is reasonable for our intended applications, since tasks are defined in terms of the external world only. Thus, for a propositional formula α (one with no occurrences of the modal operator K), we often abuse notation and write $\mathcal{I}, c \models \alpha$, where c is a configuration.

For the remainder of the paper, fix an interpretation function π that depends only on the configuration. We use $\mathcal{I}[I,\mathcal{A}]$ and $\mathcal{I}[I,\mathcal{A},\mathcal{P}]$ to denote the interpreted system $(\mathcal{R}[I,\mathcal{A}],\pi)$ and $(\mathcal{R}[I,\mathcal{A},\mathcal{P}],\pi)$, respectively. We write $\mathcal{I} \models \alpha$ if $\mathcal{I}, r, m \models \alpha$ for every point (r,m) in $\mathcal{I}$.

3 Knowledge as an Analysis and Specification Tool

3.1 Skeletal Knowledge-Based Programs

Work in distributed systems has shown that the formal notion of knowledge is a powerful tool for analyzing traditional protocols. Knowledge is also useful for design purposes too; it allows one to design high-level protocols, called *knowledge-based programs* [FHMV95],[5] that focus on the informational aspects of a task without drowning in implementation details. Roughly speaking, knowledge-based programs describe what actions the agent should perform as a function of its knowledge. In this paper, we use a slightly more general notion, that of *skeletal knowledge-based programs* (*SKBP*s). An SKBP describes what transition an agent should bring about as a function of its knowledge. Hence, an SKBP can be used by different agents with different actions that implement similar sets of transitions. This is particularly useful in our context, where all agents considered implement a fixed set of transitions.

Definition 3.1: An SKBP is a set of pairs of the form $(K\alpha, \tau)$, where $K\alpha \in \mathcal{L}$ and $\tau \in \Lambda$.

An SKBP can be viewed as a big case statement of the form

case of
  if $K\alpha_1$ **then** τ_1;
  if $K\alpha_2$ **then** τ_2;
  ...
  if $K\alpha_n$ **then** τ_n;

Each condition of the case statement is a test on the knowledge of the agent. The interpretation of this protocol is that the agent non-deterministically performs an action that implements the transition corresponding to a condition that is satisfied.[6]

We refer to $K\alpha_1, \ldots, K\alpha_n$ as the (knowledge) *conditions* of this SKBP. If α_i is a propositional formula (i.e., it contains no occurrences of the K operator), we call $K\alpha_i$ *positive.* In the remainder of this paper, we restrict our attention to positive SKBPs. The fact that we do

[5]The similarity in names between knowledge-based program and knowledge-based (or expert) systems in AI is coincidental.

[6]Hence, unlike the `case` statement in certain programming languages, the order of appearence of the conditions does not matter.

not allow nested K's is not a serious restriction in the case of a single agent—every formula can be denested so that there are no nested K's However, the fact that we do not allow tests of the form $\neg K\alpha$, which means that an agent cannot perform an action based on lack of knowledge, is a nontrivial restriction in some applications. We return to this issue when we discuss learning in Section 4.2. Nevertheless, as we shall see, positive SKBPs still allow us to capture many intuitions of interest for our intended application. Moreover, the restriction to positivity makes it much easier to capture the notion of knowledge complexity, defined in Section 3.2.

Intuitively, a protocol $\mathcal{P}$ for agent $\mathcal{A}$ implements $\mathsf{Pg} = \{(K\alpha_i, \tau_i) \mid i = 1, \ldots, n\}$ if, at every local state l in which the agent's knowledge is $K\alpha_i$, $\mathcal{P}$ assigns an action a that implements τ_i. However, recall that an agent's knowledge in a local state is defined with respect to some system that determines its set of possible worlds. Hence, in order to determine which local states should be substituted for each knowledge condition, we must first specify the system with respect to which the agent's knowledge is defined. Formally, we adopt a semantics similar to that of [FHMV95] for knowledge-based programs, modified so as to handle our use of transitions rather than actions.

Definition 3.2: The *standard translation* of SKBP Pg with respect to interpreted system $\mathcal{I}$ and agent $\mathcal{A} = \langle L, \mathit{Actions}\rangle$ is the protocol $\mathsf{Pg}^{\mathcal{I}}$, where $\mathsf{Pg}^{\mathcal{I}}(l) \stackrel{\text{def}}{=} \{a_\tau \in \mathit{Actions} \ : \ \mathcal{I}, l \models K\alpha,\ (K\alpha, \tau) \in \mathsf{Pg},$ and a_τ implements $\tau\}$. If this latter set is empty, then $\mathsf{Pg}^{\mathcal{I}}(l) \stackrel{\text{def}}{=} \{a_{Id}\}$, where a_{Id} is the identity action, which maps a global state to itself. A protocol $\mathcal{P}$ for $\mathcal{A} = \langle L, \mathit{Actions}\rangle$ *implements* Pg from I if, for every $l \in L$ that occurs in $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$, we have that $\mathcal{P}(l) \subseteq \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}(l)$.

Example 3.3: Consider the following SKBP for the controller of Example 1.1a:

$$\mathsf{Pg}=\{\textbf{If } Kg \textbf{ then } \mathrm{Move}(\mathrm{arm}_2);\ \textbf{If } K\varphi \textbf{ then } \mathrm{Move}(\mathrm{arm}_1)\},$$

where φ holds when the object's position is in $[0,4] \cup [7,10]$ and g holds in the goal region. Consider the systems $\mathcal{I}[I, \mathcal{A}]$ and $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$ described in Example 2.9. In $\mathcal{I}[I, \mathcal{A}]$, the local states in which Kg holds are $[5,6]$, while in $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$, Kg holds in $[5,6,7]$. The local state in which the controller's position reading is 7 occurs within the runs of $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$ only when the actual position is 6. However, in the system $\mathcal{I}[I, \mathcal{A}]$, this local state can occur when the robot's position is anywhere in $[6,8]$.

The standard translation of Pg with respect to $\mathcal{I}[I, \mathcal{A}]$ is $\{([0,4] \cup [6,10], \mathrm{Move}_1), (\{5\}, \mathrm{Move}_2\})$. However, the standard translation of Pg with respect to $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$ is $\{([0,4], \mathrm{Move}_1), ([5,7], \mathrm{Move}_2)\}$. Notice that certain global states that occur in some runs of $\mathcal{I}[I, \mathcal{A}]$, such as $((7, \mathit{Table}), 8)$, do not occur in any run of $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$ ∎

Our notion of implementation leads to a natural notion of correctness: a an SKBP is correct if all protocols consistent with it satisfy the given task. This notion is referred to as *strong correctness* in [FHMV95]. Adapting their definitions to our presentation, we would say that $\mathcal{P}$ *represents* Pg if $\mathcal{P}(l) = \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}(l)$, and that $\mathcal{P}$ *is consistent with* Pg if $\mathcal{P}(l) \subseteq \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}(l)$. Notice that the set of protocols consistent with an SKBP Pg is (in general a strict) superset of the protocols that represent Pg. Moreover, a protocol consistent with a given SKBP is guaranteed to exist (see Lemma 7.1), although there may not be any protocol that represents it [FHMV95]. We have defined implementation in terms of consistency, rather than representation, because, in our context, strong correctness seems more appropriate than just requiring that all protocols that represent the SKBP satisfy the task. We are willing to accept a protocol as long as its behaviors are compatible with the SKBP, even if it does not generate all the behaviors of the SKBP. which are not represented by or consistent with any protocol.)

In this paper, we are interested in a particular class of implementations of skeletal knowledge-based programs.

Definition 3.4 : Protocol $\mathcal{P}$ for agent $\mathcal{A}$ is a *good* implementation of Pg from I if $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\alpha_1 \vee \cdots \vee K\alpha_n$, where $K\alpha_1, \ldots, K\alpha_n$ are the knowledge conditions of Pg.

By restricting our attention to the good implementations of SKBPs, we transform SKBPs from abstract program specifications to abstract knowledge specifications. Now, a skeletal knowledge-based program not only specifies what an agent should do, via the notions of standard translations and implementations, it also specifies a class of agents that are qualified to execute this specification. These agents have an implementation of the SKBP in which they always know enough so that one of the tests for knowledge holds at every global state. The importance of this property will become clearer when we present our definition of upper bound on the knowledge complexity of a task. From now on, unless otherwise noted, by an implementation of a SKBP Pg, we always mean a *good* implementation.

Note that an agent may have a good implementation of an SKBP with conditions $K\varphi_1, \ldots, K\varphi_k$ even though it can reach local states in $\mathcal{I}[I, \mathcal{A}]$ in which it knows none of the above conditions. If this happens, the SKBP must be preventing the agent from reaching such states. Intuitively, this can be due to one of two reasons: actions leading to states of relative ignorance are avoided and/or actions that lead to an increase of knowledge are taken.

The notion of *performs* can now be generalized to SKBPs.

Definition 3.5: A set I of global states (for agent $\mathcal{A}$) is t_0*-consistent with* task *Task* if $\mathsf{proj}_{config}(I) = \{\mathsf{C}(0) \mid \mathsf{C} \in \mathit{Task}\}$. Pg *performs Task* if all its good implementations from every set I t_0-consistent with *Task* perform *Task* from I, and it has a (good) implementation from some I t_0-consistent with *Task*.

Notice that, in defining the notion of an SKBP Pg performing *Task*, we restrict attention to sets I of initial states that are t_0-consistent with *Task*. Clearly, if we start Pg in an initial state that is not the initial state of some configuration in *Task*, it will generate an execution that is not in *Task*. Thus, we must restrict to initial states that are t_0-consistent with *Task* in order to get a reasonable notion of implementation.

In practice, it may be difficult to transform an SKBP to a standard protocol. However, we believe that, as has been the case in distributed systems, using knowledge-based analysis and design gives a useful methodology by allowing us to leverage the ability of SKBPs to abstract away the idiosyncrasies of local state. Thus, for example, rather than discuss the content of the frame buffer of a robot's vision system, an SKBP allows us to talk about the robot knowing that there is an obstacle in front of it.

3.2 Knowledge Complexity

We now wish to define a formal concept of informational complexity of a task that can serve to quantify the amount of knowledge an agent must attain in order to perform the task. Typically, the type of statement we want to make is that an agent must *eventually* come to know a certain fact (or one of a set of facts) in order to perform the task. For example, in [HM90], it was shown that to perform *coordinated attack*, the agents needed to eventually have common knowledge of the fact that at least one message was delivered.

Recall from the introduction that we are interested in *manipulation tasks*, where the goal is to move an object from some initial configuration to a goal configuration. We hope to find a set of propositional formulas such that can be thought of as describing sets of configurations such that, if the agent knows one of these formulas at every step, then the task can be performed by an SKBP that uses these tests. Intuitively, moving from one set of configurations to another gets the agent closer to its goal.

Keeping these intuitions in mind, we define a notion of informational upper bound appropriate for manipulation tasks.

Definition 3.6: If $\varphi_1, \ldots, \varphi_k$ are propositional formulas, we say that $\{\varphi_1, \ldots, \varphi_k\}$ is an *upper bound on the knowledge complexity of a task Task*, or just *Task* is $O(\{\varphi_1, \ldots, \varphi_k\})$, if there exists an SKBP Pg with conditions $K\varphi_1, \ldots, K\varphi_k$ that performs *Task*.

Notice that, in this definition, we are implicitly assuming that there is a fixed interpretation π on the configuration space $\mathcal{E}$, and we are restricting attention to interpreted systems $\mathcal{I}$ that use this interpretation. This definition should also make it clear why we are particularly interested in *good* implementations of a knowledge-based program. Suppose that we require *all* the implementations of an SKBP Pg (including the non-good ones) to perform *Task*. Unless some of the conditions in Pg are tautologies, there will be an agent that does not know any of the tests of Pg at any state. This agent will have an implementation $\mathcal{P}$ of Pg which has an execution in which it constantly performs the identity transition. Typically, the projection of this execution will not be in *Task*, and under the stronger definition, Pg would not perform *Task*. To avoid this problem, we restrict to *good* implementations. By doing so, we are, in fact, saying that an SKBP comes with some minimal requirements for its execution: the ability to know one of its knowledge conditions at each state.

Of course, if we are to use sets of (propositional) formulas as a measure of knowledge complexity, we must define an ordering on such sets, to allow us to say when the information characterized by one set is more difficult to attain than the information characterized by another set.

Definition 3.7: Given a configuration space $\mathcal{E}$, and sets A and B of propositional formulas, we say that B *dominates* A (with respect to $\mathcal{E}$), and write $A \preceq_{\mathcal{E}} B$, if for every formula $\psi \in B$, there is a formula $\varphi \in A$ such that $\mathcal{E} \models \psi \Rightarrow \varphi$.

Notice that if $A \preceq_{\mathcal{E}} B$, then for every formula $\psi \in B$, there is a formula $\varphi \in A$ such that knowing ψ implies knowing φ. It is easy to see that $\preceq_{\mathcal{E}}$ defines a partial order (that is, a reflexive, transitive relation) on sets of formulas. As the following result shows, this ordering does capture a reasonable notion of hardness.

Proposition 3.8: *If $A \preceq_{\mathcal{E}} B$, Task is $O(A)$, and $\mathcal{E} \models \vee_{\varphi\in B}\varphi$, then Task is $O(B)$.*

Proof: Intuitively, if *Task* is $O(A)$, there is an SKBP Pg with appropriate knowledge conditions that performs *Task*. Since $A \preceq_{\mathcal{E}} B$, for each condition $\varphi_i \in A$ in this SKBP there is a stronger condition $\psi_{f(i)} \in B$ such that ψ implies φ. As we show in the Appendix, the SKBP obtained by replacing each φ_i in Pg by $\psi_{f(i)}$ performs *Task*. Moreover, the requirement that $\mathcal{E} \models \vee_{\varphi\in B}\varphi$ suffices to ensure that there is a good implementation of some SKBP Pg that performs *Task*. Although this requirement can be weakened, something like it is necessary. For example, it is easy to see that $A \preceq_{\mathcal{E}}$ *false*, but there is no good implementation of a protocol whose only test is $K\mathit{false}$. See the Appendix for further details. ∎

Example 3.9: Consider Examples 1.1a and 1.1b again. Recall that in both variants a central controller is in charge of a two-armed robot, which must switch between horizontal and vertical motions when the object is in the goal region $[3,7] \times \{\mathit{Side}\}$. It is easy to see that Task_{rob} is $O(\{\neg g, g\})$: the $SKBP$ $\{(K\neg g, \text{Move}(\text{arm}_1)), (Kg, \text{Move}(\text{arm}_2))\}$, according to which the agent moves arm_1 if he knows $\neg g$ and moves arm_2 if he knows g, performs Task_{rob}. However, we can get a better bound. Let φ_1 denote being in $[0,4] \cup [6,10]$ and let φ_2 denote being in $[3,10]$. Clearly $\{\varphi_1, \varphi_2\} \preceq_{\mathcal{E}} \{\neg g, g\}$. Moreover, it is easy to verify that the following SKBP performs Task_{rob}: $\{(K\varphi_1, \text{Move}(\text{arm}_1)), (K\varphi_2, \text{Move}(\text{arm}_2))\}$. There is yet another upper bound for Task_{rob}, incomparable to the two we have just presented. Let φ_1' denote being in $[0,6]$. It is easy to see that $\{\varphi_1', \varphi_2\}$ is incomparable to $\{\varphi_1, \varphi_2\}$ and $\{g, \neg g\}$. Moreover, it is not hard to show that the $SKBP$ $\{(K\varphi_1', \text{Move}(\text{arm}_1)), (K\varphi_2, \text{Move}(\text{arm}_2))\}$ also performs Task_{rob}. For suppose that protocol $\mathcal{P}$ for agent $\mathcal{A}$ is a good implementation of this $SKBP$ from $\{(0, \mathit{Table})\}$. Let r be a run of $\mathcal{I} = \mathcal{I}[\{(0, \mathit{Table})\}, \mathcal{A}, \mathcal{P}]$. Consider the first time m that $\mathcal{I}, r, m \models K\varphi_2$. Notice that there must be such a time, since $\mathcal{A}$ performs $\text{Move}(\text{arm}_1)$ until $K\varphi_2$ holds. If $K\varphi_2$ never holds, then $\mathcal{A}$ must eventually reach position 7, at which point $K\varphi_1$ cannot hold, contradicting our assumption that $\mathcal{P}$ is a good implementation. Our argument also shows that at the point (r, m), the configuration must be in $[3,7]$. Hence, when the agent performs $\text{Move}(\text{arm}_2)$, it gets into the goal region. ∎

We can define a notion of lower bound that corresponds to our notion of upper bound.

Definition 3.10: We say that the set A of propositional formulas is a *lower bound* on the K-complexity of *Task*, or *Task* is $\Omega(A)$, if, for every set of formulas B such that *Task* is $O(B)$, we have that $A \preceq_{\mathcal{E}} B$.

Notice that $\{\mathit{true}\}$ is a lower bound for the K-complexity of any task. Obviously, this lower bound does not give much insight. Ideally, we would like a *tight* bound: a lower bound that is also an upper bound. Unfortunately, it seems difficult to get tight bounds. For example, we can show that Task_{rob} has no tight bound.

Proposition 3.11: *There is no set A of primitive propositions such that Task_{rob} is $\Omega(A)$ and $O(A)$.*

Proof: Suppose $Task_{rob}$ is $\Omega(A)$ and $O(A)$. Then $A \preceq_{\mathcal{E}} \{\varphi_1, \varphi_2\}$ and $A \preceq_{\mathcal{E}} \{\varphi'_1, \varphi_2\}$. Thus, A must contain formulas ψ_1 and ψ'_1 such that $\mathcal{E} \models \varphi_1 \Rightarrow \psi_1$ and $\mathcal{E} \models \varphi'_1 \Rightarrow \psi'_1$. Since $Task_{rob}$ is $O(A)$, there must be an $SKBP$ Pg that performs $Task_{rob}$ and has conditions $K\psi_1$ and $K\psi'_1$. Clearly, neither the pair $(K\psi_1, \mathrm{Move}(\mathrm{arm}_2))$ nor the pair $(K\psi'_1, \mathrm{Move}(\mathrm{arm}_2))$ can be in Pg. To see this, consider an agent that has perfect sensors, and knows exactly what location it is in. In particular, for this agent, both $K\psi_1$ and $K\psi'_1$ hold in the initial position, where the transition $\mathrm{Move}(\mathrm{arm}_2)$ is clearly inappropriate. Thus, both $(K\psi_1, \mathrm{Move}(\mathrm{arm}_1))$ and $(K\psi'_1, \mathrm{Move}(\mathrm{arm}_1))$ must be in Pg. But since $\varphi_1 \vee \varphi'_1$ holds in every location, so does $\psi_1 \vee \psi'_1$. Thus, the agent with perfect sensors always performs the transition $\mathrm{Move}(\mathrm{arm}_1)$ according to Pg, so Pg does not perform $Task_{rob}$. ∎

Although we cannot provide a tight bound for $Task_{rob}$, we might still hope to provide useful (nontrivial) lower bounds. While this can be done, we have found it more useful to use a different notion of lower bound, that is closer to our original intuition of the agent eventually needing to know one of a collection of facts.

Definition 3.12: We say that $\{\varphi_1, \ldots, \varphi_k\}$ is a *weak lower bound* on the K-complexity of *Task*, or *Task* is $\Omega_w(\{\varphi_1, \ldots, \varphi_k\})$, if, for every I t_0-consistent with *Task*, every agent $\mathcal{A}$, every protocol $\mathcal{P}$ for $\mathcal{A}$ that performs *Task* from I, and every run r in $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$, there exists some time m such that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}], r, m \models K\varphi_1 \vee \ldots \vee K\varphi_n$.

We can use this notion of lower bound to prove that the controller in Example 1.1b cannot perform $Task_{rob}$.

Theorem 3.13: *$Task_{rob}$ is $\Omega_w(\{g\})$.*

Proof: Suppose that $\mathcal{P}$ is a protocol for agent $\mathcal{A}$ that performs $Task_{rob}$ from $\{(0, \mathit{Table})\}$. Let r be a run in $\mathcal{I} = \mathcal{I}[\{(0, \mathit{Table})\}, \mathcal{A}, \mathcal{P}]$. Since $\mathsf{proj}_{config}(r)$ is in $Task_{rob}$, there must be some time m such that $\mathsf{proj}_{config}(r, m) \in [3, 7] \times \{Side\}$. Let m_0 be the earliest such time. It follows that at the point $(r, m_0 - 1)$, agent $\mathcal{A}$ must perform the action $\mathrm{Move}(\mathrm{arm}_2)$. We claim that $\mathcal{I}, r, m_0 - 1 \models Kg$. For suppose not. Then there must be some point (r', m') such that $\mathsf{proj}_{local}(r, m_0 - 1) = \mathsf{proj}_{local}(r', m')$ and $\mathcal{I}, r', m' \models \neg g$. Since $\mathsf{proj}_{local}(r, m_0 - 1) = \mathsf{proj}_{local}(r', m')$, it follows that the agent performs the same action at the points $(r, m_0 - 1)$ and (r', m'), namely, $\mathrm{Move}(\mathrm{arm}_2)$. It follows that r' is not in $Task_{rob}$, a contradiction. ∎

Corollary 3.14: *The controller in Example 1.1b cannot perform $Task_{rob}$.*

Proof: Suppose $\mathcal{P}$ is a protocol for the controller of Example 1.1b that performs $Task_{rob}$ starting from $\{(0, \mathit{Table})\}$. Since the controller's error bound is 4, its local state could be any one of $0, \ldots, 4$ while it is in the initial configuration $(0, \mathit{Table})$. It clearly must perform the action $\mathrm{Move}(\mathrm{arm}_1)$ when it is in the initial configuration, thus we must have $\mathcal{P}(0) = \cdots = \mathcal{P}(4) = \mathrm{Move}(\mathrm{arm}_1)$. It follows that there is a run r of $\mathcal{P}$ in which the agent reaches $(8, \mathit{Table})$ while its local state is always in $[0, 4]$. Clearly, Kg cannot hold at any point in r: it cannot hold up to the time the agent reaches $(8, \mathit{Table})$, since it does not hold in any local state in $[0, 4]$. It also cannot hold after the agent reaches $(8, \mathit{Table})$, since g does not hold. It follows from Theorem 3.13 that $\mathcal{P}$ does not perform $Task_{rob}$. ∎

Actually, we can prove Corollary 3.14 without appealing to Theorem 3.13. We simply observe that the run r constructed in the proof of the corollary is not in $Task_{rob}$. However, we feel that the appeal to Theorem 3.13 explains *why* this controller cannot perform $Task_{rob}$: its sensing capability is too weak to allow it to gain the appropriate knowledge. A similar argument can be used to show that a controller with an error bound of 3 also cannot perform $Task_{rob}$. However, a controller with an error bound of 2 can perform $Task_{rob}$, since it can implement the $SKBP$ with tests for φ_1' and φ_2 discussed in Example 3.9.

3.3 Knowledge Capability

Having defined the notions of upper and lower bounds on the K-complexity of a task, we turn to the capabilities of an agent.

Definition 3.15: Let $A = \{\varphi_1, \ldots, \varphi_k\}$ be a set of propositional formulas. Agent $\mathcal{A}$ is *K-capable* of A with respect to initial global states I if $\mathcal{I}[I, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$.

That is, an agent is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ if it always knows one of $\varphi_1, \ldots, \varphi_k$, although not necessarily the same one. Notice that this does *not* imply that the agent has the same knowledge in different runs or in different points along a single run. It may know φ_4 initially, then, after performing some action, it will know φ_7, and forget about φ_4.

This definition embodies the notion that sensing is nondeterministic. The robot may be able to guarantee that it will come to know one of several facts, but not any one of them in particular. For example, given a position sensor with ± 1 error, sensing the position when in location 4 will yield knowledge of one of the following three facts: "the location is between 2 and 4", "the location is between 3 and 5", and "the location is between 4 and 6". Yet, knowledge of any one particular statement is not guaranteed. Notice that K-capability, like our notions of upper and lower bound (and unlike the notion of weak lower bound) requires the agent to know one of $\varphi_1, \ldots, \varphi_k$ at every point in the system. However, this requirement is made for the system $\mathcal{I}[I, \mathcal{A}]$, not systems of the form $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$. But this difference is not a significant one, as the following lemma shows.

Lemma 3.16: *Let $\varphi_1, \ldots, \varphi_k$ be propositional formulas. Then $\mathcal{I}[I, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$ iff $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$ for every protocol $\mathcal{P}$ for agent $\mathcal{A}$.*

Proof: See Appendix. ∎

We could have also defined a notion of K-capability with respect to I and $\mathcal{P}$, for a protocol $\mathcal{P}$, but this does not seem to be quite so useful a notion. The following result illustrates how we intend to use K-capability.

Theorem 3.17: *If $Task = O(\{\varphi_1, \ldots, \varphi_k\})$, and $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I, where I is t_0-consistent with Task, then $\mathcal{A}$ can perform Task from I.*

Proof: If $Task = O(\{\varphi_1, \ldots, \varphi_k\})$ then there exists an SKBP Pg with knowledge conditions $K\varphi_1, \ldots, K\varphi_k$ that performs *Task*. It follows from Lemma 7.1 (see Appendix) that Pg has a good implementation from I. Since Pg performs *Task* and I is t_0-consistent with *Task*, by definition, any good implementation of Pg from I performs *Task*. This implies that $\mathcal{A}$ can perform *Task*. ∎

Theorem 3.17 suggests a useful methodology for determining whether an agent can perform a particular task: If we are able to show that the K-complexity of *Task* is $O(\{\varphi_1, \ldots, \varphi_k\})$, and that agent $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I, then we can conclude that $\mathcal{A}$ can perform *Task* from I.

Example 3.18: We return to the scenario of Example 1.1. We have seen that $Task_{rob}$ is $O(\{g, \varphi\})$, where φ denotes being in $[1,4] \cup [6,10]$. To show that controller of Example 1.1a, can perform this task, it suffices to show that it is K-capable of $\{g, \varphi\}$ with respect to $\{(0, \mathit{Table})\}$. This is easily verified. For example, in position 3, its possible readings are $2, 3, 4$, each of which makes it know g or makes it know φ. We conclude that this controller can perform $Task_{rob}$.

On the other hand, the controller of Example 1.1b, whose error bound is 4, is easily seen not to be K-capable of $\{g, \varphi\}$. For example, a reading of 5 can be obtained from anywhere within $[1,9]$, and this region is contained neither in g nor in φ. However, this does not imply that the controller cannot perform $Task_{rob}$ (although, as Corollary 3.14 shows, in fact it cannot). ∎

We end this section with a result that complements Theorem 3.17, and shows that if a task can be performed by every agent that is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$, there is an SKBP for this task with conditions $\{K\varphi_1, \ldots, K\varphi_k\}$.

Theorem 3.19: *Suppose that (1) for all I t_0-consistent with Task, if $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I, then $\mathcal{A}$ can perform Task from I, and that (2) some agent is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ from some I t_0-consistent with Task. Then $Task = O(\{\varphi_1, \ldots, \varphi_k\})$.*

Proof: The idea is to identify one particular agent $\mathcal{A}$ that is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$. In a sense, this agent knows one of $\{\varphi_1, \ldots, \varphi_k\}$ at each local state, and no more than that. We use the protocol of this agent to construct an SKBP that performs *Task*. We can then show that any execution of a good implementation of this SKBP is identical (when projected to $\mathcal{E}$) to some execution of $\mathcal{A}$'s protocol (which we know performs *Task*). The details can be found in the Appendix. ∎

4 Control Variables and Learning

The concepts developed in the previous sections form the core of an approach to the analysis of information aspects of tasks. There are a number of extensions that add to the flexibility and power of this approach. Here, we examine two such extensions: (1) adding flexibility to SKBPs through the use of control variables and (2) relaxing knowledge attainment requirements to allow for learning.

4.1 Using Control Variables

Suppose that I want to paint my wall green, but I have at my disposal only blue and yellow paint. Intuitively, I should first paint the wall blue, and then paint it yellow. That is, we should execute a program like

  while K(wall is not blue) apply blue paint;
  while K(wall is not green) apply yellow paint.

Unfortunately, this is not an SKBP. Nor is there any obvious way to implement the sequential control embodied by this program using SKBPs as we have defined them. As most programmers know, constructs such as **while** and sequential execution are often implemented using a number of control variables, or program counters, which control program execution (although most programmers are not—and should not be!—concerned with the implementation details). In their present form, skeletal knowledge-based programs do not support such convenient and natural constructs. This stems from our insistence that the interpretation function depend only on the configuration. No tests on control variables can appear in the SKBP, because the value of a control variable does not depend on the configuration. Moreover, an SKBP cannot perform the action of setting the value of a control variable, because such an action is not a transition. This renders $SKBP$s limited in their ability to describe constructs such as sequential execution.

There are several solutions for this problem. One approach, which we formalize here, is to allow for additional control bits. An SKBP extended to allow control bits would also need to allow tests on the values of the control bits, and actions that change the value of the bits. We could similarly allow (program) counters (which could take arbitrary nonnegative integer values, not just the values 0 and 1), tests on the value of the program counter, and increment and decrement operations. Additional data structures could also be allowed; whatever choice is made, it is important to specify the additional tests and actions that are allowed.

We can easily describe the program above using an SKBPwith one control bit (i.e.,, Boolean variable) b, which we assume is initially 0 (*false*). Roughly speaking, it would simply be

  if $K(\neg b \wedge$ wall is not blue) **then** apply blue paint;
  if $K(\neg b \wedge$ wall is blue) **then** $b := 1$;
  if $K(b \wedge$ wall is not green) **then** apply yellow paint.

We formalize the addition of control bits to SKBPs as follows. An *m-bit system* is a system in which the local state of the agent is made up of two disjoint components: an element of an arbitrary set L of local states (corresponding to our standard concept of a local state) and a tuple in $\{0,1\}^m$ (describing the values of the m bits). We extend the set Φ of primitive propositions by adding m new propositional symbols, $b_1, \ldots, b_m$. Let $\mathcal{L}_m$ be the result of starting with Φ_m and closing off under conjunction, negation, and knowledge. The proposition b_i (for $i = 1, \ldots, m$) is assigned `true` when the ith bit is 1. The propositions in Φ depend only on the environment state, as before.

Definition 4.1: An *m-bit system* is a system in which the agent's set of local states has the form $\{0,1\}^m \times L$. An *m-bit SKBP* is an SKBP consisting of pairs $(K\alpha, \tau)$ where $\alpha \in \mathcal{L}_m$ and $\tau = (\tau', S)$, where $\tau' \in \Lambda$ and $S : 2^m \rightarrow 2^m$ assigns values to $b_1, \ldots, b_m$ based on their old values.

It is now straightforward to generalize the notions of "implementation", "good implementation", "can execute", and "can perform *Task*" to m-bit SKBP. There is one minor subtlety. To ensure correct flow of control, we must assume that the control bits are initialized when the execution of an m-bit protocol commences. We have arbitrarily chosen 0 as the initial value. Thus, when it comes to defining what it means for an SKBP to perform *Task*, we restrict to initial global states of where the agent's local state has the form $(0, \dots, 0, l)$.

Definition 4.2: If I is a set of global states, define $I_{+m} = \{(c, ((0, \dots, 0), l)) : (c, l) \in I\}$. The m-bit *SKBP* Pg *performs Task* if all its good implementations from every set I_{+m} t_0-consistent with *Task* perform *Task* from I_{+m}, and it has a (good) implementation from some I_{+m} t_0-consistent with *Task*.

Definition 4.3: Let $\varphi_1, \dots, \varphi_k \in \mathcal{L}$; $\{\varphi_1, \dots, \varphi_k\}$ is an m-bit *upper bound* on the K-complexity of a task *Task*, written $Task = O_m(\{\varphi_1, \dots, \varphi_k\})$, if there exists an m-bit *SKBP* Pg with knowledge conditions $K(\varphi_1 \wedge \beta_1), \dots, K(\varphi_k \wedge \beta_k)$, where $\beta_1, \dots, \beta_k$ are propositional formulas that mention only the primitive propositions $b_1, \dots, b_m$, that performs *Task*.

Notice that our old O notation is equivalent to O_0.

It is straightforward to generalize Theorem 3.17 using the O_m notation.

Definition 4.4: Given an agent $\mathcal{A} = \langle L \times Actions \rangle$, define agent $\mathcal{A}_{+m}$ as $\mathcal{A}_{+m} \stackrel{\text{def}}{=} \langle \{0,1\}^m \times L, Actions \times Set_m \rangle$, where Set_m is the set of functions from 2^m to 2^m.

Theorem 4.5: *If $Task = O_m(\{\varphi_0, \dots, \varphi_k\})$ and $\mathcal{A} = \langle L, Actions \rangle$ is K-capable of $\{\varphi_0, \dots, \varphi_k\}$ in $\mathcal{I}[I, \mathcal{A}]$ for I t_0-consistent with Task, then $\mathcal{A}_{+m}$ can perform Task from I_{+m}.*

Proof: With slight modification to accommodate the m additional control bits, the proofs of Lemma 7.1 and Theorem 3.17 generalize to this case. ▮

4.2 Learning

If *Task* is $O(\varphi_1, \dots, \varphi_k)$, we know that there is an SKBP that performs *Task* using only the tests $K\varphi_1, \dots, K\varphi_k$. Hence, if an agent always knows one of $\varphi_1, \dots, \varphi_k$, it can perform *Task*; this is the essence of Theorem 3.17. In practice, however, it may be unreasonable to expect that we can build an agent that always knows one of $\varphi_1, \dots, \varphi_k$. For example, suppose our agent must assemble some device which requires using a wrench. A high-level protocol for this task would probably call for knowledge of the location of this wrench. The agent may not always know this location. However, it can always learn it by examining the contents of the tool box and the drawer. More generally, although the agent may not always know one of $\varphi_1, \dots, \varphi_k$, it may be able to learn one of these formulas.

This example suggests a useful methodology for designing agents: First, try to understand the knowledge requirements of the task. Then, see if you can build an agent that can learn these requirements. Roughly, we say that an agent can learn $\{\varphi_1, \dots, \varphi_k\}$ if it can execute a learning protocol that terminates with the agent knowing one of these formulas. For technical reasons, we require the learning program to have no side effects on the environment, so that if the agent begins execution of the learning program in configuration c, it ends in the same configuration (although the configuration may change during the execution of the learning program).

Definition 4.6: Agent $\mathcal{A} = \langle L, Actions\rangle$ *can learn* $\{\varphi_1, \ldots, \varphi_k\}$ in $\mathcal{I}[I, \mathcal{A}]$ if there exist a set $L_T \subseteq L$ and a protocol $\mathcal{P}$ for $\mathcal{A}$ such that for all runs r of $\mathcal{P}$, there exists some $m \in N$ such that (1) $\mathsf{proj}_{config}(r(m)) = \mathsf{proj}_{config}(r(0))$; (2) $\mathsf{proj}_{local}(r(m)) \in L_T$; (3) $\mathcal{I}[I, \mathcal{A}], r, m \models K\varphi_1 \vee \ldots \vee K\varphi_k$; (4) for all $m' < m$, we have that $\mathsf{proj}_{local}(r(m')) \notin L_T$. Let $\mathcal{P}(c, l) = \{(c, l') : l' \in L_T$ and there exists a run r of $\mathcal{P}$ and time m such that $r(0) = (c, l)$, $r(m) = (c, l')$, and for all $m' < m$ we have $\mathsf{proj}_{local}(r(m')) \notin L_T\}$.

Hence, in order to be able to learn $\{\varphi_1, \ldots, \varphi_k\}$ in $\mathcal{I}[I, \mathcal{A}]$, agent $\mathcal{A}$ needs a learning program $\mathcal{P}$ and a termination condition such that (1) any execution of $\mathcal{P}$ eventually terminates, and (2) upon termination $\mathcal{A}$ knows one of $\{\varphi_1, \ldots, \varphi_k\}$ and the environment is restored to its initial configuration.

It may seem that if knowledge of one of $\varphi_1, \ldots, \varphi_k$ suffices to perform *Task*, and agent $\mathcal{A}$ can always learn one of these formulas, then $\mathcal{A}$ will be able to perform *Task*. Unfortunately, this is not always the case. For example, consider a task that requires reaching some goal configuration in a bounded period of time. We may have an SKBP for performing this task that requires knowledge of one of $\varphi_1, \ldots, \varphi_k$ at each point in time, but if we employ lengthy subroutines to learn these formulas, we may not be able to meet the time constraints of the task. However, when the task is flexible in terms of execution time, we can combine the learning subroutines with the main protocol.

Definition 4.7: *Task* is said to be *elastic* if for every C_1 and C_2 (where C_1 is a finite sequence of configurations and C_2 is a C-history) such that $\mathsf{C}_1 \cdot \mathsf{C}_2 \in Task$, it is the case that $\mathsf{C}_1 \cdot c \cdot \mathsf{C}_2 \in Task$ as well. (Where $c \in \mathcal{E}$ and $\cdot$ is the concatenation operator.)

The following result illustrates the role we envision for learning results. Whereas in Theorem 4.5 we showed that, under appropriate conditions, an agent that is K-capable of $\{\varphi_0, \ldots, \varphi_k\}$ can perform $O(\{\varphi_0, \ldots, \varphi_k\})$ tasks, we now show that an agent that can learn $\{\varphi_0, \ldots, \varphi_k\}$ can perform such tasks.

Theorem 4.8: *If Task* $= O_m(\{\varphi_0, \ldots, \varphi_k\})$*, Task is elastic, and* $\mathcal{A} = \langle L, Actions\rangle$ *can learn* $\{\varphi_0, \ldots, \varphi_k\}$ *in* $\mathcal{I}[I, \mathcal{A}]$ *for some* t_0*-consistent* I*, then* $\mathcal{A}_{+(m+1)}$ *can perform Task from* $I_{+(m+1)}$.

Proof: Intuitively, we show that $\mathcal{A}$ can perform *Task* by behaving as though it is K-capable of $\{\varphi_0, \ldots, \varphi_k\}$. Whenever it reaches a state in which it does not have sufficient information, it employs an appropriate learning subroutine. The details can be found in the Appendix. ∎

Roughly, an agent that employs a learning subroutine can be viewed as running an implementation of the following SKBP:

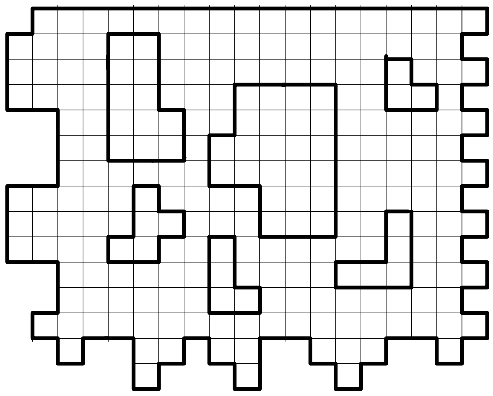

Figure 2: A maze

case of
  if $K\alpha_1$ **then** τ_1;
  if $K\alpha_2$ **then** τ_2;
  ...
  if $K\alpha_n$ **then** τ_n;
  else learn one of $K\alpha_1, \ldots K\alpha_n$;

This is a special class of SKBPs in which not all conditions are positive. In general, we believe that negative conditions play precisely this role, acting as learning subroutines.

The concepts introduced so far, together with results such as Theorems 3.17,4.5, and 4.8, suggest the following methodology for task and agent analysis: (1) Characterize the knowledge complexity of a task; (2) Characterize the knowledge capabilities of the agent; (3) Understand what the agent is capable of learning; (4) Combine these results to understand whether an agent can perform the task. In addition, answers to the first question provide necessary insight for the design of agents capable of performing a particular task. We illustrate these ideas in the following example.

Example 4.9: We examine the problem of maze searching. This domain, which has received considerable attention in the past (e.g., [Bud75, BK78]), allows us to illustrate the use of our formal language in a nontrivial application. More importantly, we shall show that existing work in this area, due to Blum and Kozen [BK78], can be best understood as performing a knowledge-complexity analysis of this domain that naturally fits within the above methodology. This perspective was not explicitly taken by the original work. The first author to adopt such an information-analytic perspective of Blum and Kozen's results was Donald [Don94], in the context of his theory of information invariants. This work, in turn, led us to attempt to provide a general language and methodology, based on the concept of knowledge, for capturing such information complexity analysis.

A maze is a finite, two-dimensional, obstructed checkerboard (see Figure 2). To search a maze, a robot, started on any cell, must eventually visit every reachable cell without passing through any of the obstacles. At each time step, the robot can move one unit in any one of the directions north, east, south, or west, as long as the target cell is not part of an obstacle. Budach [Bud75] has shown that a finite-state robot cannot search all mazes. Later, Blum

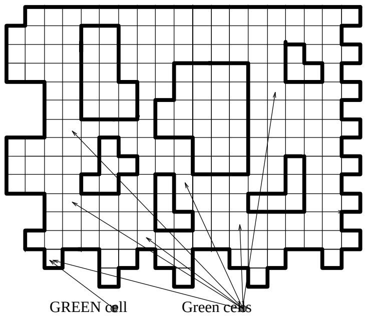

Figure 3: *GREEN* and *Green*

and Kozen showed that two finite-state robots can search all mazes, and that a single robot with a counter can search all mazes as well. We shall show that Blum and Kozen's work can be interpreted as characterizing the knowledge complexity of maze-searching for agents and the knowledge capabilities of a number of agents.

In this domain, the configuration space consists of pairs of the form

(maze, non-obstructed cell of that particular maze).

Each such state can be a possible initial state, i.e., the robot may start at any cell in any maze. The physical capabilities of the robot are such that it can move to any non-obstructed cell to its immediate north, east, south, or west.

The task description corresponds to the set of trajectories in which all non-obstructed cells within the maze are visited. That is, the robot is thrown into some random cell in some random finite maze and must visit all non-obstructed cells in this maze. Clearly, this is an elastic task.

Assume that the language contains the following propositions: *b-north*, *b-east*, *b-south*, *b-west*, *Green*, and *GREEN*. A state satisfies *b-north*, *b-east*, *b-south*, or *b-west* when the adjacent *north/east/south/west* cell is obstructed. A state satisfies the proposition *Green* when one or more of the four vertices of the cell is green. A vertex is green if it is the unique point (x_0, y_0) of some boundary BDRY (i.e., either the boundary of the whole maze or the boundary of one of the obstacles) such that for all $(x, y) \in$ BDRY, $[y_0 \leq y$ or $(y_0 = y \;\&\; x_0 \leq x)]$. In particular, if this unique point lies at the southwest corner of the cell, *GREEN* is satisfied (see Figure 3).

Blum and Kozen prove that a robot with an infinite counter that always knows the value of the propositions *b-north*, *b-east*, *b-south*, and *b-west* can search all finite mazes. What is interesting from our perspective is the manner in which this result is proved.

First, we can conclude from Blum and Kozen's work that maze searching is $O_3(CON(\{GREEN, Green,$ $north, b\text{-}east, b\text{-}south, b\text{-}west\})$, where $CON(\{\alpha_1, \ldots, \alpha_k\})$ consists of all the formulas of the form $\beta_1 \wedge \ldots \wedge \beta_k$, where β_i is either α_i or $\neg\alpha_i$. That is, any robot that always knows the value of the propositions $GREEN, Green, b\text{-}north, b\text{-}east, b\text{-}south, b\text{-}west$ can search all finite mazes. In fact, Blum and Kozen provide a conditional plan for searching all finite mazes in which the only conditions refer to the value of these propositions. Control of execution of this plan requires no more than three extra bits.

Blum and Kozen also show that an agent with an infinite counter that always knows the value of the propositions *b-north*,*b-east*,*b-south*,*b-west* can learn the value of *Green* and *GREEN*. Because maze searching is an elastic task, these results can be combined, a manner similar to Theorem 4.8, to show that such a robot can search all finite mazes. ▮

5 Related Work

We have attempted here to unify work on knowledge in multi-agent systems in the distributed systems community with work on information and sensing in the robotics community. As we mentioned earlier, we were particularly influenced by the earlier work of Donald [Don94] and Erdmann [Erd94] on the robotics side, and the work of Fagin et al. [FHMV95] on the distributed systems side. We briefly discuss the connection between our results and related work in this section.

5.1 Erdmann's Abstract Sensors

In [Erd94], Erdmann argues that the role of sensors is to provide sufficient information to choose "good" actions, and that they should be constructed to fulfill this task. An action is good if it makes progress towards attaining the goal state according to some progress measure. Hence, given a progress measure, we can assess the sensing requirements of a task by examining which actions make progress in which states. Erdmann shows how we can obtain a progress measure for a task from an algorithm for that task.

Erdmann's description of sensors is abstract, given in terms of the sets of states they can distinguish between. Formulas, too, are given semantics in terms of the set of states in which they hold. An abstract sensor is a sensor that tells the agent that it is within some set S. This is naturally captured by our concept of *knowledge*: if S is the set of states in which φ holds then we can say that, given the sensor reading, the agent knows φ. Hence, we see that the semantics of Erdmann's abstract sensors is closely related to the semantics of knowledge.

Our work can be viewed as formalizing some of Erdmann's ideas using the concept of knowledge. While we have added new concepts to those discussed by Erdmann, it would have been possible to develop essentially similar ideas using a purely set-theoretic framework, as in Erdmann's work.[7] However, the logical framework we provide is more suitable for extending these ideas to multi-agent systems in which the issue of information requirements arises naturally (e.g., in the problem of task distribution [DJR93]). In this context, a set-theoretic representation of an agent's information is quite cumbersome and opaque, while the epistemic language used here is much more transparent and intuitive. Indeed, we believe this is true even in the single-agent case. In addition, suggests other tools for describing and analyzing the capabilities of agents such as the notions of K-capability and learning we have introduced here.

[7]Indeed, this is true of most applications of knowledge theory. More generally, one could do away with any formal logical language (that has an adequate semantics) reasoning with its model theory directly.

5.2 Donald's Capability Classes

Donald [Don94] attempted to classify sensors into capability classes, to quantify the information capabilities of sensor systems, and to characterize the relationship between different capability classes. This work motivated many of the questions we are concerned with, leading us to adopt a more "complexity-theoretic" approach, as well as our notion of K-capability.

There are, however, significant differences between Donald's framework and our framework. In its aim and its semantics, our work is much closer to Erdmann's work. Like Erdmann, we emphasize the sensing requirements of tasks. Donald, on the other hand, emphasizes the sensing capabilities of system. Donald's notion of capability of systems is more detailed than our notion of K-capability, and its definition is less abstract and more geometric. Because his concepts are defined with respect to a lower, more detailed system description than ours, many of them have no analogues in our framework.

5.3 Knowledge in Multi-Agent Systems

Logics of knowledge were introduced into the study of distributed systems by Halpern and Moses [HM90] and into artificial intelligence by Moore [Moo85] and Rosenschein [Ros85]. The semantics of knowledge we have adopted is based on [HM90, Ros85], but much of the formal development, e.g., the concepts of runs, systems, and knowledge-based programs is essentially taken from [FHMV95], although there are some differences, as we discussed earlier.

Previous work in distributed systems has used knowledge as a tool for analyzing and reasoning about the information requirements of tasks. Lower bounds on the information requirements of certain basic tasks in distributed systems have also been established. For example, as we mentioned earlier, Halpern and Moses [HM90] show that common knowledge is required to perform coordinated attack and that it cannot be attained in many systems of interest. Chandy and Misra [CM86] consider a system of n processors in which the property of mutual exclusion with respect to some critical section is maintained, and show that under certain assumptions on this system, a process must have certain knowledge when it enters the critical section. Then they establish a lower bound on the number of messages that must be sent for this knowledge to be attained. Consequently, one can deduce that at least this number of messages must be passed among processors if the critical section is to be maintained. These results provide important motivation for the approach we have taken to the analysis of tasks.

5.4 Knowledge, Actions, and Plans

Epistemic logic has played an important role in formalizing the process of planning under uncertainty. Most notably, the work of Moore [Moo85] and Morgenstern [?] is concerned with supplying appropriately expressive knowledge representation tools for agents that must reason about their knowledge in the course of their planning activities. Such work considers the issue of knowledge preconditions for plans and the conditions under which an agent knows how to perform an action (where that last term is quite broadly defined). While there are many similarities between these concerns and the concerns of our paper, they differ in terms of their goal and viewpoint. Whereas Moore and Morgenstern are concerned with formalizing the task of planning from the perspective of the agent, we are concerned with supplying tools to designers of agents. Hence, their perspective is internal, taking the point of view of the planning agent, and our perspective is external, taking the point of view of an external analyst or designer. With these distinct objectives come different assumptions. We assume that our designer has an accurate model of the domain and of the robot's actuators. The designer's task is to choose appropriate sensors and software that will allow the robot to perform its tasks. She is not constrained to have some particular amount of knowledge; rather, she will attempt to discover the amount of knowledge needed for performing the task. Moore and Morgenstern, on the other hand, consider planning agents who lack knowledge of the precise effects of their actions and may need to actively plan in order to learn this information. Having taken the design perspective and relying on an accurate model of the domain, the need for greater expressive power and realistic modeling of the agent's knowledge that motivates some of the developments of Moore and of Morgenstern is less of an issue for us.

6 Future Work

We have shown how formal measures of informational complexity and capability can be used to analyze robotic systems. We have only scratched the surface here. There are a number of interesting issues that are worth exploring further. We briefly list a few of them here:

- We have focused on the problems where we can discuss what must be known at every step of the computation. As we mentioned earlier, there are times when we are interested only in what must be known eventually. It would be of great interest to extend our notions of knowledge complexity and knowledge capability so that they can deal with this. Notice that, among other things, this would mean extending our notions so that we can use a richer language, involving temporal connectives, to express complexity and capability.

- The notion of K-complexity differs from those of time and space complexity in that K-complexity values are not totally ordered. However, there appears to be an interaction between K-complexity on the one hand and time and space complexity on the other. Intuitively, while a robot with minimal knowledge might be able to perform a task, with more knowledge it might be able to perform the task more efficiently in terms of computation time or space. As we have seen, allowing the agent additional control bits can enable the agent to gain knowledge. It would be interesting to understand better the tradeoffs between time/space complexity and K-complexity.

- We have implicitly assumed that an agent can use all the information implicit in its local state. In general, it may be quite difficult for an agent to compute what it knows, as a function of its internal state. For example, there may be a great deal of information encoded in the local state of a vision system. Getting a computational notion of knowledge that deals with this information is an interesting and difficult problem (see the discussion of logical omniscience and algorithmic knowledge in [FHMV95]).

- Once we allow the agent to learn, we can explore the possibility of *knowledge reductions*. Roughly speaking, we can say that a set A of propositional formulas is reducible to B if knowing the formulas in B suffices for learning the formulas in A. For example, consider a robot in an obstacle-free maze that can move in any direction. Suppose the robot is equipped with a touch sensor, allowing it to detect if it is adjacent to the boundary. Hence, this robot is K-capable of $\{\mathit{Side}, \neg\mathit{Side}\}$, where *Side* is true if the robot is immediately adjacent to the boundary. Suppose the proposition *near* is true if the robot is one step away from the boundary. The robot is not K-capable of $\{\mathit{near}, \neg\mathit{near}\}$, but clearly the robot can learn $\{\mathit{near}, \neg\mathit{near}\}$ (perhaps with the aid of a few control bits). In this case, we can say that *near* is K-reducible to $\{\mathit{Side}, \neg\mathit{Side}\}$. Using K-reducibility together with learning could enhance our general methodology of designing programs top-down, starting with knowledge and then implementing the knowledge tests.

- We are particularly interested in applying our ideas to the problem of task distribution, in which information plays a crucial role. In task distribution, a central controller for the system must be replaced by a set of distributed controllers. In order to succeed, the distributed controllers must have sufficient information about the state of the other components of the system as well as about the state of the external world. However, we would like them to have this information with as little additional overhead of communication.

  The concept of knowledge, on which the formalism introduced in this paper rests, extends naturally to such multi-agent settings. Subscripted knowledge operators can be used to denote information of a particular agent. Intuitively, $K_{\mathsf{a}}\varphi$ says that agent a knows φ, and $K_{\mathsf{b}}\varphi$ says that agent b knows φ. Moreover, using nested knowledge operators, we can describe knowledge one agent has about another agent's knowledge. For example, $K_{\mathsf{b}}K_{\mathsf{a}}\varphi$ says that agent b knows that agent a knows φ.

  Conceptually, designing a centralized controller is easier than designing a set of distributed controllers. Yet sometimes, a distributed solution is desirable or essential. An intermediate approach would be to design, or synthesize, a centralized controller, or SKBP, for a task, and use it to derive a distributed SKBP; this distributed SKBP would tell us what information each agent requires. For example, the following distributed SKBP performs the task discussed throughout this paper:

  $$\mathsf{Pg}(\mathrm{arm}_1)=\{\textbf{If } K_1 g \textbf{ then } \mathrm{Stop}; \textbf{ If } K_1\varphi \wedge \neg K_1 g \textbf{ then } \mathrm{Move}\},$$
  $$\mathsf{Pg}(\mathrm{arm}_2)=\{\textbf{If } K_2 K_1 g \textbf{ then } \mathrm{Move}; \textbf{ If } K_2\neg K_1 g \textbf{ then } \mathrm{Stop}\}.$$

  An examination of this SKBP[8] shows that the second arm is always required to know whether the first arm knows g or not. In designing a distributed controller for this system, we must take care to provide such knowledge to the second arm, whether directly, via communication, or indirectly, via some observation. An algorithm for automatically transforming an SKBP for a centrally controlled system to a distributed SKBP was presented in [BS95]. While the applicability of this algorithm is still unclear, we hope that pursuing these ideas will lead us to a better understanding of decentralization.

[8]This is a more general form of SKBP in which a negative knowledge condition appears.

- Our approach (as well as Donald's and Erdmann's) holds the capabilities of the robots fixed and measures only sensing complexity. However, we would also like to understand whether it is possible to combine our measure of informational complexity with a measure of actuation complexity to obtain a better characterization of the information complexity of tasks and capabilities of robots.

More generally, we view our work as continuing a tradition of attempting to understand some basic and difficult issues in the design of situated systems. There seems to be quite a way to go until all the current ideas in this area converge to a single accepted model. We believe that the framework presented here makes some progress towards this goal.

7 Appendix: Proofs

Proposition 3.8: *If $A \preceq_{\mathcal{E}} B$, Task is $O(A)$, and $\mathcal{E} \models \vee_{\varphi \in B}\varphi$, then Task is $O(B)$.*

Proof: Suppose $A = \{\varphi_1, \ldots, \varphi_k\}$, and $B = \{\psi_1, \ldots, \psi_m\}$. Because *Task* $= O(A)$, there exists some SKBP $\mathsf{Pg} = \{(K\varphi_i, \tau_i) \mid i \in \{1, \ldots, k\}\}$ that performs *Task*. Because $A \preceq_{\mathcal{E}} B$, there is a function $f : \{1, \ldots, m\} \to \{1, \ldots, k\}$ such that $\mathcal{E} \models \psi_i \Rightarrow \varphi_{f(i)}$. Let Pg' be the SKBP $\{(K\psi_i, \tau_{f(i)}) \mid i \in \{1, \ldots, m\}\}$. Clearly its knowledge conditions are $\{K\psi_1, \ldots, K\psi_m\}$. We claim that it performs *Task*, and that, therefore, *Task* $= O(B)$. In order to prove this claim, it suffices to show that (1) any good implementation of Pg' is a good implementation of Pg and (2) there is a good implementations of Pg'.

To prove (1), suppose that $\mathcal{P}$ is a good implementation of Pg' for some agent $\mathcal{A} = (L, \mathit{Actions})$ from some t_0-consistent I. At each local state $l \in L$, agent $\mathcal{A}$ performs some action a_l that implements some transition $\tau_{f(i)}$ such that $(K\psi_i, \tau_{f(i)}) \in \mathsf{Pg}'$ and $\mathcal{I}[I, \mathcal{A}, \mathcal{P}], l \models K\psi_i$. Since $\mathcal{E} \models \psi_i \Rightarrow \varphi_{f(i)}$, the properties of the K operator guarantee that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}], l \models K\psi_i \rightarrow K\varphi_{f(i)}$. Thus, at each local state $l \in L$, agent $\mathcal{A}$ performs some action a_l that implements some transition $\tau_{f(i)}$ such that $(K\varphi_{f(i)}, \tau_{f(i)}) \in \mathsf{Pg}$ and $\mathcal{I}[I, \mathcal{A}, \mathcal{P}], l \models K\varphi_{f(i)}$. This implies that $\mathcal{P}$ is an implementation of Pg from I. To see that $\mathcal{P}$ is a good implementation of Pg from I, notice that, since $\mathcal{P}$ is a good implementation of Pg' from I, we have $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\psi_1 \vee \ldots \vee K\psi_m$. By the arguments above, it follows that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_{f(1)} \vee \ldots \vee K\varphi_{f(m)}$. Thus, $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_1 \vee \ldots, K\varphi_k$, as desired.

To see that (2), there is a good implementation of Pg' starting from some t_0-consistent I, let $\mathcal{A}^* = (L^*, \mathit{Actions}^*)$ be the agent, that, intuitively, has perfect information. More precisely, let $L^* = \mathcal{E}$; for each transition τ, let a_τ be the action defined by $a_\tau(c, c) = (\{(d, d) : d \in \tau(c)\}$. Let $\mathit{Actions} = \{a_\tau : \tau \in \Lambda\}$. Let $I^* = \{(c, c) : c$ is the initial configuration of some C-history in *Task*$\}$. Let $\mathcal{P}^*$ be the protocol for agent $\mathcal{A}^*$ defined via $\mathcal{P}^*(c) = \{a_{\tau_{f(i)}} : c \models \psi_i, 1 \leq i \leq m\}$. Clearly, the only global states that arise in runs in $\mathcal{I}[I^*, \mathcal{A}^*, \mathcal{P}^*]$ have the form (c, c) for $c \in \mathcal{E}$. It easily follows that if $c \models \psi$ for some propositional formula c, then $\mathcal{I}[I^*, \mathcal{A}^*, \mathcal{P}^*], c \models K\psi$. Since we have assumed that $\mathcal{E} \models \psi_1 \vee \ldots \vee \psi_m$, it follows that $\mathcal{I}[I^*, \mathcal{A}^*, \mathcal{P}^*] \models K\psi_1 \vee \ldots \vee K\psi_m$. It is also easy to see that $\mathcal{P}^*$ implements Pg' from I^*. Thus, there is a good implementation of Pg' from some t_0-consistent set of initial states. ∎

Lemma 3.16: *Let $\varphi_1, \ldots, \varphi_k$ be propositional formulas. Then $\mathcal{I}[I, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$ iff $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$ for every protocol $\mathcal{P}$ for agent $\mathcal{A}$.*

Proof: First suppose that $\mathcal{I}[I, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$. By definition, $S[I, \mathcal{A}, \mathcal{P}] \subseteq S[I, \mathcal{A}]$. It easily follows that for every propositional formula φ and every point (r, m) in $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$, if $\mathcal{I}[I, \mathcal{A}], r, m \models K\varphi$ then $\mathcal{I}[I, \mathcal{A}, \mathcal{P}], r, m \models K\varphi$. Thus, we have that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$. For the converse, suppose $\mathcal{A} = (L, \mathit{Actions})$. Let $\mathcal{P}^*$ be the protocol that allows the agent to perform every possible action at every local state; that is, $\mathcal{P}(l) = 2^{\mathit{Actions}} \setminus \emptyset$ for each $l \in L$. It is easy to see that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}^*] = \mathcal{I}[I, \mathcal{A}]$. Hence, if $\mathcal{I}[I, \mathcal{A}, \mathcal{P}^*] \models K\varphi_1 \vee \ldots \vee K\varphi_k$, then surely $\mathcal{I}[I, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$. ∎

The following lemma is used in the proof of Theorem 3.17, and is also of interest in its own right, since it shows an important special case where an implementation is guaranteed to exist.

Lemma 7.1: *If Pg is an SKBP with positive knowledge conditions $K\varphi_1, \ldots, K\varphi_k$, and $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I, then Pg has a good implementation from I.*

Proof: Let $\mathcal{P} = \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A}]}$. We claim that $\mathcal{P}$ is a good implementation of Pg. In order to prove this, we have to show that $\mathcal{P}(l) \subseteq \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}(l)$, and that $\mathcal{I}[I, \mathcal{A}, \mathcal{P}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$. The latter fact follows immediately from Lemma 3.16, since $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I. The former also follows along much the same lines as the proof of Lemma 3.16. Since $S[I, \mathcal{A}, \mathcal{P}] \subseteq S[I, \mathcal{A}]$, it follows that for every propositional formula φ and run r in $\mathcal{I}[I, \mathcal{A}, \mathcal{P}]$, if $\mathcal{I}[I, \mathcal{A}], r, m \models K\varphi$ then $\mathcal{I}[I, \mathcal{A}, \mathcal{P}_s], r, m \models K\varphi$. Thus, every knowledge condition that holds at the local state l in the system $\mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}$ also holds at l in the system $\mathsf{Pg}^{\mathcal{I}[I,\mathcal{A}]}$. Thus, $\mathsf{Pg}^{\mathcal{I}[I,\mathcal{A},\mathcal{P}]}(l) \supseteq \mathsf{Pg}^{\mathcal{I}[I,\mathcal{A}]}(l) = \mathcal{P}(l)$. ∎

Theorem 3.17: *If $\mathit{Task} = O(\{\varphi_1, \ldots, \varphi_k\})$ and $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I, where I is t_0-consistent with Task, then $\mathcal{A}$ can perform Task from I.*

Proof: If $\mathit{Task} = O(\{\varphi_1, \ldots, \varphi_k\})$ then there exists an SKBP Pg with knowledge conditions $K\varphi_1, \ldots, K\varphi_k$ that performs *Task*. It follows from Lemma 7.1 that Pg has a good implementation from I. Since Pg performs *Task* and I is t_0-consistent with *Task*, by definition, any good implementation of Pg from I performs *Task*. This implies that $\mathcal{A}$ can perform *Task*. ∎

Theorem 3.19: *Suppose that (1) for all I t_0-consistent with Task, if agent $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I then $\mathcal{A}$ can perform Task from I, and that (2) some agent is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ from some I t_0-consistent with Task. Then $\mathit{Task} = O(\{\varphi_1, \ldots, \varphi_k\})$.*

Proof: We must find an *SKBP* Pg with conditions $K\varphi_1, \ldots, K\varphi_k$ that performs *Task*. We do this as follows: We first define a particular agent $\mathcal{A}$ that can perform *Task*. We use the program $\mathcal{P}_{\mathcal{A}}$ it uses to perform *Task* to define the required *SKBP* Pg. Then we show that any execution of Pg is identical (when projected to $\mathcal{E}$) to an execution of $\mathcal{P}_{\mathcal{A}}$. Since $\mathcal{P}_{\mathcal{A}}$ performs *Task*, all of its executions are in the task, and hence, we will have shown that all executions of Pg are in the task.

Agent $\mathcal{A} = (L, Actions)$ is constructed as follows. Let $L = \{l_0, \ldots, l_k\}$. For each configuration c, let $Holds(c) = \{l_j : c \models \varphi_j, j = 1, \ldots, k\}$ if $c \models \varphi_1 \vee \ldots \vee \varphi_k$, and let $Holds(c) = \{l_0\}$ otherwise. For a transition $\tau \in \Lambda$, let a_τ be an action defined by $a_\tau(c, l) = \cup_{c' \in \tau(c)}(\{c'\} \times Holds(c'))$. We define $Actions = \{a_\tau : \tau \in \Lambda\}$.

Let $I_\mathcal{A}$ be $\{(\mathsf{C}(0), l) | \mathsf{C} \in \mathit{Task},\ l \in Holds(\mathsf{C}(0))\}$. We claim that $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to $I_\mathcal{A}$. It is easy to see that our construction guarantees that $\mathcal{I}[I_\mathcal{A}, \mathcal{A}], l_j \models K\varphi_j$. Thus, it suffices to show that the local state l_0 does not arise in $\mathcal{I}[I_\mathcal{A}, \mathcal{A}]$.

To see this, first notice that if I and I' are sets of initial states t_0-consistent with *Task*, and $\mathcal{A}$ and $\mathcal{A}'$ are two agents situated in $(\mathcal{E}, \Lambda)$, then the set of configurations that arise in $\mathcal{I}[I, \mathcal{A}]$ and $\mathcal{I}[I', \mathcal{A}']$ must be the same, since this set of configurations depends only on the possible initial configurations (given our assumption that for every transition $\tau \in \Lambda$, there is an action that each of $\mathcal{A}$ and $\mathcal{A}'$ can perform that agrees with τ when projected to configurations). Next, observe that assumption (2) in the theorem guarantees that there is some agent $\mathcal{A}'$ and some initial set I' of global states t_0-consistent with *Task* such that $\mathcal{A}'$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ with respect to I'. Thus, $\mathcal{I}[I', \mathcal{A}'] \models K\varphi_1 \vee \ldots \vee K\varphi_k$. It follows that $\mathcal{I}[I', \mathcal{A}'] \models \varphi_1 \vee \ldots \vee \varphi_k$. Since the same configurations arise in $\mathcal{I}[I', \mathcal{A}']$ and $\mathcal{I}[I_\mathcal{A}, \mathcal{A}]$, and the truth of a propositional formula depends only on the configuration, it follows that $\mathcal{I}[I_\mathcal{A}, \mathcal{A}] \models \varphi_1 \vee \ldots \vee \varphi_k$. Thus, $Holds(c) \neq \{l_0\}$ for any configuration c that arises in $\mathcal{I}[I_\mathcal{A}, \mathcal{A}]$. Hence, the local state l_0 does not arise in $\mathcal{I}[I_\mathcal{A}, \mathcal{A}]$. It follows that $\mathcal{I}[I_\mathcal{A}, \mathcal{A}] \models K\varphi_1 \vee \ldots \vee K\varphi_k$, as desired.

It now follows from assumption (1) that $\mathcal{A}$ can perform *Task* from $I_\mathcal{A}$, and there must be a protocol $P_\mathcal{A}$ for $\mathcal{A}$ that performs *Task* from $I_\mathcal{A}$. Define the SKBP $\mathsf{Pg} = \{(K\varphi_i, \tau) \in \mathcal{P} : a_\tau \in P_\mathcal{A}(l_i), i = 1, \ldots, \}$. We claim that Pg performs *Task*. In order to prove this, we must show that if $\mathcal{B}$ is an agent and $\mathcal{P}_\mathcal{B}$ is a good implementation of Pg from some set of initial states $I_\mathcal{B}$ that is t_0-consistent with *Task*, then $\mathcal{P}_\mathcal{B}$ performs *Task* from $I_\mathcal{B}$. Since $\mathcal{P}_\mathcal{B}$ is a good implementation of Pg from $I_\mathcal{B}$, we have that $\mathcal{I}[I_\mathcal{B}, \mathcal{B}, \mathcal{P}_\mathcal{B}] \models K\varphi_1 \vee \cdots \vee K\varphi_k$, and that if $a \in P_\mathcal{B}(l)$, then there is some j such that $(K\varphi_j, \tau_a) \in \mathcal{P}$ and $\mathcal{I}[I_\mathcal{B}, \mathcal{B}, P_\mathcal{B}], l \models K\varphi_j$. Suppose that r is an execution of $P_\mathcal{B}$ from $(c, l) \in I_\mathcal{B}$, and let C denote the C-history defined by r (i.e., $\mathsf{C} = \pi_{\mathit{config}}(r)$). We complete this proof by showing that there is an execution of $P_\mathcal{A}$ from some global state $(c, l') \in I_\mathcal{A}$ that defines precisely the same C-history C. This implies that $\mathsf{C} \in \mathit{Task}$, because $P_\mathcal{A}$ performs *Task*. Therefore, we have that $P_\mathcal{B}$ performs *Task* from $I_\mathcal{B}$.

Let $a_0^\mathcal{B}, a_1^\mathcal{B}, \ldots$ be the sequence of actions performed by $\mathcal{B}$ along the run r, and let $\tau_0, \tau_1, \ldots$ be the transitions that $a_0^\mathcal{B}, a_1^\mathcal{B}, \ldots$ implement. Let $\varphi_{i_0}, \varphi_{i_1}, \ldots$ be a sequence of formulas (each of which belongs to $\{\varphi_1, \ldots, \varphi_k\}$) such that $(K\varphi_{i_n}, \tau_n) \in \mathcal{P}$ and $\mathsf{C}(n) \models \varphi_{i_n}$. That such formulas exists follows from the fact that $\mathcal{P}_\mathcal{B}$ implements $\mathcal{P}$: By definition, the action assigned by $\mathcal{P}_\mathcal{B}$ at a state (c, l) implements a transition assigned by $\mathcal{P}$ to some knowledge condition $K\varphi$ that holds at (c, l); thus, $c \models \varphi$.

We define $r_\mathcal{A}$ by taking $r_\mathcal{A}(n) = (\mathsf{C}(n), l_{i_n})$ for all n. Obviously, $\mathsf{proj}_{\mathit{config}}(r_\mathcal{A}) = \mathsf{proj}_{\mathit{config}}(r)$. It remains to show that $r_\mathcal{A}$ is an execution of $\mathcal{P}_\mathcal{A}$. In order to prove this we have to show that for all n: (1) $r_\mathcal{A}(n+1) \in a_{\tau_n}(r_\mathcal{A}(n))$, and (2) $a_{\tau_n} = \mathcal{P}_\mathcal{A}(l_{i_n})$ (where a_{τ_n} is the unique action of $\mathcal{A}$ implementing τ_n.)

Proof of (1): By definition, $(c, l_j) \in a_{\tau_n}(\mathsf{C}(n), l_{i_n})$ iff $c \in \tau_n(\mathsf{C}(n+1))$ and φ_j holds at c. But by construction, $\mathsf{C}(n+1) \in \tau_n(\mathsf{C}(n))$ (since this configuration was obtained by applying

τ_n to $\mathsf{C}(n)$) and $\varphi_{i_{n+1}}$ holds at $\mathsf{C}(n{+}1)$). Hence, $(\mathsf{C}(n{+}1), l_{i_{n+1}}) = r_{\mathcal{A}}(n{+}1) \in a_{\tau_n}(\mathsf{C}(n), l_{i_n}) = a_{\tau_n}(r_{\mathcal{A}}(n))$.

Proof of (2): The action taken at time n by $\mathcal{A}$ is the (only) action that implements the transition τ_n that $\mathcal{B}$'s action at time n implements. Recall that $\mathcal{P}$ was obtained by adding a pair (φ_j, τ_m) whenever a_{τ_m} is assigned to l_j. Hence, it must be the case that $\mathcal{P}_{\mathcal{A}}(l_{i_n}) = a_{\tau_n}$, which is what we wanted to show. ▮

Theorem 4.5: *If Task* $= O_m(\{\varphi_1, \ldots, \varphi_k\})$ *and* $\mathcal{A} = \langle L, \mathit{Actions}\rangle$ *is K-capable of* $\{\varphi_1, \ldots, \varphi_k\}$ *in* $\mathcal{I}[I, \mathcal{A}]$ *for* I t_0*-consistent with Task, then* $\mathcal{A}_{+m}$ *can perform Task from* I_{+m}.

Proof: With slight modification to accommodate the m additional control bits, the proofs of Lemma 7.1 and Theorem 3.17 generalize to this case. ▮

Theorem 4.8: *If Task* $= O_m(\{\varphi_1, \ldots, \varphi_k\})$*, Task is elastic, and* $\mathcal{A} = \langle L, \mathit{Actions}\rangle$ *can learn* $\{\varphi_1, \ldots, \varphi_k\}$ *in* $\mathcal{I}[I, \mathcal{A}]$ *for some* t_0*-consistent* I*, then* $\mathcal{A}_{+(m+1)}$ *can perform Task from* $I_{+(m+1)}$.

Proof: First, recall that by Theorem 4.5, if $\mathcal{A}$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$, it can perform *Task* with the aid of m additional bits from I_{+m}. Thus, an m-bit SKBP Pg exists that performs *Task*.

In order to show that $\mathcal{A}_{+(m+1)}$ can perform *Task*, we must provide a program for it that performs *Task*. Intuitively, this is done as follows: we describe an agent similar to $\mathcal{A}$ that is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ from I. Consequently, we know that this agent has an implementation $\mathcal{P}$ of Pg that performs *Task* from I_{+m}. We let $\mathcal{A}$ execute this implementation. However, this implementation is not defined on all states of $\mathcal{A}$. In particular, it is not defined on those state in which $\mathcal{A}$ does not know any of the conditions $\{\varphi_1, \ldots, \varphi_k\}$. In those states, we let $\mathcal{A}$ learn these conditions. After learning one of these conditions, $\mathcal{A}$ will reach a local state on which the given implementation is defined. Notice that this approach requires $\mathcal{A}$ to switch between $\mathcal{P}$ and the learning subroutines. Thus, an additional bit is needed to keep track of whether $\mathcal{P}$ is being executed or the learning program, and m additional bits are needed to execute $\mathcal{P}$ itself; $\mathcal{A}_{+(m+1)}$ has these additional bits. Finally, although learning may take a while, because the task is elastic, we do not mind the detours the learning subroutine may cause.

We proceed as follows. Let $\mathcal{P}_L$ be the protocol for $\mathcal{A}$ that learns $\{\varphi_1, \ldots, \varphi_k\}$ in $\mathcal{I}[I, \mathcal{A}]$. Let $\mathcal{A}' = (L', \mathit{Actions}')$, where

- $L' = \{l \in L \mid \mathcal{I}[I, \mathcal{A}], l \models K\varphi_1 \vee \ldots \vee K\varphi_k\}$
- $\mathit{Actions}' = \{a \circ \mathcal{P}_L \mid a \in \mathit{Actions}\}$ where $a \circ \mathcal{P}_L(c, l) \stackrel{\text{def}}{=} \mathcal{P}_L(a(c, l))$.

Notice that $\mathcal{A}'$ is well defined, and implements the same transitions that $\mathcal{A}$ implements. This follows from the fact that for every global state s in $\mathcal{I}[I, \mathcal{A}]$, it is the case that $\mathsf{proj}_{\mathit{config}}\mathcal{P}_L(s) = \mathsf{proj}_{\mathit{config}}(s)$. (Here is where we are using the fact that a learning program has no side effects on the configuration.) Moreover, we know that $\mathsf{proj}_{\mathit{local}}\mathcal{P}_L(s) \in L'$, since $\mathcal{P}_L$ is a learning program for $\mathcal{A}$. Moreover, $\mathcal{A}'$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$ in $\mathcal{I}[I', \mathcal{A}']$, where $I' = \cup_{s \in I}\mathcal{P}_L(s)$. (Notice that $\mathsf{proj}_{\mathit{config}}(I) = \mathsf{proj}_{\mathit{config}}(I')$, and so I' is t_0-consistent with *Task*.) In order to see this, note that the set of global states that arise in $\mathcal{I}[I', \mathcal{A}']$ is a subset of the global states that arise in $\mathcal{I}[I, \mathcal{A}]$. This follows from the fact that (1) $L' \subseteq L$, (2) $I' \in \mathcal{I}[I, \mathcal{A}]$, and (3) for all $a' \in \mathit{Actions}'$ and global states s in $\mathcal{I}[I, \mathcal{A}]$, we have $a'(s) \in \mathcal{I}[s, \mathcal{A}]$. Next, we note that, by definition, if $\mathsf{proj}_{local}(s) \in L'$, then $\mathcal{I}[I, \mathcal{A}], s \models K\varphi_1 \vee \ldots \vee K\varphi_k$. Because $\mathcal{I}[I', \mathcal{A}'] \subseteq \mathcal{I}[I, \mathcal{A}]$ and the formulas φ_j are propositional, we conclude that $\mathcal{I}[I', \mathcal{A}'], s \models K\varphi_1 \vee \ldots \vee K\varphi_k$ as well.

We have shown that $\mathcal{A}'$ is K-capable of $\{\varphi_1, \ldots, \varphi_k\}$. Therefore, $\mathcal{A}'_{+m}$ has an implementation $\mathcal{P}'$ of Pg that performs *Task* from I_{+m}. We define a protocol $\mathcal{P}$ for $\mathcal{A}_{+(m+1)}$ that acts as follows: when the first bit is 0, it emulates the behavior of $\mathcal{P}'$ whenever possible. When this is not possible, it switches the first bit to 1, indicating that the learning program should be executed, and it executes the learning program. Once it finishes the execution of the learning program, it switches the first bit back to 0 and resumes emulation of $\mathcal{P}'$:

- If $l \in L'$ and $d_1 = 0$, then $\mathcal{P}(0, d_2, \ldots, d_{m+1}, l) = \mathcal{P}'(d_2, \ldots, d_{m+1}, l')$.
- If $l \notin L'$ and $d_1 = 0$, then $\mathcal{P}(0, d_2, \ldots, d_{m+1}, l) = Set(d_1 = 1)$.
- If $l \in L_T$ and $d_1 \neq 0$, then $\mathcal{P}(1, d_2, \ldots, d_{m+1}, l) = Set(d_1 = 0)$.
- If $l \notin L_T$ and $d_1 \neq 0$, then $\mathcal{P}(1, d_2, \ldots, d_{m+1}, l) = (\mathcal{P}_L(l), Id_{m+1})$ where $(\mathcal{P}_L(l), Id_{m+1})$ is identical to $\mathcal{P}_L(l)$ on $\mathcal{E} \times L$, and Id_{m+1} is the identity assignment on $d_1, \ldots, d_{m+1}$.

As should be apparent from its definition, every execution of $\mathcal{P}$ from $I_{+(m+1)}$ corresponds to some execution of $\mathcal{P}'$ into which some finite subsequences are inserted, corresponding to the (terminating) executions of $\mathcal{P}_L$. We know that every execution of $\mathcal{P}'$ from I_{+m} is in *Task* and that *Task* is elastic. We conclude that every execution of $\mathcal{P}$ must also be in *Task*. ∎

Acknowledgment: We are grateful to Bruce Donald for numerous insightful discussions, comments, and suggestions, and to Nir Friedman for comments on previous drafts. This work was partially supported by AFOSR and NSF grants AF F49620-92-J-0547 and IRI-9220645.

References

[BK78] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc 19th Symp. Found. Comp. Sci.*, pages 132–142, 1978.

[BLMS] R. I. Brafman, J. C. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *Journal of the ACM.* To appear.

[BS95] R. I. Brafman and Y. Shoham. Knowledge considerations in robotics and distribution of robotic tasks. In *Proc. Fourteenth International Joint Conference on AI*, 1995.

[Bud75] L. Budach. On the solution of the labyrinth problem for finite automata. *EIK*, 11:661–672, 1975.

[Can89] J. F. Canny. On computability of fine motion plans. In *Proc. of the 1989 IEEE Int. Conf. on Rob. and Aut.*, pages 177–182, 1989.

[CM86] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[DJR93] B. R. Donald, J. Jennings, and D. Rus. Information invariants for cooperating autonomous mobile robots. In *Proc. of Int. Symp. on Robotics Research*, 1993.

[Don94] B. R. Donald. On information invariants in robotics. *Artif. Intell.*, 72(1):217–304, 1994.

[Erd94] M. Erdmann. Understanding actions and sensing by designing action-based sensors. In *IEEE ICRA Workshop on Design*, 1994.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[Hin62] J. Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, NY, 1962.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, 1990.

[Lat91] J. C. Latombe. *Robot Motion Planning*. Kluwer Academinc Publishers, Boston, 1991.

[Moo85] R. C. Moore. A formal theory of knowledge and action. In *Formal Theories of the Common Sense World*, 1985.

[Rei79] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, pages 421–427, 1979.

[Ros85] S. J. Rosenschein. Formal theories of knowledge in AI and robotics. *New Generation Comp.*, 3:345–357, 1985.

[RS85] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. 25th IEEE Symp. on Found. of Comp. Sci.*, pages 144–154, 1985.