Learning from Instruction: A knowledge-level capability within a unified theory of cognition

Scott B. Huffman and **Craig S. Miller** and **John E. Laird**
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan 48109-2110
huffman@engin.umich.edu

Abstract

How does working within a unified theory of cognition – an architecture – provide useful constraint when modeling large timescale tasks, where performance is primarily determined by knowledge, rather than the architecture's basic mechanisms? We present a methodology for extracting the constraint that comes from the architecture, by deriving a set of architectural entailments which afford certain model properties over others. The methodology allows us to factor the effect that various architectural properties have on a model. We demonstrate the methodology with a case study: a model of learning procedures from natural language instructions, Instructo-Soar, within the Soar architecture.

Introduction

When constructing a model of human behavior, one is confronted with a multitude of design decisions. Over the course of Allen Newell's career, a resounding theme was the useful constraint a unified theory of cognition (UTC) offers in designing cognitive models. A UTC, as embodied in the form of an architecture, posits a set of mechanisms capable of supporting intelligent behavior. By constructing a model within a UTC, we restrict ourselves to designs consistent with the architecture's properties. This constraint is useful because it produces models based on principles that are globally motivated by a broad range of cognitive behavior and phenomena. Since models are based on a consistent set of underlying mechanisms, modeling within a UTC also facilitates the integration of models to cover a larger range of behavior.

The constraint provided by a UTC is most evident for tasks at low timescales. For these tasks, behavior is strongly determined by the architecture itself. This is because there is not enough time for more than a small number of applications of the architecture's primitive mechanisms. However, as the timescale increases, to the level of minutes or hours for task performance, behavior is determined more and more by an intelligent agent's *knowledge*. This is Newell's intendedly rational band [Newell, 1990], in which the agent moves towards approximating a pure knowledge-level system. The large timescale allows the agent to retrieve, compose, and apply relevant knowledge to reach its goals within the current task environment. Because of the distance in timescales between the knowledge level and an architecture's primitive mechanisms, it is less evident how a UTC constrains the design of models for behavior at these higher timescales. The question then is, "What constraints does a UTC offer in designing a model of knowledge-level behavior?" and "How can we reap these constraints?"

We focus our attack on this question by considering *knowledge-level capabilities* (KLC's). These are capabilities that an agent can apply on many different tasks, that occur over large timescales (minutes or longer), and whose performance is primarily determined by the agent's knowledge. Examples include planning, experimentation in the environment, complex analogical reasoning, and learning from instruction. These are not primitive architectural capabilities, because their properties are determined by not only the architecture and the task, but also the agent's knowledge.

In this paper, we present a methodology for developing KLC models within the confines of a UTC. The methodology utilizes *architectural entailments* derived from the architecture's basic mechanisms and properties. The UTC's entailments, together with the constraints of performance required by the task, and human behavior, powerfully guide the development of a KLC model. We illustrate the methodology with a case study, detailing a particular KLC (the learning of procedures from natural language instructions) within a particular UTC (Soar). The case study demonstrates the types of constraints that can be derived from a UTC, and how they lead to KLC model properties.

A Methodology for KLC modeling

The methodology of modeling KLC's within a UTC allows for constraint from three sources: (1) functional and agency constraints on task performance, (2) architectural entailments, and (3) human behavioral data.

We describe each of these in turn.

Constraints on task performance

In addition to simply being able to perform the tasks associated with a KLC, building models within a UTC adds two additional constraints on how performance can be modeled: agenthood and functionality.

Firstly, a KLC must be defined with an eye towards future integration into a complete cognitive agent. This view rules out models that violate the general requirements of agenthood; for instance, that an agent be continually receptive to its environment, even while performing complex cognitive tasks.

Secondly, elements of the model should not exist simply to mimic human behavioral data. Rather, every element should contribute to the agent's performance; that is, the model should be *functional*. Non-functionalities represent technological limitations of the brain. Since such non-functionalities impact a wide range of tasks, they should not appear in models of particular tasks, but should be a part of the *architecture*.

Architectural entailments

Architectural entailments are ramifications of the combined architectural components that define the UTC. For example, an architectural entailment might be whether or not long-term memory is directly examinable. They determine which techniques, structures, etc., are most "natural" within the architecture. Models that are most afforded by the entailments are preferred. As Newell puts it: "There are many ways for Soar to...do any [intendedly rational] task. A natural way is one that involves little additional knowledge and little computational apparatus and indirection...One of the general arguments for the importance of getting the architecture right, even though it appears to recede as the time scale increases, is that doing what is natural within the architecture will tend to be the way humans actually do a task." [Newell, 1990, p. 416-17]

Human behavior data

Ideally, we would like to produce operational models that reproduce human behavior. Unfortunately, human data is strongly colored by subjects' prior knowledge, especially for knowledge-level capabilities. To reproduce the behavior correctly, a simulation must have not only the right model of task performance, but also the right prior knowledge.

One strategy for dealing with prior knowledge effects is to choose versions of tasks that *minimize* prior knowledge. For example, concept learning is often studied with novel, artificial concepts, problem solving with puzzles, and memory with nonsense syllables. However, for broad tasks that require the integration of various types of knowledge, it can be difficult to obtain data minimally affected by prior knowledge.

A second strategy for dealing with prior knowledge is to structure the model so that prior knowledge can be a parameter to the model. A Soar model of cryptarithmetic [Newell, 1990] provides an extreme example of this. In this model, the results of subjects' performance of particular sub-operations, such as choosing the next digit to assign to a letter, are taken directly from protocol data (i.e., parameterized). Thus, the Soar model reproduces the search behavior of the subjects in detail, but does not model how the sub-operators are carried out once selected. Newell points out that "this allows us to test the upper levels without the lower levels, the conceptual advantage being to factor out the influences of the lower levels by design beforehand, rather than by analysis from the total simulation record" [Newell, 1990, p. 376]. Within a UTC-based model of a particular task, parameters should be at the knowledge level, involving the presence (or absence) of knowledge. Other types of parameters, affecting the application of knowledge by the architecture, are at the architectural level, and should affect all tasks.

Soar's architectural entailments

Soar's entailments derive from its basic properties.

Basic properties of Soar

Following Newell [1990, p. 160], we can describe Soar as having six basic properties:

1. **Problem spaces represent all tasks.**
2. **Productions for long-term memory.** All long-term knowledge is stored in associative productions, including search control, operator application knowledge, and semantic and episodic knowledge.
3. **Attribute/value representation.** All declarative structures in working memory are represented as attribute/values.
4. **Preference-based decision procedure.** Productions produce symbolic preferences for working memory elements, including problem spaces, states, operators, and substructures of these objects. The decision procedure interprets the preferences to select appropriate values for working memory.
5. **Impasse-based automatic subgoaling.** Subgoals are created automatically in response to the inability to make progress on a problem.
6. **Chunking based learning.** Chunking produces a new production whenever a result is created during processing in a subgoal. The new production, or chunk, summarizes the result of processing within the subgoal; in future executions of related tasks, chunks will fire, avoiding the subgoal.

Entailments of Soar

Starting from Newell's characterization of Soar [1990, pp. 227-30], we have created a list of Soar's entailments, broken into four categories: behavior, knowledge, long-term memory, and learning. This is not an exhaustive list, but includes entailments that affect our case study of the next section. Numbers in brackets after each entry indicate the properties of Soar (in

the list above) that the entailment derives from. The dependencies are shown graphically in Figure 1.

- **Behavior.**
  1. Goal oriented. Soar creates its own goals whenever it does not know how to proceed [1,5].
  2. Interrupt driven. All productions are being matched continuously, and bring to bear whatever knowledge is appropriate to the current situation. The preference-based decision process allows knowledge to indicate that a more important course of action than the current course should be followed. Impasse-driven subgoals based on the old course of action are terminated when an alternative course of action is selected. [2,4,5].
  3. Serial application of discrete, deliberate operators. A single operator is selected and applied at a time within each goal. [1,4]
  4. Continual shift to recognitional (impasse-free) performance. Whenever an impasse is resolved, chunks are learned that allow the impasse to be avoided in the future. These chunks implement recognitional performance – they simply recognize the situation and act appropriately, without requiring further deliberation (i.e. subgoals). [5,6]
- **Knowledge.**
  5. All knowledge is related to problem-space structures: problem spaces, states, and operators. [1]
  6. Use of an indefinitely large body of knowledge. Soar can use large amounts of knowledge because of the structuring of knowledge as individual productions, and the factoring of that knowledge into multiple problem spaces. [1,2]
  7. Ability to detect a lack of knowledge. Impasses are detected architecturally, and indicate a lack of knowledge to make a decision. [4,5]
- **Long-term memory.**
  8. Associative, recognitional long-term memory. Productions form an associative long-term memory. Knowledge is recalled only when appropriate retrieval cues appear in working memory. [2]
  9. Impenetrable long-term memory. The productions in long-term memory cannot be examined by the agent, for instance by being tested by other productions. Thus, the agent is not fully aware of what is in its long-term memory; access is only gained through working memory retrieval cues. [2]
- **Learning.**
  10. Learning determined by performance. The chunks that are learned are directly dependent on the processing that takes place in a subgoal to produce a result. In particular, the generality (transferability) of chunks is dependent on the generality of the problem solving in the subgoal. [5,6]
  11. Learning based on prior knowledge. Since learning is dependent on performance, and performance is dependent on the agent's existing knowledge, it follows that what is learned depends critically on the agent's preexisting knowledge. [5,6]
  12. Impenetrable learning process. The chunking process is impenetrable to the agent; it cannot learn to alter its architectural learning process.[1] [6]
  13. Monotonic learning. Productions are never removed from long-term memory; once learned, chunks cannot be forgotten. However, their effects can be overridden by other chunks to recover from incorrect learning [Laird, 1988]. [2,4,6]
  14. Routine learning. Chunking is part of the routine performance of the agent, rather than a special process that is deliberately invoked by the agent. Thus, the decision to learn is not under the agent's control (except indirectly as the agent decides when to return results from subgoals). [5,6]

Instructo-Soar: A case study

To demonstrate the methodology of UTC-based KLC model building, we present a case study. We concentrate on the effect of architectural entailments, since that constraint is unique to UTC-based modeling. The KLC is the learning of procedures from interactive, situated natural language instructions. Interactive means that the student requests instruction when it is needed; situated means that the student is within the task domain, attempting to perform tasks, during the instruction episode. The model is embodied in a system, Instructo-Soar, which has been described elsewhere [Huffman and Laird, 1993]. Instructions are given to Instructo-Soar in natural language sentences, requested whenever there is an impasse in task performance. The system can learn completely new procedures, learning both the goal concept (what it means to successfully perform the procedure) and how to perform the procedure. The system also learns to extend known procedures to new situations. This is a KLC because the task takes on the order of minutes, and a subject's knowledge – of natural language, of the task domain, of the mapping between them, and of learning strategies – primarily determines performance.

Although situated, interactive instruction has been shown to be highly effective [Bloom, 1984], there have not been many studies of learning effects during this type of instruction. Most studies of instruction have either focussed on broad instructional strategies, or on the effect of linguistic form of individual instructions (see [Bovair and Kieras, 1990; Singley and Anderson, 1989] for reviews). It has been shown that subjects

[1] However, since learning is dependent on performance and knowledge, chunking can form the basis for many different learning strategies (e.g., induction, analogical learning, abstraction, etc.). These strategies are essentially KLC's that are mediated by knowledge and can be learned.

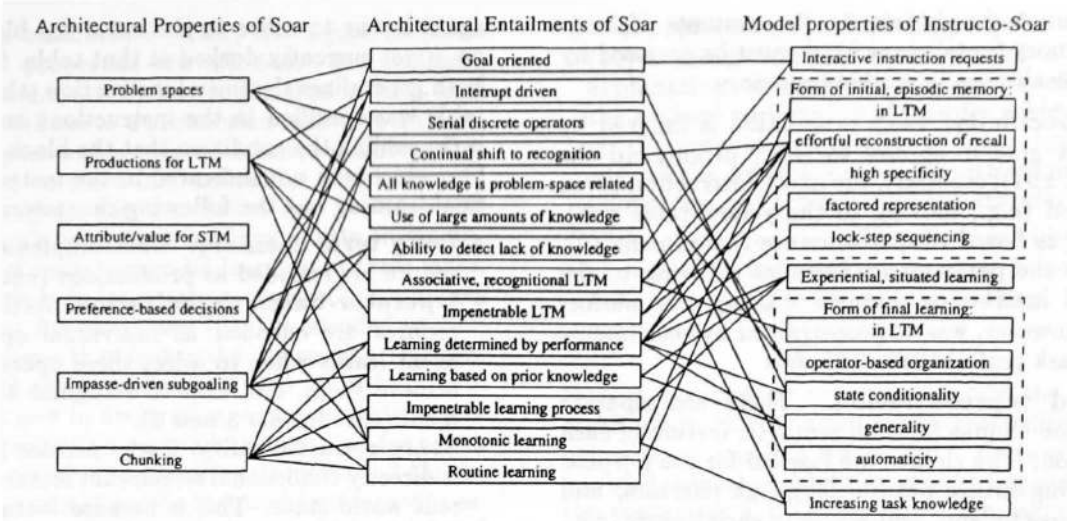

Figure 1: How the entailments of Soar arise and affect a KLC model.

learn better when they have more prior knowledge of the domain [Kieras and Bovair, 1984], which our model predicts. Also, subjects learning from worked-out examples learn more general knowledge when they attempt to explain each step of the example (the self-explanation effect [Chi *et al.*, 1989]). Although not interactive instructions, the worked-out examples provide a similar kind of information to the student.

The Instructo-Soar model exhibits two stages of learning behavior in learning a new procedure. Initial learning is rote and episodic in nature, and exhibits low transfer. Later executions result in high transfer, general learning, based on an explanation process. Whenever the agent is given an instruction (or recalls one it was given in the past), it attempts to explain to itself why that instruction leads to achievement of its current goals. This explanation takes the form of an *internal simulation* of the instructed action – the agent asks "What would happen if I did that?" If this internal simulation results in successful goal achievement (or failure if the instruction is to avoid an action), then Instructo-Soar can learn a general rule capturing the key conditions under which the instructed action applies. A similar "learning by (simulated) doing" process applies when the agent is given an instruction that applies in a hypothetical situation; for instance, a conditional, such as "If the light is on, push the button." In such cases, the agent first creates a hypothetical state (e.g., one with the light on), and then simulates the instructed action, attempting to understand why it leads to achieving the agent's goals.

Next, we examine each of the major properties of the model and indicate how they derive from architectural entailments. The goal is to show how the entailments have impacted the model, thus revealing the effect of developing the model within Soar, our UTC.

Interactive instruction requests.

The agent asks for instruction whenever its available knowledge is insufficient to reach its goals. It can do this because it is goal oriented (entailment 1), and can detect its lack of knowledge (entailment 7).

Form of initial, episodic instruction learning.

As a side effect of natural language comprehension when initially reading the instructions for a new procedure, the agent learns a set of chunks which provides an episodic memory of the instructions. This memory has the following characteristics:

- **Long term memory.** The episode is encoded as a set of productions in Soar's long term memory (entailment 8). Thus, instruction sequences are remembered without an unrealistic load on short term memory, and instructions are not forgotten (entailment 13).
- **Recognitional memory; effortful reconstruction for recall.** The episode is encoded as a set of *recognition rules*, in which the features of the instruction to be recalled make up the *conditions*, rather than the actions, of each chunk. Recognition is the basic form of long-term memory (entailment 8), and processing impasses results in recognitional learning (entailment 4); here as a side-effect of natural language comprehension (entailment 14). The chunks are recognitional rather than recall rules, because the conditions in chunks are dependent on the behavior in the subgoal they are produced from (entailment 10); and in this case, the behavior (language comprehension referent resolution) is dependent on the semantic features of the instruction. To recall parts of the instruction episode, the system must deliberately reconstruct features of the episode, to be recognized by these rules. This is because the

agent cannot simply inspect the contents of long-term memory (entailment 9); it must be accessed by placing recall cues in working memory.

- **High Specificity.** Each instruction is indexed by the exact goal it applies to (e.g., picking up the red block `rb1`); similarly, the exact instruction given is encoded (e.g., moving to the yellow table `yt1`). Learning is based on performance (entailment 10), and here the performance includes referring to the goal and instruction features. Using this performance, however, was not constrained by entailments but by task performance demands.
- **Factored representation.** There are separate recognition chunks for each semantic feature of each instruction. The chunks are learned for the purpose of resolving future natural language referents, and this requires independent access to each feature (e.g., after reading "the red block", you may read "the red one" and need to recover the referent from only its color feature). This behavior requires a set of individual feature chunks because long-term memory cannot be examined (entailment 9); thus, we cannot simply memorize a monolithic semantic structure.
- **Lock-step sequencing.** Instructions are sequenced by being chained one to the next, instead of indexed by the situations they should apply in. Learning is performance based (entailment 10), and the performance of *reading* the instructions does not depend on the instructed action's situations of applicability.

Experiential, situated learning method.

By internally simulating instructed actions, the agent learns general rules that propose the actions directly when they will allow a goal to be reached. This experiential learning method is afforded because learning is based on performance (entailment 10) and is impacted by prior knowledge (entailment 11). Thus the "natural" way for a Soar system to learn about task performance is to perform the task itself (thus making maximal use of the task domain knowledge it already has) – here, internally, in case the task cannot be completed. There are alternatives; for example, reasoning directly in the space of possible condition sets about the conditions under which instructed actions might apply. However, any such method would require adding additional knowledge to Soar (about conditions, condition sets, how to search that space, etc.). In addition, it would be difficult to learn chunks with exactly the desired conditions, since the chunking process is impenetrable (entailment 12) – an agent cannot simply say "now I'll add a chunk to my memory with the following conditions and actions."

Form of final, general learning.

The experiential, situated learning method allows Instructo-Soar to learn the procedure in a general form. For example, when learning to pick up a block, the agent learns to move to the table the block is sitting on, if not currently docked at that table. This learning both generalizes the initial instruction (the color of the table was specified in the instruction) and specializes it (by adding the condition that the block be on the table, which was not indicated in the instruction). This final learning has the following characteristics:

- **Long term memory.** Rules implementing a procedure are encoded as productions (entailment 8).
- **Operator-based organization.** Actions and procedures are encoded as individual operators; the agent learns when to select these operators, how to achieve them, and how to recognize their achievement (entailments 3 and 6).
- **State conditionality.** Each operator proposal rule is directly conditional on relevant features of the current world state. This is because learning is determined by performance (entailment 10); here, an internal performance of the instructed operator from a particular state. The implementation displays reactivity, since if the world state unexpectedly changes, rules proposing the next operator to perform will match or mismatch accordingly (entailment 2).
- **Generality.** The features in each proposal rule are only those *required* for successfully reaching the goal. This is because learning is determined by performance (entailment 10) – here, goal achievement – and prior knowledge (entailment 11) – here, knowledge of how to simulate the basic operators.
- **Automaticity.** The learned rules move the system to recognitional performance (entailment 4). No effort is required to recall instructed operators, as it is for the rote initial learning.

Monotonically increasing task knowledge

There is no structural limit on the amount of task knowledge the agent can acquire from instruction (entailment 6). No forgetting or symbol-level interference will occur; that is, the learning of new productions will not overwrite old ones (entailment 13). Interference may occur at the knowledge level; productions may bring conflicting knowledge to bear on the task. This indicates that there is an actual conflict in the agent's knowledge (e.g., perhaps the instructor has given instructions that conflict).

Analyzing the UTC's effect

Figure 1 summarizes the relationship between the properties of Soar, the entailments they give rise to, and properties of the Instructo-Soar model. In essence, this figure factors the effects of each UTC property and entailment on the model. It gives a detailed answer to the question of how the use of the UTC affects the model. In addition, we can now make arguments about what effect changing individual properties of the architecture may have on the model.

Entailments provide an intermediate level of granularity to compare architectures at. Where another

architecture's entailments differ, the figure indicates which model properties are affected. For instance, ACT* [Anderson, 1983] has a declarative long-term memory in addition to a production memory. This entails a fully penetrable, non-associative memory, that could be used to store instructions as they are read. From our analysis, the associative and impenetrable nature of Soar's long-term memory affects Instructo-Soar's form of initial, episodic learning in three ways: (1) the episode is in long-term memory, (2) it requires reconstruction for recall, and (3) it is encoded in multiple productions that factor the instruction features. In ACT*, (1) the episode could still be in long-term memory, but now in declarative, non-associative memory, (2) no reconstruction would be required for recall, but rather a search of declarative memory, and (3) the representation would not be factored into separate associative pieces; rather, any feature could be retrieved by searching declarative memory.

Model predictions

Thus far, we have described the model's properties in terms dependent on the model's data structures and processes. However, in order to evaluate the model as a testable hypothesis, behavioral predictions must be described independent of the model and architecture. Based on its properties, Instructo-Soar makes a number of testable predictions:

- Initial learning of a new procedure is rote and episodic. Thus: (1) the procedure can only be performed in situations very similar to the original situation it was taught in (low transfer); (2) subjects will have difficulty performing the procedure starting from any state other than the initial state (i.e. they can't start in the middle very well), and (3) performance is slow.
- Learning a general form of a new procedure takes multiple (internal or external) executions of the procedure. Thus: (4) the amount of transfer – the ability to perform the procedure in novel situations – increases with the number of executions, until the maximum transfer level is reached; (5) the execution time decreases with the number of executions. Empirically, we have found that this execution time decrease displays the power law of practice [Huffman and Laird, 1993]. After some number of executions, the procedure is fully generalized. At this point, (6) performance is automatic (recognitional); thus, performance will be fast; (7) subjects will exhibit the *Einstellung* effect; and (8) subjects will be able to execute the procedure equally well starting from any of the intermediate states along the path of the instructed action sequence.
- General learning is the result of a deliberate explanation process, applied to each individual instruction, using prior domain knowledge (here, knowledge of basic operators' effects). Thus, self-explanation [Chi *et al.*, 1989] improves both (9) performance time and (10) transfer – subjects who do not attempt such explanations will achieve only low-transfer; and (11) the amount and quality of transfer achieved depends on the amount and quality of prior domain knowledge of the subject – subjects with minimal domain knowledge will achieve only low-transfer learning.

These predictions seem broadly consistent with what is known about instructional learning, but more experiments are needed to fully evaluate them.

Conclusion

Unified theories of cognition specify the underlying architecture of cognition, which most directly affects behavior at smaller timescales. However, even for knowledge-level capabilities, developing models within the context of a UTC can provide a large amount of useful constraint in model design decisions. The constraint is provided by the behavioral entailments of the architecture, which indicate the natural way to perform the task within the architecture. In this paper we have presented a case study of a methodology for KLC modeling, in which a set of entailments for the Soar architecture were derived from its basic properties. We have shown how these entailments have affected development of a KLC model, Instructo-Soar, that learns procedures from natural language instruction.

References

[Anderson, 1983] John R. Anderson. *The Architecture of Cognition.* Harvard Univ. Press, 1983.

[Bloom, 1984] Benjamin S. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.

[Bovair and Kieras, 1990] S. Bovair and D. Kieras. Toward a model of acquiring procedures from text. In Barr *et al*, editors, *Handbook of Reading Research, Volume II.* Longman, 1990.

[Chi *et al.*, 1989] M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Sci.*, 13:145–182, 1989.

[Huffman and Laird, 1993] S. Huffman and J. Laird. Learning procedures from interactive natural language instructions. In *Machine Learning: Proceedings of the Tenth International Conference.* 1993.

[Kieras and Bovair, 1984] D. Kieras and S. Bovair. The role of a mental model in learning to operate a device. *Cognitive Science*, 8:255–273, 1984.

[Laird, 1988] John E. Laird. Recovery from incorrect knowledge in Soar. In *AAAI-88*, 1988.

[Newell, 1990] Allen Newell. *Unified Theories of Cognition.* Harvard Univ. Press, 1990.

[Singley and Anderson, 1989] M. Singley and J. Anderson. *The transfer of cognitive skill.* Harvard Univ. Press, 1989.