Interactive Grounding and Inference in Instruction Following

Dario D. Salvucci (salvucci@drexel.edu)
Department of Computer Science, Drexel University
3141 Chestnut St., Philadelphia, PA 19104, USA

Abstract

Learning by instruction is one of the most common forms of learning, and a number of research efforts have modeled the cognitive process of instruction following, with many successes. However, most computational models remain brittle with respect to the given instructions and lack the ability to adapt dynamically to variants of the instructions. This paper aims to illustrate modeling constructs designed to make instruction following more robust, including (1) more flexible grounding of language to execution, (2) processing of instructions that allows for inference of implicit instruction knowledge, and (3) dynamic, interactive clarification of instructions during both the learning and execution stages. Examples in the context of a paired-associates task and a visual-search task are discussed.

Keywords: Instruction following; cognitive architectures; cognitive code; interactive task learning.

Introduction

Learning by instruction can be defined, in simplified terms, as the process by which a teacher provides a learner with instructions for a task and the learner follows the instructions to perform the task. The process of learning by instruction has been a focus of numerous cognitive-modeling efforts in past decades, such as those using ACT-R (e.g., Anderson et al., 2004; Salvucci, 2013; Taatgen & Lee, 2003; Taatgen, Huss, Dickison, & Anderson, 2008) and Soar (e.g., Howes & Young, 1997; Huffman & Laird, 1995; Lewis, Newell, & Polk, 1989). More recently, there has been an increased focus on interactive task learning through natural interaction with a human instructor (see Laird et al., 2017; Kirk & Laird, 2014).

This research on instruction following has primarily focused on skill acquisition and improvements with learning over time. In contrast, less attention has been paid to the translation of instructions to knowledge: while past efforts have generally included a basic version of the translation process, this aspect of the models is often brittle and dependent on a very particular specification of instructions. For example, the model described in Salvucci (2013) accepts simplified textual instructions and would break easily with only slightly different instructions. Some efforts have aimed to make such a process more robust—for instance, by utilizing a more flexible knowledge representation (e.g., Taatgen et al., 2008) or by relying on dynamic interaction with the teacher (e.g., Laird et al., 2017). Nevertheless, the robustness of instruction following and learning continues to be a challenge for modern cognitive models and architectures.

This paper explores several ways in which instruction following can be made more flexible and robust. Specifically, this work examines three areas for improving robustness: grounding of instruction language to knowledge representations, inference of implicit instruction knowledge, and dynamic interaction to clarify or augment instruction knowledge. The models here have been developed using the Think architecture (https://github.com/salvucci/think) which uses a cognitive-code approach (Salvucci, 2016) to embed the theories and mechanisms of cognitive architectures (primarily ACT-R: Anderson et al., 2004) into a modern programming language (in this case, Python). The following sections provide an overview of the modeling approach, focusing on the instruction interpretation and execution processes, and then discuss the three areas of improvement along with a description of the associated models.

Modeling with Cognitive Code

Before exploring the fuller model of instruction following, first we present a basic model of a simple task to illustrate the workings of Think's cognitive code and how it contrasts with traditional production-system cognitive architectures. The sample task examined here is the so-called paired-associates task (Anderson, 1981): the participant reads a word, tries to recall and type a digit associated with that word, and then reads the associated digit, eventually learning the word-digit pairings. Cognitive code allows for a clean separation of task and model code, each of them running on separate threads and interacting via elements of the environment (e.g., a desktop-computer display, keyboard, mouse, speakers, etc.).

Table 1 shows the simple (Python) code that implements the task itself. The code first clears the display and then presents the word stimulus on the display, then waits 5 seconds (as dictated by the experiment). If the participant keys in a response, the response is logged for correctness; otherwise, if no response is keyed, an incorrect response is logged. The trial ends when the code presents the associated digit and waits another 5 seconds. These steps are repeated for the various stimulus pairings and across trial blocks.

The cognitive code representing the cognitive model, in Table 2, is similarly straightforward. The first line directs the vision module to wait for a particular stimulus of type *word*, and when it is found, encodes the visual object in the `word` variable. Each line of code incurs a passage of virtual time that aligns temporally with the task code (as well as any other cognitive threads that may be running; see Salvucci & Taatgen, 2010). The next few lines attempt to recall a memory chunk that associates the word with its associated digit, and if successful, the model types the digit. Finally, the model encodes the visual digit and stores the association between word and digit in memory.

Because cognitive code is grounded in a modern programming language familiar to most programmers, learning to write models under this approach is much easier than with

Table 1: Paired-associates task code.

```
self.display.clear()
self.display.add_text(50, 50, word, isa='word')
self.wait(5.0)
if not self.responded:
    self.log('incorrect response')
self.display.add_text(50, 50, digit, isa='digit')
self.wait(5.0)
```

Table 2: Paired-associates model code.

```
visual = self.vision.wait_for(isa='word')
word = self.vision.encode(visual)
chunk = self.memory.recall(word=word)
if chunk:
    self.motor.type(chunk.get('digit'))
visual = self.vision.wait_for(isa='digit')
digit = self.vision.encode(visual)
self.memory.store(word=word, digit=digit)
```

production systems, which are not nearly as familiar to programmers today. Cognitive code also takes advantage of common programming idioms—for instance, returning `None` for a failed memory retrieval, compared to a more complex production-system method of handling such a failure. On the other hand, production systems offer some of their own benefits, such as a more flexible partial ordering of execution. Nevertheless, cognitive code aims to provide the major advantages of cognitive modeling—broadly speaking, accounting for and predicting cognitive, perceptual, and motor performance—to a wider programming audience with as low barriers as possible to getting started in the modeling process.

Instruction Interpretation and Execution

We now take the next step in our modeling, moving from single-task models (like the one above) to a more general model that takes instructions and can execute a variety of tasks. The model of instruction following proposed here can be characterized at the highest level in terms of two stages: interpretation and execution. The interpretation stage involves translating the given instructions into a mental representation that encodes the necessary cognitive, perceptual, and motor actions that combine to perform the desired task. The execution stage involves recalling each instruction and then actually performing the cognitive, perceptual, and/or motor actions involved. Our model also aims to account for a realistic passage of time through both stages—most notably, processing instructions step by step over time (as opposed to assuming an already-encoded full set of instructions in memory).

The interpretation stage is implemented in the model as follows. Each instruction is assumed to be spoken aloud by the teacher such that the model can, like an experiment participant, hear and process the information incrementally. (Alternatively, we could assume that each instruction is presented on-screen to the participant; the model would behave largely the same except for utilizing visual instead of aural channels.) The model then interprets each instruction by attempting to understand its meaning and converting it to an associated mental representation. Because of the real-time nature of how the model receives instructions, the decay in the architecture's memory system—again, based on ACT-R—necessitates some practice of these instruction chunks so that they can be properly recalled in the next stage.

For example, consider the instructions in Table 3 for the paired-associates task. When interpreting these instructions, the model starts with the first statement—'*To perform a task*'—and understands that what follows are instructions for this particular task. Then, the model translates each step of the instructions to one or more relations, implemented as ACT-R-like chunks—for example, `WaitFor(word)` or `If(Recall(digit, word), Type(digit))` for the first two steps in Table 3. Each of the chunks is boosted in memory to ensure later recall.

Table 3: Sample instructions for the paired-associates task.

To perform the task
Wait for a word
If you can recall the digit for the word, type the digit
Wait for a digit
Remember the word and the digit
Repeat

The execution stage then uses the stored mental representations to perform the given task. At each step, the model recalls the chunk(s) for that step and performs the actions associated with the step—for instance, the `WaitFor(word)` chunk would invoke the visual system in waiting for a visual stimulus, and when found, the model would note the encoded object as the *word*. In doing so, the model builds up a context such that it may use information later (such as when the word and digit need to be remembered together in the sample task).

While our description above might suggest that interpretation and execution are two discrete stages that occur one after another, in fact these two stages are often interwoven: partial instructions might be provided so that a learner can practice a subgoal of the task; the learner may forget certain instructions and need to refresh their memory; the learner may also realize that their mental representation is ambiguous or deficient in some way and need clarification during execution; and so on. Such examples will be expanded further in the next section.

Interactive Grounding and Inference

The above description of instruction following as interpretation and execution are quite general; however, a simple

straightforward implementation of these processes may yield a model that is very fragile with respect to the instructions. The predecessor to this work (Salvucci, 2013) focused on a model that could account for behavior across a wide range of tasks, and did not emphasize aspects of instruction flexibility; the model had only a minimal interpreter for the simplest pseudo-English instructions (e.g., '*Wait-for visual-change*'). At the same time, the general problem of natural-language understanding with respect to instruction following is of course an extremely difficult problem in its own right, and such a general model is not currently feasible. Thus, our primary aim is to develop a model that minimizes the most general natural-language challenge but still allows for as much flexibility and robustness as possible. We now describe severals ways in which we can generalize the previous approach, including incorporation of ideas from other efforts into a single integrated account of instruction following.

Instruction Grounding

One critical aspect of the interpretation stage can be characterized as instruction grounding in which the natural-language statements and their subcomponents are grounded to objects and actions in the real world. Several recent efforts in the Soar community in particular have made significant strides in this area (e.g., Lindes, Mininger, Kirk, & Laird, 2017; Mohan, Mininger, Kirk, & Laird, 2012). For example, the Lucia system incorporated into the Rosie agent (Lindes et al., 2017) provides grounding for simple objects (e.g., '*the green rectangle*'), prepositional phrases (e.g., '*the green square to the left of the blue square*'), and whole sentences; in doing so, Rosie can learn tasks (in this case, simple games) and uses the grounded knowledge to reason about and act upon the associated objects in the world.

We follow a similar approach here, interactively receiving the instructions in sequence and incrementally grounding each component. Consider the paired-associate instructions presented earlier in Table 3. The parser implemented in Think takes a natural-language phrase such as '*Wait for a word*' and builds a declarative memory chunk `WaitFor(word)` as a mental representation of the phrase. The concept of *word* is grounded to the next visual object that appears to the model, and the model will store the mapping from *word* to this object in the current context (equivalent to ACT-R's imaginal buffer; see Anderson et al., 2004). In other cases where a specific visual object is referenced (e.g., on a crowded screen), the model allows the (virtual) experimenter to "point out" visual information—for instance, hearing the phrase '*Read the letter*' while pointing at the object—which gives the model an associated visual point along with the verbal information (Salvucci, 2013). Later in the model's simulation run, actions such as *Wait for* or *Read* will be grounded to their respective psycho-motor actions during the execution stage.

Sometimes, ambiguity in grounding can arise when the same object is referred to by different words or phrases. For example, consider a case in which a teacher directs the learner to '*Wait for a digit*' and then later to '*Type the number*'. From the context, and in this case the lack of any other realistic interpretation, a human participant could understand the change and ground both *digit* and *number* to the same object, whereas a simpler model interpreter could not make this leap. The model here allows for the inclusion of potential synonyms in its declarative memory, which are assumed to be part of a person's general knowledge (not something learned during the task). When a term is discovered that cannot be grounded—e.g., '*Type the number*' when the model has not yet seen a *number*—the model checks for other potential interpretations within its existing context. In our example, the model will have already grounded the *digit*, and thus it can search for and find an interpretation whereby *number* and *digit* are the same object.

Table 4: Sample instructions for the visual-search task.

(a)	To perform the task Find the 'C' Move the mouse to it Click on it Repeat
(b)	To perform the task Find the 'C' Click on it Repeat
(c)	To perform the task Click on the 'C'

Another common and useful aspect of natural language for instructions arises in anaphora resolution, or more specifically, pronoun resolution. Table 4(a) provides sample instructions for a visual-search task in which the participant finds the letter 'C' among a set of distractors (such as the letter 'O'). Pronoun resolution—specifically here, resolving the meaning of the word *it*—allows for two major benefits: first, it allows for more natural expressions of the instructions of the part of the teacher; and second, it allows the model to ground multiple references to the same physical object (in this case, the same 'C' mentioned earlier in the instructions). Much like the model of Lindes et al. (2017), the model here builds up a representation context incrementally, first noting that there is an object 'C', and later noting that *it* must refer to this earlier object. Admittedly, pronoun and anaphora resolution are much more complex in the general case; however, even the straightforward method here covers many simpler cases and already nicely enhances the flexibility of the model's parsing and interpretation.

Instruction Inference

Beyond the language of the instruction steps, some of the variability from a teacher's instructions arises in inclusion or exclusion of the steps themselves. In particular, some steps

may be explicitly stated in one circumstance but only implicitly suggested in another; in the latter case, the model must infer any intermediate instructions or actions. Table 4 includes three alternative sets of instructions: (a) long-form instructions that explicitly direct the participant to *Find*, *Move* to, and finally *Click* on the desired target, plus an explicit *Repeat* step; (b) shorter instructions that skip the *Move* step; and (c) even shorter instructions that only direct the participant to '*Click on the 'C'*' without any other steps. In each case (and one might easily imagine further alternatives), we would expect the same behavior from the learner.

Our approach to this challenge represents a blend of two key ideas in earlier work. First, more recent models of instruction following developed in ACT-R (e.g., Taatgen et al., 2008) have encoded instructions along with a set of preconditions and postconditions, such that an instruction step may execute only when its preconditions have been satisfied, and its execution then results in postconditions that may in turn be needed by other steps. Second, the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987) has as one of its core principles the idea of resolving an impasse: when the next action cannot be easily determined, the architecture generates an impasse and creates a subgoal to resolve this impasse. The model borrows the spirit of each approach in the execution of instructions. Certain actions, such as clicking the mouse on an object, have a natural precondition, such as moving the mouse to that object. When attempting to follow such an action, if the precondition is not met, the model first tries to execute a subgoal that will resolve that precondition, which could potentially trigger another subgoal. In Table 4(c), the *Click* action requires the *Move* action, which in turn requires the *Find* action—and thus the single instruction '*Click on the 'C'*' triggers the same sequence of actions as the equivalent three steps in Table 4(a).

Along these lines, similar simple inferences could be made in other ways for these instruction sets. For example, Table 4(c) omits the final *Repeat* step, but it would be reasonable to assume that if a participant would continue to be presented with similar stimuli, they would infer this repeat on their own, and the model does the same.

The approach here is not as general as Soar's impasse mechanism, since it does not claim to be a general approach to subgoaling; the approach is more akin to the precondition/postcondition work of Taatgen et al. (2008), although here, conditions are not stored in the declarative memory chunks but instead embedded in the execution processing of the individual actions. Our approach could also be viewed as a basic form of backward chaining seen in other systems (e.g., Langley & Choi, 2006). On a larger scale, the interaction between teacher and learner (described shortly) may lead to even more complex scenarios—for example, a teacher could modulate instructions based on learner's expertise, common and shared knowledge, and so on, leaving the learner to infer simple steps or perhaps to derive more complex inferences between new pieces of knowledge.

Interactive Learning and Execution

As stated earlier, although we have mostly emphasized separate interpretation and execution stages to this point, the reality of instruction following is often much more complex, involving interaction between teacher and learner throughout the learning process. This idea has been a focus of the work on interactive task learning (see Laird et al., 2017), in which "the learner actively tries to assimilate the meaning of the instruction while performing the task, and learning occurs in conjunction with that task's performance." During the interpretation stage, a learner might stop when confused by an instruction and ask the teacher what is meant by that instruction. During the execution stage, the learner might realize that there is actually ambiguity where they did not anticipate (e.g., two words on the screen when looking for a *word*).

The current model is built with interactivity in mind, allowing for a stream of communication between the (simulated) teacher and the (model) learner. The teacher provides verbal instructions to the model, and the model performs the task over time—but at any stage, either of them may interact with the other to communicate questions or information (see the description of "communicative grounding" in Chai et al., 2018). For example, consider a situation for the paired-associates task in which the model remembers a *digit* but then is instructed to '*Type the number*'. As mentioned earlier, the model has one avenue to solve this ambiguity, namely in recalling *number* as a possible synonym of *digit*. But what if this synonym pair was not known to the learner, or is a distant semantic relation that could not be easily inferred (e.g., *digit* and *target*)? If no synonym is available, the model stops and asks the teacher a question such as '*Which is the number?*', and waits for a response to process using its audition module. When the response is given—e.g., '*the digit*'—the model remembers this association and uses it for future processing. (The association might even be forgotten if the memory chunk decays too much before its next use, which would trigger the model to repeat the question to the teacher.) In this way, the model gains additional ways to augment its understanding and clarify ambiguities and/or gaps in its knowledge.

Discussion

This paper has provided an overview of several ways in which computational models of instruction following can be made more flexible and robust with respect to variations in the instruction and learning process. Our discussion of the relationship to human data has been somewhat non-traditional for a cognitive-modeling effort: we have generally argued that human participants can adjust to these variations and then shown that the model can do the same. From a more traditional perspective, we can note that these models do indeed provide a reasonable fit to human performance. For example, Figure 1 shows the results of 10 simulations of the paired-associates model along with the results from human participants over blocks of trials (Anderson, 1981). The model fits the human data well for both correctness, $R = .98$, $RMSE = .08$, and

response time, $R = .99$, $RMSE = .17$, with ACT-R memory theory driving the predictions. In fact, for any of the variants of the instructions described here, the model results would be largely the same; small differences might arise due to extra cognitive processing of, for example, synonyms or interactive communication, but the qualitative behavior and fit of the model would not change in a significant way.

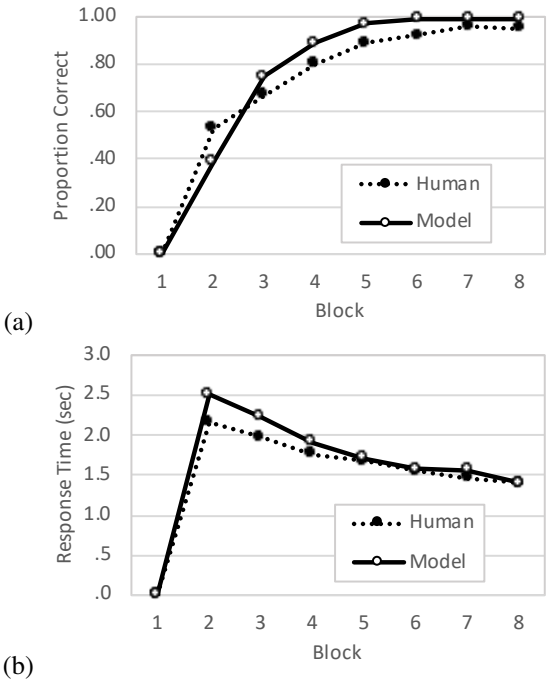

Figure 1: Human and model results for the paired-associates task showing (a) correctness and (b) response time.

Thus, unfortunately our current human data does not allow us to fully validate the techniques here. To address this challenge, our current work is part of a larger effort to develop an *undifferentiated agent* which is not specific to any one task but instead can take instruction and then perform a wide range of tasks (see, e.g., Salvucci, 2013). We are working toward applying such an agent to a battery of tasks, addressing various challenges along the way, especially with respect to the types of inference (or "gap-filling") that might be done more robustly with a fuller knowledge ontology and reasoning system. This effort aims to provide dual benefits of deeper understanding of human behavior and broader development of systems for practical applications such as synthetic teammates (e.g., Myers et al., 2018).

Acknowledgments

This work was funded in part by a grant from the Air Force Office of Scientific Research (#FA9550-18-1-0371).

References

Anderson, J. R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory*, *7*(5), 326.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. In *IJCAI* (pp. 2–9).

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human–Computer Interaction*, *12*(4), 311–343.

Huffman, S. B., & Laird, J. E. (1995). Flexibly instructable agents. *JAIR*, *3*, 271–324.

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

Laird, J. E., Gluck, K., Anderson, J., Forbus, K. D., Jenkins, O. C., Lebiere, C., ... others (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*(4), 6–21.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, p. 1469).

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514–521).

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding language for interactive task learning. In *Proceedings of the First Workshop on Language Grounding for Robotics* (pp. 1–9).

Mohan, S., Mininger, A. H., Kirk, J. R., & Laird, J. E. (2012). Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, *2*, 113–130.

Myers, C., Ball, J., Cooke, N., Freiman, M., Caisse, M., Rodgers, S., ... McNeese, N. (2018). Autonomous intelligent agents for team training. *IEEE Intelligent Systems*, *34*(2), 3–14.

Salvucci, D. D. (2013). Integration and reuse in cognitive skill acquisition. *Cognitive Science*, *37*(5), 829–860.

Salvucci, D. D. (2016). Cognitive code: An embedded approach to cognitive modeling. In *Proceedings of the 14th ICCM* (pp. 15–20). University Park, PA: The Pennsylvania State University.

Salvucci, D. D., & Taatgen, N. A. (2010). *The multitasking mind*. Oxford University Press.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*(3), 548.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors*, *45*(1), 61–76.