A Cognitive Task Analysis of Rapid Procedure Acquisition from Written Instructions

Pat Langley PATRICK.W.LANGLEY@GMAIL.COM
Institute for the Study of Learning and Expertise, Palo Alto, California 94306 USA

Howard E. Shrobe HES@CSAIL.MIT.EDU
Boris Katz BORIS@CSAIL.MIT.EDU
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA

Abstract

In this paper, we examine the problem of learning complex procedures from instructional text. We present a cognitive task analysis that introduces theoretical constraints, considers candidate representations for activities, and decomposes the problem into subtasks that involve syntactic processing, semantic interpretation, and procedure construction. We also discuss possible uses for the acquired knowledge and outline some approaches to the empirical evaluation of cognitive systems that learn procedures from text. We do not present an implemented system, but we analyze the task in enough detail to guide future research on this important topic.

1. Background and Motivation

Many technical activities require execution of complex procedures, which people have traditionally had to learn through long, arduous training. Artificial intelligence has great potential for developing systems that assist humans in carrying out such procedures and even for automating them entirely, but their content must first be acquired. There have been two classic responses to the latter challenge. One involves encoding the necessary knowledge bases manually; this has produced many successes, from expert systems for diagnosis to intelligent agents used in air-combat exercises, but the approach is time consuming and expensive. Another involves learning control policies from traces of expert behavior or trial-and-error runs in simulators; this has generated robust controllers for land and air vehicles, as well as impressive game players, but this scheme requires far too many training cases and usually produces uninterpretable expertise. We need a more effective approach to overcoming the knowledge acquisition bottleneck.

Because many procedures are already described in manuals and other written documents, a natural response is to acquire knowledge about them by reading and interpreting these sources. There has been considerable work on learning by reading, but it has focused on extracting simple facts and concepts from texts, not on mastering procedural content. We want to extend this idea to acquiring complex procedures from instructional text. For example, the US Navy puts substantial resources into training personnel on the maintenance and repair of many types of complex equipment. One

worthwhile objective is an intelligent system that reads the manual for each such device, extracts and stores the procedures it describes in a standard format, and later accesses this knowledge as needed to assist engineers in their maintenance activities. Similar examples abound in everyday settings, from following cooking recipes to operating microwave ovens to repairing outboard motors.

In this paper, we present a cognitive task analysis (Newell & Simon, 1972) of this problem, identifying the main representational and processing issues but not reporting an implemented system. Task analyses occur *before* one collects observations about the behavior of human subjects and typically examines different ways that one might tackle the problem at hand, including which problem space to search and ways to explore it effectively. In the next section, we review previous AI work on expert-level behavior, after which we discuss theoretical constraints that guide our analysis. We then propose hierarchical task networks as a representation for procedures and consider three stages involved in extracting such cognitive structures from instructional text. After this, we outline a performance component that could use this acquired knowledge and suggest some approaches to empirical evaluation. We will see that each stage of the learning task has been studied separately, so the paper's main contribution is to clarify how an integrated cognitive system can address them in tandem to acquire and use procedural knowledge efficiently and effectively.

The sections that follow make a number of theoretical contributions to the literature on cognitive systems. These include postulates about the representation of complex procedures, the intermediate structures that lead to them, and the mechanisms that underlie their acquisition from text, including syntactic analysis, semantic inference, and procedure construction. We also present empirical claims about the ability to learn such content from text, including robustness to the omission of omitted actions, conditions, and effects, to the presence of extraneous steps not relevant to goals, and to different levels of abstraction in the instructions. Because we do not present an implemented system, we cannot yet test these hypotheses, but they offer clear priorities for future research.

2. Prior Research on Expert Behavior

Artificial intelligence and cognitive systems have demonstrated repeatedly their ability to automate complex activities that humans find difficult to master. Moreover, there is general agreement that this ability depends centrally on storing and accessing expertise about the application domain. Differences of opinion have arisen primarily about how AI systems should encode this expertise and how to acquire such content. It is worth reviewing briefly two competing responses to these issues and some examples of their successful application.

The first approach, which rose to prominence during the 1970s and 1980s, is commonly known as *expert systems* (Giarratano & Riley, 2018; Waterman, 1986). These typically express expertise as a collection of rules that codify explicit knowledge about the domain. After interviewing experts or reading documents, human 'knowledge engineers' create domain content manually, much as a traditional programmer but using modular formalisms that map well onto human knowledge elements. The expert systems paradigm has led to many fielded artifacts that have improved productivity or replaced humans. Prototypical applications have dealt with classification tasks, but others have involved complex procedural knowledge. Examples of the latter include interactive tutors for mathematics (Anderson et al., 1995), synthetic pilots for tactical air combat (Jones et al., 1999), and compelling virtual characters for interactive drama (Mateas & Stern, 2005). The main drawbacks

of this approach are that construction and maintenance of knowledge bases can be time consuming, error prone, and expensive, much as with traditional software development.

These difficulties led directly to research on machine learning, which initially attempted to automate the creation of knowledge bases, usually sets of rules, from a moderate number of training cases. However, despite early successes at producing fielded systems (Langley & Simon, 1995), the research community gradually turned to statistical methods that required large training sets. The increased availability of such data has led to compelling applications, especially ones that involve classification. The absence of large data sets for action-oriented domains has caused some researchers to use simulated environments for learning reactive policies by massive trial and error. This approach has produced impressive results in game playing (Clark & Storkey, 2015) and physical control (e.g., self-driving vehicles), but the induced expertise has generally been difficult to interpret and it has made little contact with human knowledge about the domains.

Recent research on interactive task learning has explored a middle path that tackles expertise acquisition by tutoring AI systems in natural modalities like language and sketches.[1] For example, Hinrichs and Forbus (2014) describe a system that learns concepts and moves for games from a combination of natural language and sketches. Similarly, Kirk and Laird (2014) report a program that acquires game knowledge from a sequence of text instructions and examples, while Sarathy et al. (2018) combine spoken language and video to learn skills for using everyday objects. More recently, MacLellan et al. (2018) offer an analysis of the paradigm that covers a wide range of instructional types, interaction patterns, and communication modalities. Interactive task learning has given impressive results on the incremental acquisition of procedural knowledge, but it relies on explicit tutoring that often starts at a high level and works its way downward, which can make the paradigm tedious and time consuming for human instructors.

An alternative approach, far more scalable, would take advantage of procedural content that has already been described in detail within technical manuals and similar documents. There has been some work on learning by reading (e.g., Friedman et al., 2017), but this has focused on extracting concrete facts from text rather than acquiring complex procedures. Moreover, most such techniques require training on a corpus of documents before they can be applied to new material (e.g., Liang, 2016; Mitchell et al., 2015). There have been some efforts on extracting procedures from text using statistical methods (e.g., Kiddon et al., 2015; Park & Nezhad, 2018), but they require thousands of examples to produce only simple structures, so we do not consider them a viable response to the problem. Nevertheless, there is clear potential for combining the rich representations of activity associated with interactive task learning and mechanisms for natural language processing to automate the acquisition of complex procedural knowledge from instructional text.

3. Theoretical Constraints on Procedure Acquisition

In later sections, we provide a cognitive task analysis of learning complex procedures from instructional text. However, such an analysis should not occur in a vacuum; it should take into account widely adopted theoretical postulates about representation, performance, and learning in cognitive systems. These assumptions will serve constraints on the design of any computational artifact that

1. This emphasis distinguishes it from earlier work on programming by demonstration (Lieberman, 2001).

addresses the task. The ideas we present will be familiar to many readers, but taken together, they serve as the foundation for an integrated approach to procedure learning that has never been tried.

We will start with postulates about representation, since any tenets about cognitive structure are logically prior to ones about the processes that operate on them.

- *Procedural knowledge consists of **modular elements**, only some relevant to particular problems.*

That is, knowledge about a complex activity is not a black box or opaque mathematical function, but rather a set of cognitive structures that can be composed as needed. These may take the form of rules, frames, skills, or graphs, but they are encoded as distinct modular elements. Moreover, only a subset of these structures will be needed for any given task.

- *Procedural knowledge is **relational** in that each element describes a configuration of entities.*

In other words, procedures typically involve multiple objects (e.g., an agent and a manipulated entity), and they incorporate details about those objects' configuration. A classic example is a spatial relation that must hold before one carries out an action (e.g., a compartment's door must be open before inserting an object) or that becomes true after application (e.g., the object is inside it).

- *Procedural knowledge is **causal** in that it describes effects of activities under certain conditions.*

Procedures specify how to alter the world, typically in terms of changes between relations that hold before their execution and ones that hold afterward. For example, an object will be outside a compartment before taking the action of inserting it and the same object will be inside the compartment afterwards. This maps directly onto everyday notions of action-oriented causality.

- *Procedural knowledge is **hierarchical** in that it decomposes complex activities into simpler ones.*

Humans do not view procedures as simple sequences of actions; they divide them into components that abstract away from details at high levels, yet can expand them further as needed. The language used to describe procedures reflects their hierarchical nature and simplifies their acquisition.

- *Procedural knowledge is **disjunctive** in that it specifies different ways to address the same task.*

Many procedures include alternative means to achieve the same ends. These may involve genuine choice or they may be appropriate under different conditions. Such disjunctions offer procedures substantially greater flexibility than is possible for simple sequences of actions and, when combined with hierarchical decomposition, they even support recursion.

- *An important source of domain content for procedural learning is **instructional text**.*

Despite the current popularity of statistical induction from large data sets, this scheme bears little resemblance to human learning (Langley, 2016), which benefits greatly from explicit instruction. Written manuals are a key source of such input and can serve as a repository of procedural information for intelligent systems to acquire and then carry out complex activities.

- *Syntactic processing of instructional text uses grammatical knowledge to **parse** sentences.*

Most AI research on language processing assumes that documents follow the grammatical rules of the language in which they are written. The parsing process associates parts of speech and lexical features with words, as well as identifying phrases that organize them into constituent structures. There is general agreement on this idea, but it is worth stating explicitly.

- *Semantic processing of such text uses domain knowledge and common sense to extract* ***meaning****.*

The purpose of reading is not to parse sentences but to infer their meanings, which are encoded in terms of case frames (e.g., agent, action, object) or a similar structural formalism. A typical assumption is that meaning extraction operates on the results of syntactic processing, which greatly constrains the possible interpretations. Some approaches handle syntax and semantics in parallel (e.g., Jackendoff, 2007), but we will adopt the more common sequential framework.

- *Procedure learning acquires* ***cognitive structures incrementally*** *from successive experiences.*

Most work on machine learning depends on batch processing of large training sets, often by modifying weights on statistical models. In contrast, we assume that the knowledge acquisition process is incremental, with each training experience leading to creation of new, modular mental structures, much as appears to happen in human learning.

- *Procedure learning is* ***abductive****, explaining training items by linking them to existing structures.*

Incremental learning does not create new cognitive structures in isolation. The acquisition process connects them to elements acquired previously, using stored contents to make sense of the later experiences. This involves a variety of abduction that reasons over individual training cases to produce plausible candidates for addition to the knowledge base.

- *Procedure learning is* ***cumulative****, in that later acquisition builds on structures created earlier.*

Just as abductive learning draws on existing knowledge to explain new experiences, so do the acquired structures build on these background elements in a cumulative manner. For instance, a high-level procedure might refer to low-level ones that were learned previously, using them as scaffolding to make the acquisition process more tractable.

- *Learning is* ***rapid****, in that it acquires complex procedures from reasonably few training items.*

Unlike the statistical methods that dominate the literature on machine induction, human learning occurs very rapidly. As noted earlier, each training experience leads to new cognitive structures, especially when these are directly stated in written instructions. This is a crucial feature for AI systems that aim to learn complex procedures from text in an effective and scalable manner.

None of these theoretical ideas will be contentious to the cognitive systems community, but they are rare enough in the recent AI literature that it seems important to state them explicitly and to clarify the reasons for adopting them. Taken together, they place strong constraints on approaches to learning procedural knowledge from textual instructions. In addition to these theoretical commitments, we will also adopt a number of simplifying assumptions to make our analyis more tractable. These include premises that the learner knows the:

- Goals or objectives for each top-level procedure that it is asked to acquire;
- Actions to which the instructions refer, including their effects under particular conditions;
- Relational concepts used to describe the states that arise during the procedure's execution;
- Syntactic knowledge relevant to the instructions, which include grammatically correct sentences;
- Case frames that describe possible semantic relations and the constraints on these structures.

We will also posit that the learned procedures are primarily qualitative in character, being appropriate to guide physical actions but not to automate them fully, as would be necessary in robotic settings. Moreover, we will limit each set of instructions to a few paragraphs that exclude surrounding material, rather than attempting to automatically segment extended documents that describe many distinct activities. These assumptions further constrain the problem of acquiring procedures from written instructions and suggest promising avenues to approach this objective.

4. Representing Complex Procedures

Before we can describe our approach to learning procedures, we should first explain their representation, which must satisfy a number of constraints. The formalism must specify sequential behavior over time, possibly with some type of parallelism, and procedures must accept arguments or parameters that support generality. These structures must also satisfy the representational principles given earlier. They must be relational, in that they describe configurations of entities, and causal, in that they specify the effects of activities under given conditions. Procedural knowledge must also be organized in hierarchical terms to support levels of abstraction and allow conditional disjunctions that state different ways to handle the same task. Most important, the notation must be modular, encoding procedures as a collection of elements that can be acquired separately. Classical programming languages satisfy some of these criteria, but they are not inherently relational or causal, they are not sufficiently modular, and their complicated syntax does not make them easily learnable.

For this, we need a formalism that supports small, modular elements that can be composed into larger structures. A natural choice is a hierarchical task network or HTN (Nau et al., 2003), which comprises a set of *methods*, each of which specifies a task, conditions under which the method applies, and a sequence of subtasks or primitive actions. Some variants associate effects with each method, giving a causal interpretation. Each task and condition consists of a predicate and zero or more arguments, such as *(unscrew ?cap ?body)* and *(screwed-on ?cap ?body)*. These relational predicates may primitive or defined elsewhere in rules similar to Prolog clauses. The ability of methods to invoke subtasks leads directly to hierarchical programs. Different clauses may share a task name, thus supporting disjunctive behavior, and a method may call on itself, giving recursion. Despite this power, HTNs specify complex procedures in a modular way that encourages incremental learning (e.g., Langley et al., 2009). They are typically used for knowledge-guided plan generation (Nau et al., 2003; Shrobe, 2002), but also support hierarchical reactive control (Choi & Langley, 2018).

Nau et al. (2003) have shown formally that hierarchical task networks have greater expressive power than classical plans, but we will not claim they are the only representational framework that meet our criteria. For example, production systems (Klahr, Langley, & Neches, 1987) offer equivalent representational power, appear to map well onto human knowledge structures, and are modular enough to support incremental learning. The main difference is that they require multiple rules to encode a single HTN method, but research with ACT-R (Anderson, 1993) and Soar (Laird, 2012) has shown they can encode complex procedures. Other frameworks are less appropriate for our needs. Declarative representations, such as knowledge graphs and logic programs, emphasize monotonic inference rather than activities that alter the environment to achieve goals. They can be adapted to procedural settings, but do not lend themselves to them in the same way as HTNs and production

systems. Dynamic Bayesian networks, partially observable Markov decision processes, and probabilistic programs support action over time and offer modularity, but their emphasis on uncertainty would be a distraction for our needs, as such details rarely arise in instructional documents.

For these reasons, we propose to adopt hierarchical task networks for our work on rapid procedural learning, although we will draw on a number of useful extensions. These include augmenting the notation for methods to include expected effects (Li et al., 2012), associating methods with goals they achieve (Shivashankar et al., 2012), and specifying temporal constraints on subtasks (Stracuzzi et al., 2009). The latter provides support not only for hierarchical physical activities, but for ones that involve coordination among multiple agents who must time their actions to achieve joint goals, which can be especially important in applied settings. Hierarchical task networks have also been extended to include probabilistic outcomes, but, again, these are not required to specify the basic structure of procedural knowledge and we will not focus on them here.

5. Learning Complex Procedural Knowledge

Our cognitive task analysis of learning procedures from documents decomposes the problem into three subtasks. These include parsing text to produce a syntactic analysis of its content, transforming these structures into semantic representations, and using the results to extend an existing hierarchical task network. In this section we examine each of these subtasks in turn, including possible ways to address them. After this, we consider how we could combine them to learn complex procedures from instructional text.

5.1 Syntactic Processing

Because we want to learn procedural knowledge from instructional documents, the first step is to process that text syntactically. We can start this parsing task as:

- *Given:* Sentences that describe complex procedures and associated concepts;
- *Given:* Grammatical knowledge about the structure of sentences in a language;
- *Given:* Lexical knowledge about the words that appear in the language's sentences;
- *Generate:* Grammatical parses of these sentences that specify their syntactic structure.

A typical parse assigns a part of speech (e.g., adjective, noun, verb) to each word, associates features with them (e.g., singular, past), and groups words into constituent phrases (e.g., noun, verb, and adjectival phrases). The standard unit of syntactic processing is a single sentence, so that parsing instructional text requires interpreting a sequence of connected sentences. Written text differs substantially from everyday speech because it generally follows the rules of grammar and lacks the disfluencies that occur in spoken language.

Consider the instructions in Table 1 for preparing hard boiled eggs, which are paraphrased slightly from the Web site *https://www.allrecipes.com/*, This simple recipe not only specifies a sequence of steps, but also includes conditional statements about when to halt subactivities. The text does not explain the details of peeling the eggs after they have been cooked, but it clarifies the character of typical procedural instructions. Syntactic processing would produce a parse for each sentence in this paragraph. For example, it would identify the second line as a conjunction of two

Table 1. A simple recipe for preparing hard boiled eggs, which we have paraphrased slightly from the version at the Web site *https://www.allrecipes.com/*.

To make Hard Boiled Eggs,
Place eggs into a saucepan and pour cold water to cover the eggs.
Place the saucepan over high heat.
When the water just starts to simmer, turn off heat.
Cover pan with a lid, and let the pan rest for 17 minutes.
Drain the hot water and pour cold water over eggs.
Drain the cold water.
Refill the saucepan with additional cold water.
Allow the eggs to stand until they are cool.
Wait for 20 minutes.
Peel eggs under running water.

---

clauses connected by 'and'. The fourth line specifies a condition, 'When the water just starts to simmer', which refers to an event that follows placing the pan over high heat. The conditioned passage, 'turn off heat, cover pan with a lid, and let the pan rest', is an imperative statement, with the reader as its elided subject and a conjunctive verb phrase in which 'turn off', 'cover', and 'let rest' are the verbs. Other sentences have similar complexity. The instructions follow the rules of English grammar and contain few ambiguities.

Syntactic parsers have a long history in AI, dating back to the 1960s (e.g., Green et al., 1961). Over five decades of research have produced many different approaches to this task, but the most popular current methods rely on statistical models that are trained on a large corpus of annotated sentences. Although these are prevalent in the academic community and dominate the recent literature, training corpora can take considerable time to collect and substantial effort to annotate. Moreover, this process must be repeated each time one encounters a new application domain. A promising alternative is to use a broad-coverage English parser that relies on hand-entered grammatical and lexical knowledge. Despite widespread rhetoric about the benefits of statistical parsers that require training, such non-statistical manually-created systems are competitive and they have been used in successful applications (e.g., McShane, Nirenburg, Beale, & Johnson, 2012).

One such system is START (http://start.csail.mit.edu/), a mature syntactic parser developed by Katz (1988, 1990, 1997) and his colleagues over three decades, that covers a wide range of English constructions and that includes a very broad lexicon. For a given sentence, the system generates a set of nested triples of the form *[subject relation object]*, where the *subject* and *object* may themselves be triples (e.g., *[John drove [car in garage]]*. These ternary expressions provide a versatile syntax-oriented notation for language that encodes syntactic and lexical features and that highlights significant relations. These compact structures support the efficient storage, matching, and retrieval of linguistic content. START handles reference resolution within and across sentences, but it delays decisions about prepositional phrase attachments for later processing. The default system produces

Table 2. Syntactic triples generated by the START system from the first three sentences in the sample recipe. Each triple encodes a pairwise relation between items, with the numbers denoting instances of generic types. We have not included any triples that describe number, tense, or similar syntactic details.

---

1. Place eggs into a saucepan and pour cold water to cover the eggs.

[you place-4 eggs-1]
[place-4 into-6 saucepan-1]
[pour-5 has_purpose-7 cover-6]
[you pour-5 water-1]
[you cover-6 eggs-1]
[water-1 has_property-8 cold]

2. Place the saucepan over high heat.

[you place-7 saucepan-1]
[place-7 over-9 heat-1]
[heat-1 has_property-10 high]

3. When the water just starts to simmer, turn off heat.

[turn_off-1 when-1 start-2]
[you turn_off-1 heat-1]
[water-1 start-2 simmer-3]
[water-1 simmer-3 null]
[start-2 has_modifier-2 just]

---

a single parse, but it can be tuned to generate multiple interpretations if desired. START incorporates substantial knowledge about English grammar and it has been used in many applications, from mobile phone interfaces (Katz et al., 2007) to question-answering systems (Katz et al., 2006) to design assistants (Shrobe et al., 2015). Moreover, the software does not require training on an annotated corpus for new domains, which makes it a natural choice for our research effort.

Table 2 shows the syntactic triples generated by START from the first three sentences of the recipe in Table 1. Most terms come directly from the text, but a few are inferred from grammatical knowledge. For instance, some of the triples include *you* as their first element because it can be assumed when a sentence lacks an explicit subject. Similarly, the third triple states that the purpose of the *pour* activity is to *cover* the eggs, since this is a natural interpretation of *to* in the first sentence. However, these are shallow inferences that remain closely tied to the sentence, and we intend this only as an illustrative example. START is certainly not the only parsing system that we might use to process instructional text, but we favor ones that already have broad coverage of English and that do not require training on a large corpus of sentences. Nevertheless, any parser will occasionally encounter unfamiliar words and constructions that its knowledge base does not encompass. We should extend the system so that, in such cases, it asks a human to provide details about parts of speech or elements of grammar that will let it complete the parse. START makes it reasonably

easy to enter new syntactic constructions and lexical items needed for a particular application. This approach is similar in spirit to work on interactive task learning, but focuses on local repairs to syntactic knowledge rather than the acquisition of entire procedures.

5.2 Semantic Processing

Parsing is merely the first step in learning procedural knowledge from instructional text, as it provides only the grammatical structures of sentences. We must still transform these into a semantic representation that encodes their meanings. We can state this problem as:

- *Given:* Parses of sentences about complex procedures and associated concepts;
- *Given:* Knowledge that maps syntactic structures onto meaning representations;
- *Given:* Knowledge about relations between words and their connection to concepts;
- *Generate:* A set of linked semantic structures that specify the sentences' meanings.

As with parsing, the typical unit of meaning extraction is a single sentence, but this is complicated by the fact that later passages refer to entities or activities introduced in earlier ones. Resolving synonyms for the same object is an obvious example, but other cross-sentence connections are important as well. Another key challenge is that text seldom makes everything explicit, so that semantic processing must fill in the gaps using some form of plausible inference.

Let us return to the earlier instructions for cooking hard boiled eggs specifically first half of the second line. Here semantic processing should infer 'place' is an *action*, the reader is its *agent*, 'eggs' is its *object*, and the 'saucepan' is the *destination*. One must also realize that the location of the second action, 'pour', is also the saucepan. Some passages map onto descriptions of states that occur during the procedure, while others correspond to activities that transform these states. Fillmore-style (1968) case frames (e.g., with roles for agent, instrument, object, destination) are natural candidates for encoding larger-scale structures. Different frames can share arguments to provide a connected network of states and activities, similar in spirit to an abstract plan. Table 3 shows some case frames that might result from the first of the parsed sentences in Table 2. Generating such structures requires knowledge about the types of relations and actions available, as well as constraints on their arguments. For example, the action *place* includes an animate agent (you), an inanimate object (eggs), and a target destination (the pan). Such frames can also introduce details that were implicit in the text, such as that the eggs had a location before being moved into the pan.

Semantic interpretation has been studied less extensively than parsing, but there has been considerable work on the topic for at least four decades. One example is Winston and Holmes' (2018) Genesis, a story understanding system which they have already integrated with START. Genesis invokes the parser to determine the grammatical structures for sentences, then transforms the results into semantic representations that specify categories, relations, actions, and events. The latter process relies on generic rules that match against patterns of syntactic forms (encoded as START triples) and translates them into Fillmore-style (1968) case frames (e.g., agent, beneficiary, instrument, object) that group the triples' contents into larger-scale structures. Moreover, the system uses WordNet (https://wordnet.princeton.edu/) to map synonyms onto canonical encodings, giving the same internal representation for different sentences with equivalent meanings. Together, these mechanisms turn the syntactic structures produced by START into semantic ones that can be used

Table 3. Semantic case frames for relations and actions that might be inferred from a subset of the syntactic triples in Table 2. These do not include start and end times for relations that change during the activity.

(eggs ^id eggs-1) (saucepan ^id saucepan-1) (person ^id you)
(place ^id place-4 ^agent you ^object eggs-1 ^source ?f ^destination saucepan-1)
(pour ^id pour-5 ^agent you ^object water-1 ^destination saucepan-1)
(cold ^id property-8 ^object water-1)
(covers ^id covers-1 ^container water-1 ^object eggs-1)
(in ^id in-1 ^object eggs-1 ^location saucepan-1)

for summarization, question answering, and other language-related tasks. Recent work by Yang and Winston (2018) has even extended the framework to acquire constrained procedures from stories that describe how people repair malfunctioning phones.

Genesis is not the only semantic interpreter that we might use to transform parses into meaning structures, but it offers the clear advantage of already being linked to START.[2] Even so, the existing system will sometimes encounter words or syntactic outputs for which it lacks case frames, and we should extend it to handle such situations by asking a human to provide the missing translation rules. We should also augment the representation of case frames that describe activities to incorporate conditions and effects, which can then serve in the specification of procedures. This should include generic knowledge about common physical processes, such as opening containers, filling them, and heating them, as analyzed by Schmolze (1986) for many everyday activities. Finally, despite the availability of semantic constraints, there will reman instructions that have ambiguous meanings (e.g., where to attach prepositional phrases), so we should extend Genesis to support multiple interpretations that it resolves later as more information becomes available.

5.3 Constructing Procedures

Once the learner has generated a semantic interpretation of an instructional unit like a sentence or short paragraph, it can transform this meaning into one or more cognitive structures that encode procedural knowledge. We can specify this problem as:

- *Given:* A description of some procedure's initial conditions, sequential steps, and final effects;
- *Given:* Existing knowledge about primitive actions, other procedures, and associated concepts;
- *Generate:* A set of HTN methods that encode this procedure and relate it to existing knowledge.

The simplest training items will contain content about individual HTN methods or conceptual definitions. However, instruction manuals present a sequence of statements that, together, specify a complex set of linked procedures. The learner must deal with these incrementally, producing new cognitive structures that serve as background knowledge for later training elements. Any system

2. Of course, some theories of language understanding (e.g., Jackendoff, 2007) assume a deeper integration of syntactic and semantic processing. These suggest an alternative approach to extracting meaning from instructional text.

that learns hierarchical task networks or equivalent representations must address three issues. First, it must determine the hierarchical structure of the network, i.e., how to decompose high-level tasks into subtasks, subsubtasks, and so on. Second, it must identify the conditions and (optional) effects for each method, that is, when the method applies and what results it produces. Third, it must decide when distinct methods provide alternative solutions to the same high-level task. Research on HTN learning has always addressed these three challenges, although responding in different ways.

Fully explicit instruction makes these issues much simpler, in that it states directly the subtasks for a method *M*, the conditions and effects for *M*, and the task that *M* accomplishes. This is the approach taken by recent work on interactive task learning (MacLellan et al., 2018), which specifies each procedural element in such terms. This paradigm typically assumes that hierarchical activities are specified from the top down, but one could use similar techniques to communicate procedures from the bottom up. Either way, the result is a hierarchical but grounded set of methods for carrying out complex tasks. We will support such explicit instruction, which basically involves direct translation from a semantic representation extracted from text to a hierarchical task network that can be used for planning, execution, or other activity. Most of this ability should be handled by Genesis or an analogous module for meaning interpretation.

However, instructional documents are seldom so precise and complete. They may specify a sequence of actions without indicating how to organize them into a hierarchy, as in many recipes. They may omit essential conditions for a method's application, they may fail to mention some effects, and descriptions of initial situations or goals may be incomplete. Real-world instructions, like those in equipment manuals, may even leave out key steps that the writer believes will be obvious to readers. Finally, when a document presents descriptions for different methods, they may not state that they are relevant to carrying out the same high-level task. The recipe discussed earlier is explicit about major steps, but it also leaves out details about intermediate situations and about actions' conditions and effects. We need a system for learning complex procedures that handles each of these types of omissions in a robust manner.

One relevant approach, involving the ICARUS architecture (Choi & Langley, 2018), learns HTN methods from knowledge about primitive actions and from sample solutions which transform an initial state into another that satisfies a goal specification (Nejati et al., 2006; Li et al., 2009). This uses a version of means-ends analysis to reason backward from the goal description *G*, showing how each step either achieved some aspect of *G* or enabled another action that ultimately led to *G*. Because means-ends decomposes a problem into subproblems recursively, it produces a hierarchical breakdown of the task, with each subtree of this decomposition producing a new method. The method's conditions include those state elements needed to apply its component actions and its effects combine the effects of these components; two methods are given the same task name when they achieve the same subgoal. ICARUS creates new hierarchical methods as needed from each sample solution, so learning is incremental, cumulative, and rapid. The architecture can execute these structures reactively or use them for planning on novel problems.

Again, ICARUS' mechanism is not the only approach to constructing hierarchical task networks from sequences of steps, but some form of analytic or abductive learning seems necessary for rapid acquisition. Nevertheless, the existing approach requires a number of extensions to make it more robust, including the addition of new abilities for:

- Inferring omitted actions by using means-ends analysis, with limited search, to fill in the gaps between disconnected portions;
- Imputing missing conditions on high-level methods or primitive actions in an instructional sequence by noting that a previous step has no other purpose;
- Filling in the effects of high-level methods or primitive actions that were omitted from instructions but that are needed for the conditions of later steps;
- Ignoring incidental or irrelevant actions in the instructional text that do not contribute to achieving the goal description; and
- Handling instructions that refer to abstract methods rather than concrete actions by expanding the former if already defined or delaying learning until they have been acquired.

We can view each of these extensions in terms of constrained search to links disconnected portions of solution traces, for which means-ends analysis offers direct support. Augmenting the ICARUS learning module along these lines would complement the extensions to START and Genesis that we outlined earlier to let them jointly acquire complex procedures from realistic textual instructions.

5.4 System-Level Issues

Recall that our objective is a computational artifact that reads instructional text, extracts its procedural content, and encodes the latter in a hierarchical task network for later use. We have decribed the component subtasks that arise in this setting, but we will also need to combine them into an integrated cognitive system for learning procedures from documents. Fortunately, previous research has already accomplished some of this integration. We have seen that Genesis inputs parses of text from START and transforms them into frames that describe situations, events, and activities, some of which characterize multi-step procedures. However, integrating an HTN learning mechanism with such modules seems more challenging. Work in this area assumes that training cases specify an initial situation, a sequence of primitive actions, and one or more goals they achieve. Information about the component actions' conditions and effects must be available as background knowledge that goes considerably beyond traditional case frames, so we must find some way to connect the two notations. This linkage appears to be a key step in the incremental acquisition of hierarchical procedures from written instructions.

We also want an approach to procedural learning that is flexible. If the instructions make explicit the hierarchical structure of some activity, then it should translate this content directly into new HTN methods. If the text instead presents only example sequences of low-level actions, then the system should use means-ends analysis to infer and create the hierarchical structure. The same should hold for methods' conditions and effects, which may be described explicitly, remain implicit in the text, or be specified only partially. Nevertheless, the system will sometimes lack the knowledge about state predicates, primitive actions, or high-level tasks on which analysis depends. In such cases, it should ask a human for clarification about the meaning of terms or the conditional effects of an activity, although this should happen only as a last resort, as we want to avoid the detailed tutoring associated with the paradigm of interactive task learning. These requirements suggest that the integrated system include a number of distinct modes from which it selects depending on instructional content and its prior knowledge.

6. Using Learned Procedures

Procedural knowledge does not exist for its own sake. Intelligent agents, human or otherwise, draw on cognitive structures to *perform* some task, and showing use of knowledge in this manner is the standard way to demonstrate the benefits of learning, regardless of its details. To show that an approach to learning complex procedures is effective and useful, we must commit to a particular performance task and provide a corresponding performance system that can draw upon the acquired knowledge to address this problem. Only then can we show empirically that learning produces a desirable impact on behavior.

Procedural knowledge has many potential applications. The one receiving the most attention currently is reactive execution of physical devices like self-driving cars and autonomous drones. However, this would require linking hierarchical procedures with techniques for low-level sensing and control that are challenging in their own right. We might connect autonomous agents to simulated environments for urban driving, which would bypass the need for physical platforms, but the natural performance measures would still focus on fine-grained control, which is not our main objective. Another important application is training, since instruction depends on stored content to share with students, but this would require construction of an interactive tutoring system that raises other issues off the main line of our research. Similarly, we might use learned procedural knowledge to monitor human behavior to detect when personnel diverge from established practices, but this would again depend on low-level processing of video or other sensor streams that would delay progress on our core concerns.

A more promising approach is to focus on high-level execution of acquired procedures. Although most work on hierarchical task networks addresses plan generation (e.g., Nau et al., 2003), some research has used them for reactive execution (e.g., Choi & Langley, 2018), in some cases over learned knowledge. In this setting, we could provide the performance element with an initial situation, including objects and resources, and a task or set of objectives. The system would then retrieve relevant methods, decompose them as needed, and, upon reaching primitive actions, declare it was carrying them out, checking the situation that results in each case. Rather than interacting with a physical simulator, it would receive state information from a human overseer. This arrangement would support scenarios that involve unreliable actions and information-gathering activities, thus demonstrating the reactive character of procedure execution while avoiding the need to handle low-level perception and control required in other contexts.

7. Evaluation of Procedural Acquisition

We should also consider ways to evaluate the effectiveness of a system that acquires complex procedures from instructional text. One issue concerns candidate testbeds that we might use to this end. Ideal domains would include a variety of distinct procedures, each of which already has accompanying instructions. Two promising domains include:

- *Meal preparation*, for which written recipes are readily accessible (e.g., *https://sites.google.com/site/kzhaigh/dinner-co-op*) and which often require substitutions because specified ingredients are unavailable. This application offers a good starting point, not only because recipes are brief, simple, and self contained, but also because they are familiar to people throughout society.

- *Repair of shipboard equipment*, such as the engine, bilge pumps, and drive trains, for which written manuals are available and for which the possibility of different malfunctioning components requires testing and conditional response. This testbed is more challenging than the first domain, with longer and more complex procedures.

Access to subject matter experts is also important, both for access to relevant technical manuals and for the background knowledge needed to interpret them. Fortunately, many people are highly skilled in cooking, and there are enough who have experience with shipboard repair that finding them should pose little problem.

We can evaluate the learned procedures in two complementary ways. First, we would manually develop, with advice from subject matter experts, hierarchical task networks from a representative sample of instructions for each domain. As described earlier, these would specify complex procedures as a set of methods, each comprising a task, conditions, subtasks, and effects. Dependent measures for these basic studies would include precision and recall over the learned procedural steps, both primitive and higher level, for each document as compared to the target knowledge base. The aim would be to demonstrate the ability to reconstruct known procedures from instructions with high precision and recall of their elements. We would compare not only the hierarchical organization of learned procedures with target structures, but also the accuracy of their arguments, conditions, and effects. Moreover, for each document, we would record the number of times the system required human intervention (e.g., asking questions about missing syntax or case frames), along with both human and CPU time required to process the instructions.

Second, we can devise a suite of test scenarios, again with input from subject matter experts, for each domain. Each test problem would include an initial situation, a task or set of goals, and, optionally a set of facts that would remain unobserved until appropriate tests are carried out. For example, scenarios for meal preparation would specify a set of available ingredients, cooking utensils, and the type and amount of dish to be prepared. For repair of shipboard equipment, they would describe initially known behavior (e.g., water is collecting in bilge), a set of goals (e.g., no water in bilge), available spare parts and test devices, and hidden malfunctions that testing can reveal. The performance measure here would be whether the execution module can use the learned knowledge to achieve the goals for given scenarios, as reported by Nejati et al. (2016) in their work on HTN learning. Incorrect or incomplete knowledge can lead to false steps or dead ends that fail to solve the problem at hand. Typically, the system would either succeed or fail on a given task, although we could award partial credit for those with multiple goals.

Such experiments would test a number of empirical claims by examining the effect of independent factors on task performance. These might involve testing hypotheses about graceful degradation of learned behavior as one increases the:

- Number of component actions that are omitted from the text but that are needed to achieve the associated goals, which will reveal the ability to leap over instructional gaps;
- Number of conditions and effects that are omitted from methods or actions mentioned in the instructions, which will indicate the learner's robustness to such elisions;
- Frequency of extraneous actions that appear in the text but that are not necessary to achieve goals, which will measure the system's capacity to determine relevance; and

- Level of detail at which methods are described, which will demonstrate the learner's ability to process instructions at different degrees of abstraction.

These are all examples of scaling studies (Langley, 1996) that vary some factor which seems likely to affect task difficulty. We might also collect learning curves that report the rate of performance improvement as a function of the number of instructions processed. Early runs would focus on simple documents with paraphrased sentences to reduce the need for extensions to modules, while later ones would use longer sources with unmodified contents. Together, these studies should both demonstrate the ability to learn correct complex procedures from written text and to execute this knowledge to achieve goals in the domain.

8. Concluding Remarks

In the previous pages, we specified the task of learning complex procedures from instructional documents and presented a cognitive task analysis of this challenging problem. We proposed a number of theoretical constraints to guide research on the topic. These included the importance of representing activities in a modular, relational, causal, and hierarchical manner, which suggested hierarchical task networks as a promising formalism. After this, we examined three component tasks: syntactic processing of instructions to generate parses, semantic interpretation of these parses to produce case frames, and knowledge-guided transformation of these meaning structures into procedures. In addition, we reviewed three existing artifacts that address these subtasks and that could serve as elements in an integrated system. Furthermore, we considered some performance settings in which to demonstrate the learned knowledge, along with issues related to the experimental evaluation of implemented systems. The latter included likely dependent measures and some independent factors that would let us test hypotheses about the robustness of procedure acquisition.

Although a task analysis is the natural first step in tackling any difficult problem, we must also build on this start by implementing a computational artifact, demonstrating its behavior on illustrative scenarios, and running controlled experiments to test claims about its abilities. To this end, we plan to combine and extend existing software for syntactic parsing, meaning extraction, and analytical learning – specifically START, Genesis, and ICARUS – into an integrated cognitive system that acquires hierarchical procedures from instructional text. This new approach to knowledge acquisition provides an approach to creating content about goal-directed activities that is more automated than recent, complementary work on interactive task learning. Moreover, it offers an important and practical alternative to the two extremes of handcrafted knowledge bases and data-intensive induction, letting us sail safely between the Scylla and Charibdis that have dominated the straits of artificial intelligence for many years.

Acknowledgements

This research was supported by Grant N00014-20-1-2643 from the Office of Naval Research, which is not responsible for its contents. We thank Patrick Henry Winston, Zhutian Yang, Dylan Holmes, and Sue Felshin for useful discussions that influenced the ideas reported here.

References

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, *4*, 167—207.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, *48*, 25–38.

Clark, C., & Storkey, A. (2015). Training deep convolutional neural networks to play Go. *Proceedings of the Thirty-Second International Conference on Machine Learning* (pp. 1766–1774). Lille, France.

Fillmore, C. J. (1968). The case for case. In E. Bach & R. T. Harms (Eds.), *Universals in linguistic theory*. New York: Holt, Rinehart, & Winston.

Friedman, S., Burstein, M., McDonald, D., Plotnick, A., Bobrow, R., Cochran, B., & Pustejovsky, J. (2017). Learning by reading: Extending and localizing against a model. *Advances in Cognitive Systems*, *5*, 77–96.

Giarratano, J. C., & Riley, G. D. (2018). *Expert systems: Principles and programming* (4th Ed.). Pacific Grove, CA: Brooks-Cole Publishing.

Green, B. A. Wolf, A. K., Laughery, K., & Chomsky, C. (1961). Baseball, an automatic question-answerer. *Proceedings of the Western Joint Computer Conference* (pp. 219–224). Los Angeles.

Hinrichs, T. R., & Forbus, K. D. (2014). X goes first: Teaching simple games through multimodal interaction. *Advances in Cognitive Systems*, *3*, 31–46.

Jackendoff, R. (2007). A parallel architecture perspective on language processing. *Brain Research*, *1146*, 2–22

Jones, R. M., Laird, J. E., Nielsen P. E., Coulter, K., Kenny, P., & Koss, F. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, *20*, 27–42.

Katz, B. (1988). Using English for indexing and retrieving. *Proceedings of the First RIAO Conference on User-Oriented Content-Based Text and Image Handling* (pp. 313—333). Paris.

Katz, B. (1990). Using English for indexing and retrieving. In P. H. Winston & S. A. Shellard (Eds.), *Artificial intelligence at MIT: Expanding frontiers* (Vol. 1). Cambridge, MA: MIT Press.

Katz, B. (1997). Annotating the World Wide Web using natural language. *Proceedings of the Fifth RIAO Conference on Computer Assisted Information Searching on the Internet* (pp. 136–159). Montreal, QC.

Katz, B., Borchardt, G., & Felshin, S. (2006). Natural language annotations for question answering. *Proceedings of the Nineteenth International FLAIRS Conference* (pp. 303–306). Melbourne Beach, FL.

Katz, B., Borchardt, G., Felshin, S. & Mora, F. (2007). Harnessing language in mobile environments. *Proceedings of the First IEEE International Conference on Semantic Computing* (pp. 421–428). Irvine, CA: IEEE Press.

Kiddon, C., Ponnuraj, G. T., Zettlemoyer, L., & Choi, Y. (2015) Mise en Place: Unsupervised interpretation of instructional recipes. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 982–992). Lisbon, Portugal: ACL.

Kirk, J. R., & Laird, J. E. (2014). Interactive task learning for simple games. *Advances in Cognitive Systems*, *3*, 13–30.

Klahr, D., Langley, P., & Neches, R. (Eds.) (1987). *Production system models of learning and development*. Cambridge, MA: MIT Press.

Laird, J. E. 2012. *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Langley, P. (October, 1996). Relevance and insight in experimental studies. *IEEE Expert*, 11–12.

Langley, P. (2016). The central role of cognition in learning. *Advances in Cognitive Systems*, *4*, 3–12.

Langley, P., Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, *10*, 316–332.

Langley, P., & Simon, H. A. (November, 1995). Applications of machine learning and rule induction. *Communications of the ACM*, *38*, 55–64.

Li, N., Stracuzzi, D. J., Langley, P., & Nejati, N. (2009). Learning hierarchical skills from problem solutions using means-ends analysis. *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society*. Amsterdam.

Li, N., Stracuzzi, D. J., & Langley, P. (2012). Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems*, *1*, 109–126.

Liang, P. (2016). Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, *59*, 68–76.

Lieberman, H. (Ed.) (2001). *Your wish is my command: Programming by example*. San Francisco, CA: Morgan Kaufmann.

MacLellan, C., Harpstead, E., Marinier, R., & Koedinger, K. (2018). A framework for natural cognitive system training interactions. *Advances in Cognitive Systems*, *6*, 177–192.

Mateas, M., & Stern, A. (2005). Structuring content in the Façade interactive drama architecture. *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment* (pp. 93–98). Marina del Rey, CA: AAAI Press.

McShane, M., Nirenburg, S., Beale, S., & Johnson, B. (2012). Resolving elided scopes of modality in OntoAgent. *Advances in Cognitive Systems*, *2*, 95–112.

Mitchell, T. M., et al. (2015). Never-ending learning. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (pp. 2302–2310). Austin, TX: AAAI Press.

Nau, D., Au, T., Hghami, O., Kuter, U., Murdock, J., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Nejati, N., Langley, P., & Könik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the Twenty-Third International Conference on Machine Learning* (pp. 665–672). Pittsburgh, PA.

Park, H., & Nezhad, H. R. M. (2018). Learning procedures from text: Codifying how-to procedures in deep neural networks. *Proceedings of the 2018 World Wide Web Conference* (pp. 351–358). Lyon, France.

Sarathy, V., Oosterveld, B., Krause, E., & Scheutz, M. (2018). Learning cognitive affordances for objects from natural language instruction. *Advances in Cognitive Systems*, *7*, 135–156.

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems* (pp. 981–988). Valencia, Spain.

Shrobe, H. (Winter, 2002). Computational vulnerability analysis for information survivability. *AI Magazine*, *23*, 81–81.

Shrobe, H. E., Katz, B., & Davis, R. (2015). Towards a programmer's apprentice (again). *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (pp. 4062–4066). Austin, TX: AAAI Press.

Schmolze, J. G. (1986). Physics for robots. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 44–50). Philadelphia, PA: Morgan Kaufmann.

Stracuzzi, D. J., Li, N., Cleveland, G., & Langley, P. (2009). Representing and reasoning over time in a cognitive architecture. *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society*. Amsterdam.

Waterman, D. A. (1986). *A guide to expert systems*. Reading, MA: Addison-Wesley.

Winston, P. H. & Holmes, D. (2018). *The Genesis enterprise: Taking artificial intelligence to another level via a computational account of human story understanding* (CMHI Report No. 1). Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Yang, Z., & Winston, P. H. (2018). *The Genesis enterprise: Learning by asking questions and learning by aligning stories* (CMHI Report No. 3). Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.