Draft version without citations.

ES FORUM BACKGROUND PAPER FOR GROUP 2

An Ontological Perspective on Interactive Task Learning

Charles Rich
Worcester Polytechnic Institute

1 Introduction

What does it mean to "know how to do" something?

This is a question about the ontology of task knowledge, i.e., the abstract form, nature and organization of this knowledge. Answering this question is a logical prerequisite to questions about how to learn or teach such knowledge. Although our ultimate goal is identify the frontiers of the current state of the art, we will organize this paper around the main ideas and trends in current and past work on task knowledge. We are not, however, undertaking a comprehensive literature review; we will cite only representative examples in each topic area. Open questions, unsolved problems and controversies will be broken out in italics, as with the opening question above.

Another underlying premise of this presentation is that learning and teaching are best viewed as kinds of collaboration, i.e., the teacher and learner are two participants in a collaborative interaction in which the shared goal is increase the learner's abilities. Furthermore, learning and teaching are often naturally interleaved with other kinds of collaboration, such as delegation and supervision.

Finally, because discussions of ontology have a tendency to become very abstract, we will attempt whenever possible to illustrate ideas using a shared task example, which we describe in the next section.

1.1 An Example Task: Tire Rotation

Rotating the tires on a car is a task commonly performed in an automotive service center (see left of Figure 1). It entails unscrewing the four or five lug nuts on each wheel, removing the wheels and then remounting them on different hubs, according to a rotation pattern, such as back-to-front or cross-over (left rear goes to right front, etc.). The overall task involves many individual steps, with repeated subsequences (such as screwing or unscrewing all the nuts on a particular hub), alternatives (the rotation patterns), and the use of a tool (lug wrench or power wrench). Mohseni et al. have recently taught a PR2 robot (see right of Figure 1) a greatly simplified laboratory version of this task using a combination of demonstration and instructions.

Figure 1: Human tire rotation and state-of-the-art robot example.

1.2 Levels and Types of Knowledge

To start, it is useful to think about the knowledge involved in interactive task learning as stratified into the three levels shown in Figure 2. The discussion in the body of this paper will be organized according to this breakdown.

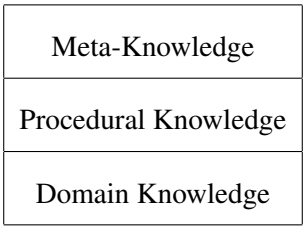

Figure 2: Levels of knowledge.

The bottommost level, *domain knowledge*, is the general knowledge about the world (or parts of the world) that pervades all aspects of the tasks being learned. This knowledge can sometimes be assumed to be already shared between the teacher and student, but it may also need to be explicitly taught. For example, any physical manipulation task, such as tire rotation, requires a commonsense understanding of everyday physics, such as the fact that an object will fall if you let go of it. Domain knowledge is used in both of the levels above it.

The middle level, *procedural knowledge*, comprises most of what is usually focussed on when teaching tasks, such as the ordering of steps, preconditions, task hierarchy, and so on. For example, the first step in removing a wheel is to unscrew the lug nuts.

The topmost level, *meta-knowledge*, is knowledge about the procedural knowledge.[1] For example, knowing how long it typically takes to rotate the tires on a car is meta-knowledge about the tire rotation procedure.

Finally, in addition to these three knowledge levels, an independent categorization to keep in mind is *symbolic* versus *non-symbolic*[2] knowledge. For example, the cross-over tire rotation pattern is an example of symbolic procedural knowledge, whereas the motion trajectory for unscrewing a lug nut is an example of non-symbolic knowledge. An example of non-symbolic domain knowledge would be the maximum weight carrying of a robot. Symbolic and non-symbolic knowledge can also be intermixed, such as in a parameterized motion trajectory with semantic constraints.

2 Domain Knowledge

Domain knowledge is the foundation of all task learning, other than the most limited form of mimicry. The type and extent of the required domain knowledge varies, of course, with the task (or more typically, collection of related tasks) that are being learned. For physical tasks in the everyday world (such as tire rotation or household tasks), the domain knowledge starts with commonsense physics, which includes knowledge of:

- *Gravity* - how objects support other objects, fall, etc.
- *Motion* - how objects move in free space and slide, roll, or bounce in contact with other objects
- *Materials* - how solid objects bend or break in response to squeezing or stretching or other manipulations; how liquids flow, pool and fill containers
- *Causality* - how actions on one object result in changes to other objects via connections between them

[1] Formally speaking, meta-knowledge could also include knowledge about domain knowledge, but persuasive examples do not come to mind.

[2] One might alternatively characterize the distinction as discrete versus continuous.

Can commonsense physics be built into robots, and if so, will that make it easier to teach them physical tasks?

As one moves into more specialized tasks, the domain knowledge also becomes more specialized. For example, the domain knowledge underlying cooking includes specialized knowledge about state changes associated with specific temperatures (such as boiling, freezing, and burning). In addition to commonsense physics, tire rotation requires some specialized knowledge about the appropriate forces for screwing and unscrewing lug nuts. A lot of such domain knowledge is non-symbolic.

Not all tasks are physical. Consider, for example, teaching arithmetic or teaching a virtual agent how to perform online tasks on your behalf. In these cases, the domain knowledge is more abstract and mathematical, such as the application program interfaces (API's) for online services.

Formal logic is a useful way thinking about symbolic knowledge, even if it is not the form in which the knowledge will actually be used in practice. From this point of view, domain knowledge provides the primitive object types (and perhaps some designated object instances), predicates, and functions, which are used for specifying the procedural and meta-knowledge, as discussed below.

What is the appropriate use of formal logic in research on interactive task learning?

3 Procedural Knowledge

Procedural knowledge is the knowledge required to execute tasks (which does not include being able to teach, learn or reason about them). This knowledge has been studied most intensively in the artificial intelligence subfield of planning. The following are the key elements of procedural knowledge:

- primitive actions
- ordering
- conditionals
- parameters
- constraints
- hierarchy
- motion trajectories

3.1 Primitive Actions

The most basic element of procedural knowledge is the *primitive actions*. It is important to note that the notion of "primitive" is contextual—what is primitive in one situation may not be primitive in another. For example, in teaching a robot how to rotate tires, unscrewing a lug nut and putting it down may be a primitive action, whereas for a different robot/teacher, there are two separate primitives actions: unscrewing and putting down.

Primitives are essentially symbols. However, primitive does not mean predefined or builtin. A particular primitive may be directly associated with a precompiled motor (or cognitive) program, but it also may be necessary to learn how to execute the primitive, in which case the primitive is ultimately a mixture of symbolic and non-symbolic knowledge.

How can/does the notion of primitive actions apply to "continuous" activities such as ballroom dancing or dribbling a basketball?

3.2 Ordering

The next most basic element of procedural knowledge is *ordering*. The most minimal form of plan is thus a sequence of primitives. Ordering knowledge also commonly takes the form of a partial order. For example, in tire rotation, the order in which you unscrew the lug nuts does not matter, but all the unscrew actions must precede unhanging the wheel.

What are the natural kinds of parallel and overlapping execution models appropriate for interactive task learning?

3.3 Conditionals

In order to achieve a formally complete computational system, all that needs to be added to primitives and ordering is *conditionals*. Conditionals can take many forms and can be used in different ways, but basically they are all about changing which actions are executed depending on the state of the world.

The most common form of conditional is a simple if-then-else, which specifies two alternative primitives (or sequences of primitives), depending on the value of some boolean condition. For example, in tire rotation, you can use either a lug wrench or power tool to tighten the nuts, depending on whether the power tool is available.

Also commonly used are *preconditions*, which when false, block the execution of a specified primitive or sequence. This is particularly useful when actions are partially ordered, because there may be something else that can be done while waiting for the precondition become true. For example, after squirting oil on a rusty nut, the unscrew action on that nut is blocked until 5 minutes have passed; meanwhile, other nuts can be unscrewed.

A less commonly used form of conditional is a maintenance condition, which is a condition that is expected to hold throughout the execution of a long-running action; if the condition ever becomes false, the action is supposed to be terminated. For example, the pneumatic power supply is expected to be maintained to the power lug nut tool throughout execution of the screw or unscrew action.

Conditions need not make only binary choices. For example, imagine a sorting task in which the color of the object is supposed to match the color of the sorting bin, where there are more than two colors.

Conditionals are a key place where domain knowledge is used in procedural knowledge, because the domain knowledge specifies the relevant and perceivable properties of the world that are tested in the conditions. For example, it is domain knowledge that specifies that the position of the wheels on the hubs is relevant to the tire rotation task, and not the color of the tires. To frame this in formal logic terms, the domain knowledge specifies the predicates that are used in the conditions.

What are the natural kinds of uncertain and probabilistic conditional models appropriate for interactive task learning?

3.4 Parameters

Using only what has been discussed above yields a very limited form of procedural knowledge, because the actions can only be applied to specific objects in specific settings—think of programs without inputs. To achieve generality and reusability, most procedural knowledge also includes *parameters*. For example, the procedural knowledge of how to rotate the tires on a car can be applied to any car. Thus the car (and implicitly all of its parts, such as the hubs, wheels, etc.) is

an input parameter of the task. Input parameters also often have restrictions on what values can be provided. These are very much like preconditions. For example, the input to the tire rotation task must be a four-wheeled vehicle.

In addition to input parameters, it is also common for procedural knowledge to include output parameters. Generally speaking the outputs of a task are considered to be the objects whose properties are changed or that are created during the execution of the task. For example, the output of the unscrew action is the nut, because its location has changed.

How can procedural knowledge be parameterized automatically?

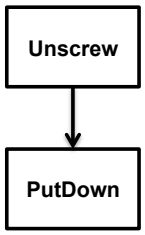

Figure 3: Example data flow constraint.

3.5 Constraints

Knowledge of parameters is important, because it enables the specification of *constraints* between the input and output values of different actions.

The simplest form of constraint is equality. For example, if you are painting a matching set of chairs, the color input parameters of all the paint actions would be constrained to be the same.

An equality constraint between an output parameter of an action and an input parameter of a subsequent action[3] is called *data flow*, as shown in Figure 3. The arrow in this example specifies that the output of the unscrew action (the nut) becomes the input to the putdown action. Data flow is an aspect of the causal structure of a procedure. Causal structure is important knowledge for, among other things, recovering from errors.

How is causal structure used in interactive task learning?

Constraints other than equality can also be part of procedural knowledge. For example, in a robotics procedure, for stability, the weight of an object placed on top of another object may be constrained to be less than the weight of the first object.

3.6 Hierarchy

Hierarchy is important because humans deal with complex tasks by breaking them down into subtasks. For example, Figure 4 shows one way to break down tire rotation into five levels of hierarchy. The tree notation in this figure is called a hierarchical task network (HTN), which has been widely used in artificial intelligence. The fringe of the tree specifies the sequence[4] of 64 primitive actions required to rotate the tires on a car (some repeated structure is elided in the figure to save space). Notice that there are multiple instances of the same type of primitive, e.g., Unscrew, with different parameters, i.e., different nuts.

The non-primitive nodes in the tree are called *abstract* tasks. The toplevel abstract task in this example is RotateTires. Other abstract tasks at intermediate levels of the hierarchy include UnscrewHub (unscrewing all the lug nuts on one wheel) and UnhangHub (removing the wheel from the car and putting it down). Although knowing these abstract tasks is not fundamentally necessary in order to execute tire rotation, it is important for two reasons: reuse and communication.

Regarding reuse, not only are there multiple occurrences of the same type of primitive in this task hierarchy, there are also multiple instances of the same type of abstract task. For example,

[3]It is logically possible, but unusual, to constrain two output parameters to be the same.

[4]Technically, in this case not a sequence, but a partial order, since some actions, such as unscrewing lug nuts on a single wheel, are unordered

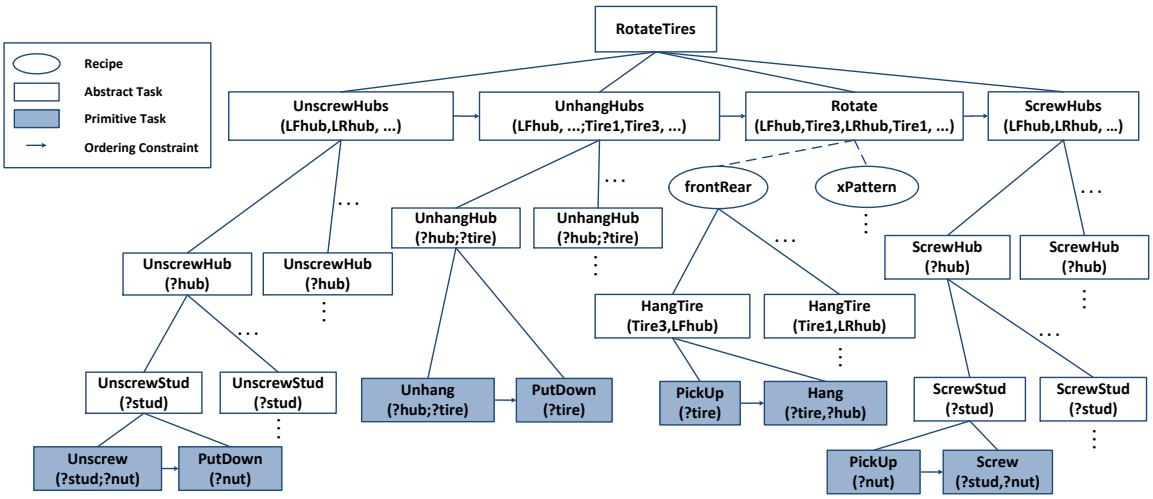

Figure 4: Example of hierarchical procedural knowledge for tire rotation.

UnscrewHub, UnhangTire, HangTire, and ScrewHub each appear four times (once for each wheel on the car). Furthermore, it is common for the same abstract tasks to be reused in other toplevel tasks in the same domain. For example, these four abstract tasks are also steps in changing a flat tire.

Abstract tasks also provide a richer communication vocabulary than just the primitives for collaborative interactions, including teaching and delegation. For example, if there is a breakdown, rather than saying "Pickup nut step 49 failed," one can say "Pickup nut failed while unscrewing the first wheel." Similarly, one can delegate an entire abstract task, rather than specifying the sequence of primitives, e.g., "now, please unhang all the tires."

Conditionality in hierarchical task networks is usually conceptualized in terms of decomposition choices, called *recipes*. For example, notice that Figure 4 is a tree with two kinds of nodes. The primitive and abstract tasks discussed above are rendered as rectangles. Each abstract task is decomposed into subtasks, each of which may be abstract or primitive. The oval recipe nodes in the tree indicate alternative decompositions.[5] For example, in Figure 4 there are two alternative recipes for the Rotate task: frontRear and xPattern, corresponding to the two standard patterns for rotating tires on a car: front wheels to rear wheels on the same side, or left-front to right-rear, etc. (The figure does not show the constraints that specify the details of these two patterns.)

The condition that determines whether a particular recipe can be chosen is called an *applicability condition*. Like the other conditions discussed in Section 3.3, applicability conditions test the state of the world and are built on domain knowledge. At execution time, if the applicability condition of more than one recipe for a given abstract task is true, then the procedural knowledge allows any of the applicable recipes to be chosen.

What is the best way of teaching recipe applicability conditions?

Like abstract tasks, recipes also serve an important communication function. For example, it is efficient to simply say,"Let's use the x pattern."

[5]Such trees are also called *and-or* trees, where the abstract tasks are *and* nodes, because all of their children are executed, and the recipes are *or* nodes, because only one of their children is chosen.

Finally, a cautionary note: the hierarchy in Figure 4 is not the only way to break tire rotation down into subtasks. For example, some people might insert an extra abstract task (say, RemoveHubs) grouping together UnscrewHubs and UnhangHubs. Someone else might not group Unscrew and PutDown into UnscrewStud. Even the choice of primitives cannot be assumed to be universal. For some people, Pickup and Screw may be a single primitive, rather than a sequence of two primitives. This adds a lot of challenges to communication and collaboration, beyond even the problem of different people using different words for the same task concept (see Section 4).

3.7 The Policy View

In reinforcement learning, which is increasing being applied to robotics, procedural knowledge is viewed somewhat differently than above. Instead of embedding the predicates that check the state of the world in conditional structures on sequences of actions, a single *policy* function, π, is used that maps from (all possible) states of the world, $\mathcal{S}$, to the possible actions, $\mathcal{A}$:

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

The policy view and the conditional plans view are mathematically equivalent in the sense that each can be mechanically translated into the other. However, they are very different in terms of the affordances they provide for human interaction. The policy view offers only a single monolithic function, π, to discuss, teach, correct, etc., whereas the conditional plans view exposes much more structure for interaction.

There has also been work on learning parameterized policies and hierarchical policies.

Can reinforcement learning algorithms be applied directly to conditional plans?

3.8 Motion Trajectories

All of the kinds of procedural knowledge discussed above are essentially symbolic. However, there is clearly also a kinesthetic dimension to "knowing how to do" a task. For example, as Figure 5 suggests, there are only certain motion paths through space that work to unhang a tire. Unlike the action sequences and hierarchies discussed above, this knowledge is in the form of continuous functions. At its simplest, such knowledge is a single motion path through 3D space, e.g., implementing a primitive action such as Unhang. More complicated versions of such knowledge may be in the form of regions in 3D space that constrain the motion, or trajectories/regions in the higher-order joint-configuration space of the manipulator. Sometimes it is also important to know the appropriate forces to be exerted at various points in the trajectory, for example, how much to tighten the lug nuts.

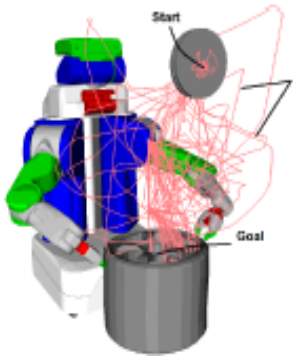

Figure 5: Motion trajectories for unhanging a tire.

What is the relationship between symbolic and continuous procedural knowledge and how can they be learned together?

4 Meta-Knowledge

Meta-knowledge is knowledge *about* other knowledge. In this section, we review some key categories of knowledge about procedural knowledge that are important for learning/teaching and collaboration generally:

- capabilities/tools required
- postconditions
- failure recovery
- duration
- difficulty
- uncertainty
- pedagogical knowledge

The preceding discussion of procedural knowledge assumes the agent performing the task is *capable* of executing all of the primitives. But this is not always true. In fact, a common motivation for collaboration is that one particular agent alone does not have all of the necessary capabilities. Knowledge of which primitives a particular agent is capable of performing (or restrictions on the parameters on the primitives) is thus an important category of meta-knowledge. For example, if a human and a robot are collaborating on tire rotation, perhaps the robot is capable of performing all of the screw/unscrew primitives, but anything involving the tire must be done by the human, because the tire is too heavy for the robot to lift. The availability of a particular tool, such as a lug nut or power wrench for tire rotation, is a closely related kind of capability meta-knowledge.

Should capabilities meta-knowledge include preferences (for different types of tasks)?

Actions do not always succeed. Knowing how to decide whether an action succeeded or failed is a kind of meta-knowledge about the action. This is called the *postcondition* of the action.[6] Both primitives and—in the case of hierarchical tasks—abstract tasks can have postconditions. Like the other conditions discussed in Section 3.3, postconditions test the state of the world and are built on domain knowledge. For example, the postcondition of the Pickup primitive is that the input object is in the agent's hand/manipulator. This condition could fail, for example, if the agent has not properly grasped the object before it moves its arm.

In what kinds of procedures is it most important to know postconditions?

Action failure introduces the issue of *failure recovery*. One of the differences between a novice and an expert in a particular task is that the expert knows more about how to recover from specific failures. For example, suppose Unscrew fails because the lug nut is frozen. An expert knows to squirt some penetrating oil on the nut and let it sit for 5 minutes and then try again. Failure recovery knowledge is obviously of great practical importance, and in some cases, can be much larger than the basic knowledge of how to do the task when everything goes well.

What is the role of generalized failure recovery knowledge?

When planning for tasks in the future, especially in the context of teaching or delegation, it is very useful to have estimates of the *duration* and *difficulty* of the tasks. For hierarchical tasks, one might have independent estimates for abstract tasks, or derive them by combining estimates for the primitives.

What is a natural metric with which to estimate task difficulty?

All of the types of meta-knowledge discussed above are subject to *uncertainty*. As a simple example, estimated task duration may be plus or minus 10%. More significantly, even the success of a task may be uncertain, for example, due to sensing difficulties. To return to the Pickup example

[6]Unlike preconditions, which are needed to control execution as discussed in Section 3.3, postconditions are not strictly needed to execute a procedure if everything goes well. This is why we are categorizing postconditions as meta-knowledge here.

discussed above, what if the robot's hand does not incorporate a sensor to detect whether it is holding something? In that case, in order to proceed intelligently with the task, it would be helpful to know the a priori probability of Pickup succeeding. Or what if the sensor is unreliable? It would then be helpful to know the probability it actually succeeded, based on the sensed information.

Is probability the best approach to expressing uncertainty for interactive task learning?

Finally, knowing how to do something is not the same knowing how to teach it. A good teacher has extra *pedagogical* knowledge about a task, such as appropriate fading and scaffolding strategies, the mistakes students typically make, and so on. Like error recovery knowledge, this pedagogical knowledge can be larger than the basic knowledge of how to do the task.

How do teachers learn pedagogical knowledge?

5 Conclusion

The discussion above has focused on ontology to the exclusion of many other important issues related to task knowledge. One of the most obvious such issues for interactive task learning is the use of natural language. Although we observe above that in hierarchical tasks, the recipes and abstract tasks (as well as the primitives) provide an important communication "vocabulary," we mean the concepts, not the particular words or phrases that are used. For example, we don't expect anyone to spontaneously say "UnhangHubs" as the second toplevel step of tire rotation. Understanding what people actually say is a major research challenge. Similar natural language challenges apply for all the types of knowledge discussed here.

Finally, we have as much as possible skirted the issue of knowledge representation in the discussion above. You have to know what the knowledge is before you think about how to best represent it for particular computational purposes.