Integration and Reuse in Cognitive Skill Acquisition

Dario D. Salvucci

Department of Computer Science, Drexel University

Received 30 June 2011; received in revised form 1 August 2012; accepted 7 August 2012

Abstract

Previous accounts of cognitive skill acquisition have demonstrated how procedural knowledge can be obtained and transformed over time into skilled task performance. This article focuses on a complementary aspect of skill acquisition, namely the integration and reuse of previously known component skills. The article posits that, in addition to mechanisms that proceduralize knowledge into more efficient forms, skill acquisition requires tight integration of newly acquired knowledge and previously learned knowledge. Skill acquisition also benefits from reuse of existing knowledge across disparate task domains, relying on indexicals to reference and share necessary information across knowledge components. To demonstrate these ideas, the article proposes a computational model of skill acquisition from instructions focused on integration and reuse, and applies this model to account for behavior across seven task domains.

Keywords: Skill acquisition; Instruction following; Learning; Cognitive architectures; ACT-R

1. Introduction

Skill acquisition is the process by which people learn to perform new tasks. The skills acquired to perform a task are almost never wholly new; instead, new task behaviors typically arise from the composition of previously known skills in novel ways and/or novel contexts. For example, adults entering a standard choice-response experiment (e.g., Schumacher et al., 2001) already know a wide array of skills to be applied in this context: They know how to listen to and follow instructions, read stimulus text, and type a response on a keyboard. The novel parts of the task, often provided via instructions, link together this knowledge in the manner defined by the experiment—for instance, the instructions may dictate how to map a stimulus to a response, how to proceed to the next trial, and so on.

Correspondence should be sent to Dario Salvucci, Department of Computer Science, Drexel University, 3141 Chestnut St., Philadelphia, PA 19104. E-mail: salvucci@drexel.edu
There are at least two critical components to skill acquisition when viewed in this way. First, skill acquisition requires integration of one component skill with another. Integration serves in part to define the sequence with which known skills are fused together to realize new task behaviors. Sequencing is not the only role of integration, however. Many component skills require information sharing, in the sense that some skills perceive and acquire task-relevant information, and other skills use this acquired information, all in service of the given task. In the choice-response example, reading the textual stimulus obtains critical information for the task, and this information must be passed to another cognitive process for determination of the proper response. In other words, integration requires that the various component skills share information and cooperate successfully such that information provided by one skill is properly directed to and used by another.

Second, skill acquisition requires reuse of component skills across multiple higher level skills and across different task domains. There are many component skills, like reading text or interacting with a computer, that re-occur across many task domains. Studies have shown that a practiced component skill can transfer to (e.g., Singley & Anderson, 1989) or be adapted for (Charman & Howes, 2003) other domains that use the same component skill. Reuse and integration are closely related, in that reuse of skills necessitates proper integration of the same skill in multiple contexts, ensuring that integration is general enough to handle these contexts.

The themes of integration and reuse have been a focus of large-scale cognitive theories for some time. These themes have been particularly strong in the development of theories as computational models instantiated in a cognitive architecture (e.g., Anderson, 1983, 2007; Newell, 1990). The importance of integration was argued perhaps most famously by Newell (1973), who lamented the lack of integration in psychological theories and urged the community to focus on developing unified theories of cognition (see also Newell, 1990). Since then, many computational models have followed an integrated approach to account for various aspects of cognition, notably in ensuring that ideas for a specific domain fit within a more general cognitive framework. For instance, computational models of game playing (Anderson et al., 2011; Laird, 2002), piloting (Jones et al., 1999), and driving (Salvucci, 2006) have demonstrated the benefits of trying to understand domain-specific behavior through an understanding of core cognitive mechanisms (e.g., memory, perceptual-motor interaction, and so on) and how they combine to interact with complex tasks and environments (see Gray, 2008; Ritter & Young, 2001).

Integration and reuse have also implicitly played central roles in recent theories and models of skill acquisition. One prominent view posits that skill acquisition arises from the interaction between domain-independent methods and domain-specific problem-solving skills (e.g., Anderson, 1987; Logan, 1988; Newell, 1990; Taatgen & Lee, 2003). Under this view, task-specific problem-solving skills arise from applying the general methods under specific conditions, thereby creating more specific (and more efficient) instances of the general methods. With practice over time, the more specific methods become the preferred methods, and this gradual transition from general to specific methods accounts for the performance speedup observed in learning situations. Integration arises naturally in this context from the “compilation” (Taatgen & Lee, 2003) of two or
more task steps into a single efficient task step (that may, e.g., bypass a declarative memory retrieval). Reuse also arises in the re-application of general rules in different contexts, which thereby creates different specific methods depending on the context (e.g., learning specific responses for different stimuli).

This article proposes a computational model of cognitive skill acquisition that provides new insights into how integration and reuse can arise in skill acquisition, specifically skill acquisition that arises from instruction following. The article extends previous models of skill acquisition by focusing on three aspects of behavior: (a) information sharing through the use of indexicals to reference existing knowledge, (b) integration of skills through a process of procedural compilation and utility learning, and (c) reuse of component skills across a diverse set of task domains. The proposed model is developed using a computational cognitive architecture, ACT-R (Anderson, 2007), that defines the abilities and limitations of the cognitive system; the architecture helps to ensure the plausibility of the mechanisms used in the model and also helps to facilitate further integration and reuse with respect to underlying mechanisms such as memory, perceptual-motor systems, and so on.

2. A model of skill acquisition from instruction following

Over the past two decades, there have been several major efforts to model skill acquisition from instruction following using computational cognitive architectures, notably those using ACT-R (Anderson, Taatgen & Byrne, 2005; Anderson et al., 2004; Taatgen, Huss, Dickison & Anderson, 2008; Taatgen & Lee, 2003) and Soar (Howes & Young, 1997; Huffman & Laird, 1995; Lewis, Newell & Polk, 1989). This section describes a new model developed using the ACT-R cognitive architecture (Anderson, 2007). ACT-R is a hybrid production-system architecture that encodes knowledge in two forms. Factual knowledge is stored in declarative memory as a network of inter-associated chunks; for example, a chunk representing the fact “4 + 5 = 9” is connected with the chunks representing the numbers 4, 5, and 9. Each chunk also has an associated continuous-valued activation that determines whether and how quickly the chunk can be recalled from declarative memory; its activation increases with practice or decreases with lack of use. Procedural (or skill-related) knowledge is stored as condition-action production rules, each of which tests its conditions against the current context and, if the conditions are met, fires and executes the specified actions.

This model proposed here uses a slightly modified version of the ACT-R architecture and focuses on issues of integration and reuse to account for skill acquisition from instructions. The model can be broken down into two major aspects. First, the model incorporates a core procedural knowledge base that performs instruction encoding and following, using indexicals (defined soon) as a means of storing acquired information and retrieving stored information. The procedural knowledge also includes production rules for well-known basic tasks (e.g., reading a word or typing a key). Second, the model makes use of ACT-R’s core learning mechanisms to compile the instruction-encoding and instruction-following rules to more efficient, task-specific rules. The learning mechanisms also affect gradual
learning with practice through a process of utility learning. The rest of this section provides an overview of each of these aspects along with a description of relevant architectural processes, including proposed modifications to the architecture. The full model and associated simulation code has been made publicly available for testing and development. 2

2.1. Instruction encoding and following

One major component of the model is dedicated to processing experimenter instructions and encoding them into declarative memory, to be later recalled and executed. The model assumes that instructions are provided verbally by an experimenter—in the model’s case, a simulated experimenter that provides verbal instructions through the simulation. When the model first hears an audio signal, it creates a declarative memory chunk to hold incoming information, listens to each word as it is received, and stores this word in the memory chunk. Because natural language processing is not a focus of this effort (cf. Lewis et al., 1989), the model contains only an elementary parser that processes simple subject-verb-object relationships as well as conditions and simple prepositional phrases.

Table 1 shows a sample set of instructions for the so-called paired-associates task to be detailed later. The model first encodes a phrase that defines the goal to be performed, namely the goal to respond to the task stimuli. It then encodes instruction steps sequentially, linking them to this top-level goal, until it hears a command that ends this sequence, such as another goal specification or an instruction to start executing a task. For example, for the instruction “recall number for word,” the model would encode a declarative chunk as follows:

<table>
<thead>
<tr>
<th>Instruction~3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>isa</td>
<td>instruction</td>
</tr>
<tr>
<td>type</td>
<td>respond</td>
</tr>
<tr>
<td>action</td>
<td>recall</td>
</tr>
<tr>
<td>object</td>
<td>number</td>
</tr>
<tr>
<td>preposition</td>
<td>for</td>
</tr>
<tr>
<td>preobject</td>
<td>word</td>
</tr>
<tr>
<td>previous</td>
<td>Instruction~2</td>
</tr>
</tbody>
</table>

Table 1
Instructions for the paired-associates task

To respond…
- Wait-for visual-change
- Read word *[pointing at the word]*
- Recall number for word
- If success type-object number
- Wait-for visual-change
- Read number *[pointing at the number]*
- Remember-state
- Repeat
The chunk is represented as a set of slot-value pairs and has the overall type *instruction* (as defined in the *isa* slot). The *type* field indicates that this instruction corresponds to the *respond* goal, and the next four slots encode the roles of each word. The final slot links this instruction to the previously encoded instruction, as will be needed for instruction following. (The numbers after the “~” sign are unique identifiers for the chunk names.)

A critical aspect of the model’s encoding is the use of *indexicals* to reference information. Indexicals can be broadly defined as expressions that link particular words to elements in the current context (Nunberg, 1993). It has been noted that indexicals are essential for understanding of instructions (Glenberg & Robertson, 1999), both for common deictic references (e.g., “this” referring to a specific object) and for visual or conceptual references. The model provides a computational interpretation of indexicals, with a symbol serving as an indexical to the needed contextual information. In the above example, both *number* and *word* serve as indexicals and (as described shortly) will be used to retrieve the information necessary to execute the instruction.

The model also handles visual indexicals that reference on-screen visual objects. In a typical experiment, the experimenter providing instructions might point out visual objects to the participant—for instance, saying “read this word” while pointing out the intended word on the screen. To model this type of interaction, the task environment includes a simulated experimenter finger with which, during the instruction phase, the simulated experimenter can point out an object while speaking out the instruction. When this occurs, the model’s production rules associate the encoded instructions with an indexical for the perceived visual location. For example, the Table 1 instruction “read word” would be encoded as follows:

<table>
<thead>
<tr>
<th>Instruction~2</th>
<th>isa</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>respond</td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>object</td>
<td>word</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td>object-location</td>
<td></td>
</tr>
<tr>
<td>previous</td>
<td>Instruction~1</td>
<td></td>
</tr>
</tbody>
</table>

This chunk has largely the same representation as the previous example, except that this instruction contains two indexicals: the visual indexical *object-location* that points to the visual object, and the indexical *word* that dictates how to store the information. Both indexicals are interpreted during instruction following to find, encode, and store the information associated with the *word* specified in the instructions.

Given these encoded instructions, the model follows and executes instructions stored (initially) as declarative knowledge. The rules associated with this process perform two functions: retrieving the next instruction step from memory, then performing the instruction step as specified—often setting a subgoal to perform a known component skill. Thus, the model must also include the already-known skills that people bring to bear when attempting a new task. For present purposes, the model includes several component skills...
needed for the task domains addressed in the next section. Table 2 lists the various component skills (beyond those for instruction processing) currently included in the model.

The component skills make use of the indexicals to share information. This information sharing was realized by means of the use of one canonical ACT-R construct and one modification to the canonical ACT-R architecture. Regarding the latter, the architecture was modified such that models no longer need to specify the type of a chunk in declarative memory—that is, specify the slots of information associated with a particular chunk. Chunk types are a significant problem for a theory of skill acquisition, because there is currently no theory in ACT-R for how such types might be learned. Eliminating the need to specify chunk types (following the example of the Soar cognitive architecture: Laird, Newell & Rosenbloom, 1987) greatly increases the flexibility of how information can be stored and retrieved by production rules.\(^3\)

Given this modification, the model uses ACT-R’s notion of problem state (see Anderson et al., 2004; Borst, Taatgen, & van Rijn, 2010) for storage of contextual information in service of the current task (see also the problem space in Lewis et al., 1989). The component skills that harvest new information (e.g., perception or memory retrieval) store the results of their processing in ACT-R’s problem-state (sometimes called the “imaginai”) buffer. For example, the “read word” step illustrated earlier uses both the word

<table>
<thead>
<tr>
<th>Skill</th>
<th>Description</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-word</td>
<td>Find a word of text and encode it into declarative memory</td>
<td>4</td>
</tr>
<tr>
<td>Search-for</td>
<td>Find a word on the display</td>
<td>5</td>
</tr>
<tr>
<td>Listen-for</td>
<td>Listen for a word and notes that word</td>
<td>3</td>
</tr>
<tr>
<td>Compare</td>
<td>Check if one thing is equal or not equal to another and note the comparison in working memory</td>
<td>5</td>
</tr>
<tr>
<td>If</td>
<td>Check a condition</td>
<td>2+</td>
</tr>
<tr>
<td>Type-key</td>
<td>Type the given key</td>
<td>1</td>
</tr>
<tr>
<td>Type-and-check</td>
<td>Type the given key with visual confirmation</td>
<td>1</td>
</tr>
<tr>
<td>Punch-finger</td>
<td>Punch (press) the given finger to press the key underneath</td>
<td>8</td>
</tr>
<tr>
<td>Press-keypad</td>
<td>Find and press one or more digits on a numeric keypad</td>
<td>8</td>
</tr>
<tr>
<td>Click-mouse</td>
<td>Move the cursor to the given location and click the mouse button</td>
<td>5</td>
</tr>
<tr>
<td>Move-mouse-to</td>
<td>Move the cursor to the given location</td>
<td>4</td>
</tr>
<tr>
<td>Say</td>
<td>Say one or more words</td>
<td>6</td>
</tr>
<tr>
<td>Remember-state</td>
<td>Remember (rehearse) the current working-memory chunk</td>
<td>5</td>
</tr>
<tr>
<td>Recall-state</td>
<td>Recall (retrieve) a previously rehearsed working-memory chunk</td>
<td>4</td>
</tr>
<tr>
<td>Multiply</td>
<td>Multiply one number by another</td>
<td>2</td>
</tr>
<tr>
<td>Divide</td>
<td>Divide one number by another</td>
<td>2</td>
</tr>
<tr>
<td>Drive</td>
<td>Drive a car down a roadway (steering/acceleration only)</td>
<td>8</td>
</tr>
<tr>
<td>Wait-for</td>
<td>Wait for an event, specifically when a visual stimulus appears</td>
<td>2</td>
</tr>
<tr>
<td>Repeat</td>
<td>Repeat the steps for the current goal</td>
<td>2</td>
</tr>
</tbody>
</table>

\textit{Note.} Processing for the “If” conditional is enacted by two dedicated rules plus its embedding in several other rules.
indexical and the \textit{object-location} indexical: It looks at the word at the given location, and then encodes this information as a problem-state chunk as follows:

\begin{center}
\begin{tabular}{lc}
<table>
<thead>
<tr>
<th>Problem-State</th>
<th>\textit{word}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“bank”</td>
</tr>
</tbody>
</table>
\end{tabular}
\end{center}

This problem-state chunk uses the \textit{word} indexical as the slot name for the information and stores the result of visual encoding (i.e., the word “bank”) as the slot value. Note that this process uses the instructions themselves to define the chunk representation, in that the indexical given in the instructions is used as the reference slot in the problem-state chunk. Other component skills access and use this problem-state information. For instance, the step “type-object number” accesses the \textit{number} information in the problem state and presses the key associated with that number. In this way, the model maintains context of its information processing in the problem-state buffer and accesses this context-dependent information through the use of indexicals.

After a problem-state chunk has served its purpose as a storage place for temporary information, it can also be stored and rehearsed in memory, in essence providing an episodic trace of past trials. For example, the “remember-state” step in Table 1 takes the current problem-state chunk—which contains both the \textit{word} and \textit{number} information—and stores it in declarative memory. In a subsequent trial, the “recall number for word” step performs a declarative retrieval for a memorized chunk with a particular word in its \textit{word} slot; if found, its \textit{number} slot information is transferred to the current problem-state chunk. Thus, older problem-state chunks can serve as memorized associations between its indexicals and their associated values, enabling recall and use of a previous context.

2.2. Skill compilation and learning

Instruction encoding and following as described above defines the model’s behavior in the early stages of learning, when instruction knowledge has just been acquired and must be interpreted from the declarative representation. To account for later stages of learning, the model relies on the ACT-R learning mechanism known as production compilation (Taatgen & Lee, 2003). Production compilation combines two consecutive rule firings into a single rule whenever possible, thus producing a more efficient and sometimes more specific method for accomplishing the same step. This process is critical for instruction following for two reasons. First, when two consecutive rules retrieve an instruction step and then act on that step, production compilation can eliminate the retrieval entirely—forming a procedural rule that simply does the step, thus converting the declarative instruction into procedural knowledge. Second, for two consecutive rules that do not involve an instruction retrieval, production compilation may be able to further collapse the rules and achieve the same effect with a single new rule (e.g., if the first rule accesses vision and the second types a key, and the second is not dependent on the first, then the rules may combine into one rule). These principles have been successfully applied to instruction following in domains like dual-choice tasks (Anderson et al., 2005), simulated
air-traffic control (Taatgen & Lee, 2003), flight management (Taatgen et al., 2008), and radar operation (Taatgen, 2005). The model proposed here incorporates the same approach for translating instructions into more efficient procedural skills.

Consider an example derived from the instructions in Table 1 and the chunk representation noted earlier. When the time arrives for the instruction “read word” to take effect, the following production rules (written here in English pseudo-code) initiate the retrieval of the instruction chunk and then performs the instruction:

\[
\text{Do-Instruction}\ast\text{Retrieve-Next-Instruction}
\]

\[
\begin{align*}
\text{If} & \quad \text{the current goal is to perform the next instruction and the retrieval resource is free} \\
\text{Then} & \quad \text{initiate retrieval of the instruction after the current instruction}
\end{align*}
\]

\[
\text{Do-Instruction}\ast\text{With-Action-Location}
\]

\[
\begin{align*}
\text{If} & \quad \text{the current goal is to perform the next instruction and the next instruction has been retrieved} \\
\text{Then} & \quad \text{set the instruction action as the current goal}
\end{align*}
\]

The production compilation process takes these two rule firings and combines them into a single rule:

\[
\text{Do-Instruction}\ast\text{Retrieve-Next-Instruction}++\text{With-Action-Location}~1102
\]

\[
\begin{align*}
\text{If} & \quad \text{the current goal is to perform the next instruction} \\
\text{Then} & \quad \text{set the current goal to read the word at the specified visual location}
\end{align*}
\]

There are two important things to note about this rule. First, the rule no longer retrieves the instruction from declarative memory—it simply does the action and bypasses the retrieval declarative instruction chunk. Second, this rule is specifically instantiated for the “read word” instruction: It has been specialized to perform this action given the previous conditions. In this way, production compilation has made a more specific, more efficient production rule for this action step.

When production compilation creates new rules in this manner, the old and new rules are both present in the knowledge base and may compete with one another when current conditions match multiple rules. ACT-R only allows one production rule to fire at a time, using a (noisy) utility calculation to determine the most advantageous rule to fire in the current context. A newly created rule is given a small utility value and initially loses out when competing with older rules. As the same rule is repeatedly re-created through compilation, its utility gradually increases through a process of reinforcement learning (Anderson, 2007). The end result of this process is a gradual shift from the general instruction-following rules to more efficient, specialized rules for the task being performed. In the canonical version of ACT-R (Anderson, 2007), the increase in a rule’s utility is explicitly specified in each model as a number representing the reward received when the behavior achieves its goal. Unfortunately, explicit specification of a reward is problematic for general instruction-following rules, because the model would be forced to specify a constant reward regardless of task domain. For
this reason, the ACT-R architecture was modified to make successive re-creations of a rule increase its utility such that it approaches the sum of the utilities of the two rules from which it was created. This change thus temporarily bypasses the issue of how and where to specify rewards across domains and for particular domains, leaving this issue for future work.

2.3. Parameter estimation

As a large-scale cognitive architecture, ACT-R incorporates a number of continuous-valued parameters that govern the timing and other aspects of its cognitive, perceptual, and motor processes. The instruction-following model generally utilizes the default values built into the canonical version of ACT-R, which have been derived in aggregate across the many task domains to which the architecture has been applied. However, a few parameter values were modified from the defaults to improve the correspondence between the human data and the model simulations. First, four parameters were preset to values that had been estimated in other work—a combination of parameters from models in the ACT-R tutorials, an earlier dual-choice model (Salvucci & Taatgen, 2008), and recent work regarding motor-feature preparation time (Kieras, 2009). Then, seven additional parameters were re-estimated to produce a good qualitative and quantitative fit to the human data across the task simulations detailed in the next section. These parameters govern the number of rehearsals for a memorized chunk, memory decay and retrieval latency, the speed of utility learning, and the latency of visual ending and eye movements. The specific parameters (and ACT-R abbreviations) and their values are as follows:

- Preset according to the ACT-R paired-associates tutorial model: retrieval threshold (:rt) = −1.7, activation noise (:ans) = 0.5, expected gain noise (:egs) = 0.1
- Preset according to dual-choice model in Salvucci and Taatgen (2008): tone encoding delay (:tone-recode-delay) = 0.14
- Preset according to Kieras (2009): preparation time for a motor movement (:motor-feature-prep-time) = 0
- Preset to ensure successful recall of instructions: number of rehearsals for an encoded instruction = 10
- Estimated: number of rehearsals of memorized information = 0 (i.e., chunk is merely stored in memory but not rehearsed via retrieval)
- Estimated: activation decay of chunks in declarative memory (:bll) = 0.5
- Estimated: multiplicative latency factor for retrieval times (:lf) = 0.3
- Estimated: scaling constant for reinforcement of re-created production rules (:alpha) = 0.02
- Estimated: scaling constant for visual encoding times (:emma-enc-fac) = 0.008
- Estimated: exponent scaling factor for visual encoding times (:emma-enc-exp) = 0.3
- Estimated for the driving model: steering scale factor = 0.5
2.4. Relation to previous models

As mentioned, several past efforts have developed models of skill acquisition from instruction following, and also addressed issues of integration and reuse either explicitly or implicitly. The proposed model extends these accounts in several ways to further illustrate how the cognitive system can acquire new skills. First, the model’s process of encoding instructions involves (visual or aural) perception of instructions and sufficient memorization of instructions such that they can be retrieved for later execution. Instructions can also specify visual locations that must be remembered to access certain information. Most earlier models bypassed the encoding process and placed already-encoded instructions in declarative memory (e.g., Anderson et al., 2004; Taatgen, 2005; Taatgen & Lee, 2003). One early model (Lewis et al., 1989) aimed to read and interpret natural-language instructions (although it did not account for the associated visual processes). Two other models (Fu et al., 2004; Huffman & Laird, 1995) did aim to account for real-time instruction encoding, as does the current model. The current model also incorporates the ability to locate and remember visual locations as part of the instructions, extending earlier work in which visual locations were preset in the production rules (e.g., Anderson et al., 2004).

Second, the proposed model heavily emphasizes the use of indexicals as a way to reference information stored in the current task context. Earlier models have used a variety of methods for sharing information across component skills. Some models have performed simple tasks that did not require problem-state information (e.g., Taatgen, 2005). Other models (e.g., Anderson et al., 2004) employed chunk types that subclassed other types in a manner akin to object-oriented computer programming. A few models used a form of implicit indexicals with variable slot names but augmented the ACT-R architecture with special buffers to enable interaction with the computer tutor (Anderson, 2005, 2007; Chapter 5). The current model, following the spirit of the most recent ACT-R work (e.g., Anderson et al., 2004; Borst et al., 2010), relies solely on the canonical problem-state buffer to maintain context and on the canonical perceptual and motor buffers for task interaction.

Third, past efforts have incorporated varying degrees of reuse in their accounts of behavior. Some efforts (e.g., Anderson et al., 2004; Taatgen & Lee, 2003) have used an instruction-following model to learn a single (albeit very complex) task—which elucidates the process of instruction following but has fewer implications for reuse in particular. A few efforts have involved reuse for more than one task: Taatgen (2005) reused instruction-following rules to account for choice tasks in three conditions (varying parameter values but not rules among conditions), and Anderson (2007) reused similar rules in accounting for mathematical problem solving of linear equations and “pyramid” problems. The current effort accounts for a larger array of tasks (seven), and as seen in the next section, provides a quantitative analysis of rule reuse across these tasks to better illustrate patterns of reuse. In addition, the current model maintains the same declarative and procedural knowledge (and parameter values) across task domains, further demonstrating how a single model can generalize to many task domains.
The current model also has important limitations, including the somewhat rigid sequencing of instruction steps and the limited grammar for understanding instructions. These and other limitations will be discussed in the concluding section. First, however, it is important to demonstrate how the model can account for behavior across a variety of task domains, thereby providing evidence of its ability to reuse and integrate component tasks through skill acquisition.

3. Task simulations

The proposed model of skill acquisition from instruction following has thus far been applied to seven tasks, chosen to be both diverse and complementary. The tasks are diverse in that their associated experiments were originally intended to measure disparate aspects of the human system, ranging from memory to visual processing to multitask performance. The tasks are also complementary in their use of instructions: Some rely almost completely on instructions to define very novel tasks (e.g., dual-choice tasks), whereas others rely more heavily on already-known skills to perform familiar tasks (e.g., driving). The next seven subsections detail applications of the model to each illustrative task, and the final subsection summarizes the findings across all tasks.

3.1. Paired associates

The paired-associates task has been used to study the relationship between response latency and accuracy. In one experiment (Anderson, 1981), participants read a noun and then an associated digit 0–9 and had to memorize and then recall the pairings, for example, king →7, pipe →1. The participant first saw the noun, recalled and typed the digit if possible, then (regardless of correctness) again viewed the associated digit. The experiment measured the response time and accuracy as a function of the number of exposures to each stimulus-response pairing. The empirical results of Anderson (1981) are shown on the left-hand side of Fig. 1. Block 1 comprised the initial presentations of the pairings to participants, and thus correctness of responses started at zero. Over subsequent blocks, participants increasingly recognized the pairings: Correctness steadily increased, and response time (for correct responses) steadily decreased.

Two previous models of this task\(^5\) account for behavior in very different ways. The first model is typical of many modeling efforts, with hand-coded rules that demonstrate how the architecture can account for experimental results. The second model is more closely related to the present here, since it interprets declarative instructions and eventually compiles them into procedural rules that perform the task; however, this model assumes that declarative instructions are already strongly activated in memory, with no account for the instruction-encoding process.

The current model’s account of the paired-associates task begins with the instructions in Table 1 discussed earlier. The instructions walk the model through the steps of the task, in a simplified pseudo-English grammar analogous to what might be said to a
human participant. The two steps that involve reading an item also use the simulated experimenter finger to point out the on-screen items. In addition, the model makes use of indexicals as previously noted, for keeping track of both word and visual information. To initiate execution of the task, the model is instructed to “start respond” after hearing these instructions. The key to the learning that occurs during the task is the role of production compilation. The model requires 26 rule firings to complete the first trial. As production compilation learning performs its work on the declarative instructions, it eliminates the need for retrievals and creates specialized rules that encode the associated pairs. By the end of a task simulation, the model requires only 18 rule firings to complete a trial; almost all the remaining rules initiate or wait for perceptual-motor processing (like reading a word or typing a key) and cannot be further optimized.

Fig. 1 shows the model’s results for 10 task simulations with respect to response time ($R = .97$, $RMSE = 0.24$) and correctness ($R = .96$, $RMSE = 0.08$). Early in the simulations, the rehearsed associations are not strongly active in declarative memory, and thus ACT-R takes more time to retrieve this information and is more likely to experience a retrieval failure. Each time a retrieval is successful, however, production compilation creates or strengthens a rule that bypasses the retrieval and simply produces the correct response for a given stimulus—leading to faster and more accurate performance. The psychological account of behavior here is essentially the same as the (second) earlier model for this task. Unlike this earlier model, however, the new model encodes the instructions
through its aural processor in a realistic way, and rehearses the instructions to ensure that
they can be recalled when needed (rather than simply presetting a high activation). The
model also receives the visual location of the items via the simulated experimenter; this
process provides little benefit for the single-item stimuli in this task, but it will provide
greater benefits for other tasks with more complex visual displays.

It should be noted here that the previous models of the paired-associates task actually
fit the data more accurately than the current model. However, the previous models also fit
only this one task domain; these models are unable to perform any other tasks and thus
provide no accounts of behavior in other domains. In contrast, the current model explic-
itly addresses the challenge of stretching across data sets, and in doing so, favors gener-
ality across diverse domains over ideal fits for any one particular domain.

3.2. Equation solving

The equation-solving task (Salvucci & Anderson, 2001) asked participants to solve
equations of a specific form, namely of the form $a \times B = A / b$, which can be solved by
computing $x$ as $(A/a)(B/b)$. Participants were instructed to encode the numbers in a partic-
ular order in four sessions: left to right ($a$, $B$, $A$, $b$), left to right in pairs ($a$, $A$, $B$, $b$),
right to left ($b$, $A$, $B$, $a$), and right to left in pairs ($b$, $B$, $A$, $a$). The experiment examined
the relationship between eye movements and cognitive processing, looking at the number
of gazes and average gaze durations as indicators of this relationship.

Fig. 2 shows the two main results from the experiment. The graph in the upper left
shows the human data for the proportion of trials in which participants fixated 4, 3, or 2
numbers; given that there were four numbers, fewer than four fixations indicated that par-
ticipants encoded one or more numbers peripherally (note that only correct trials were
included, and thus all numbers must have been encoded to compute the answer). The
human data show that participants fixated all four numbers in about two thirds of all
trials but also encoded one (and very rarely two) number in the periphery in about one
third of all trials.

The lower graph displays gaze duration—time spent looking at a number—as a func-
tion of the computation that could occur at the various steps in the instructed sequences:
0C represented gazes that included no computation (e.g., the first number); 1C repres-
ented gazes that included one computation (e.g., the gaze on $A$ after encoding $a$, when
$[A/a]$ could be computed); and 2C represented gazes that included two computations (i.e.,
the final gaze, when $[A/a]$ or $[B/b]$, and then the answer $x$, could be computed). Partici-
pants exhibited a clear effect of cognitive processing in that each computation required
an additional (approximately) 300 ms, an additional processing time that clearly impacts
the gaze durations for the respective numbers.

As for the paired-associates task, previous research has reported on a computational
model of this task (Salvucci, 2001a,b), but the skill knowledge in this model was hand-
coded to represent behavior in the task. The current model’s instructions for the sequence
$(a, B, A, b)$ are shown in Table 3. Just like the instructions in the real experiment, these
instructions dictate the sequence of encoded numbers (and pointing at their locations to
provide visual indexicals) in addition to a specification of where intermediate results should be computed. The division and multiplication instructions are translated into retrievals of declarative facts, which are assumed to be highly active and easily retrievable in memory. Entry of the final answer assumes that the model types the answer while encoding the text field to visually confirm the keystrokes.

Fig. 2. Equation-solving task, human (Salvucci, 2001a) and model results for proportion of trials with 4, 3, or 2 gazes and gaze duration as a function of the number of necessary mental computations.

Table 3
Instructions for the equation-solving task

To respond…
Read $a$ [pointing at the number]
Read $B$ [pointing at the number]
Read $A$ [pointing at the number]
Divide $A$ by $a$ to-get $A/a$
Read $b$ [pointing at the number]
Divide $B$ by $b$ to-get $B/b$
Multiply $A/a$ by $B/b$ to-get $x$
Type-and-check $x$ [pointing at text field]
Repeat

*Note. This is only one of the four instructed encoding sequences; the others follow the same pattern.*
Again like the paired-associates task, the model’s instruction-following rules combined with ACT-R’s production-assembly mechanism gradually translate these instructions into procedural rules that perform the task. The model is given 10 practice trials to mimic the practice period in the experiment, and then behavioral data are collected from 40 subsequent test trials. The model’s behavior in the test trials for the most part resembles that of the experiment participants, as seen in Fig. 2. The model does not encode numbers peripherally as often as did the participants, but it does indeed demonstrate some peripheral encoding, fitting the overall trend, albeit for three data points \( R = .93, \text{RMSE} = 0.20 \). This effect arises from ACT-R’s built-in model of eye movements and visual attention (see Salvucci, 2001a,b): Visual encoding is a function of the object being encoded and its distance from the current gaze location, and when encoding occurs quickly and attention is soon directed to another location, the first location is sometimes skipped. More significantly, the model produces a good fit to the cognitive processing times evident in the gaze durations involving zero, one, or two computations \( R = .99, \text{RMSE} = 0.14 \). In the model, the extra time for each computation arises from the memory retrieval and additional rule firings needed to recall the division and multiplication facts. The required time for each computation is slightly less than in the human data, but the model captures the linear trend with respect to the number of computations and provides a good approximation of the baseline time in the case of no computations.

### 3.3. Menu selection

The menu-selection task (Byrne, Anderson, Douglass, & Matessa, 1999) involved the selection of a target item in a vertical menu of options, as is typical in graphical human—computer interfaces. Participants in the study viewed a menu title specifying a target item (a digit or letter), and when they clicked on this area, a menu appeared below the title with a vertical list of items. In the conditions analyzed by Byrne et al. (1999), the menus had 6, 9, or 12 items, and the target item appeared randomly at any location.

Their analysis examined two measures of task performance: the response time needed to find and click on the target, and (using eye-tracking data) the distribution of initial fixation locations across the menu items. The graph at the upper left of Fig. 3 shows response time as a function of menu length and target item (i.e., the location of the target). Response times were roughly constant when the target appeared in the first four positions; after this point, there is a clear trend indicating that response time grew linearly as the target was placed farther down the menu. The graph at the lower left shows the proposal of initial fixations as a function of menu length and target item. The graph indicates a top-down trend in that the most initial fixations occur on the first item, but a sizable proportion of initial fixations (over half) occur on subsequent items 2 to 5, with the proportions decreasing to effectively zero by item 6. There was no significant effect of menu length for either of the two measures.

The instructions given to the model for the menu-selection task appear in Table 4. The model is directed to read the target (in the menu title), click on it to display the menu, search down for the target, and click the mouse at the target’s location. The second
click-mouse instruction contains the special indexical “there,” which accesses the previously noted location (i.e., the location found by the previous search instruction). The final “read” instruction ensures that the eyes return to the menu title in anticipation of the next trial (the menu disappears after the second click, and thus a person’s eyes would normally return to the menu title as the only visible object on the display).
Fig. 3 shows the results of the model simulations as compared to the human data. With respect to response time, the model produces the main effect of a linearly increasing response time across target items ($R = .90$, $RMSE = 1.08$), although the model does not predict the flatter region for items 1–4 in the human data. The increasing response time is a function of the model’s visual search down for the target, which alternates execution of two rules: one that finds the next item and another that encodes the information. (Eventually the target item is matched by a third rule.) The model unfortunately overpredicts the response times by roughly 50%; people’s search time in this task was noticeably faster than that of the model. (See Anderson, Matessa & Lebiere, 1997; Byrne, 2001; and Salvucci, 2001a,b, for alternative models.)

With respect to initial fixations, the model again reproduced the main effect, namely fewer initial fixations for lower target items ($R = .96$, $RMSE = 0.04$). This result derives from ACT-R eye-movement mechanisms described earlier for equation solving. As each item is found and encoded, there is some probability with which the model moves its visual attention to the next item before the current item is fixated, producing a skipped fixation. The model’s result curve is not as smooth as that for the human subjects, in that it is more likely to skip item 2 relative to item 3. Nevertheless, these results show that the same eye-movement mechanisms that apply for equation solving apply for menu selection as well, and provide a straightforward account of the human data in both tasks.

### 3.4. Dual choice and perfect time-sharing

The dual-choice task has been used for many years to investigate the limitations of concurrent processing (see Pashler, 1994; for a review). In a recent study of dual-choice performance, Schumacher et al. (2001) examined the behavior of participants on two particular choice tasks: an aural-vocal task with an audio stimulus and a vocal response (e.g., a low tone corresponding to the spoken response “one”), and a visual-manual task with a visual stimulus and a manual response (e.g., the stimulus “O —” corresponding to a keystroke using the right index finger). Schumacher et al.’s Experiment 1 included five sessions. An initial session asked participants to practice the tasks in isolation. The four subsequent test sessions tested participants in mixed single-task and dual-task trials, where the dual-task trials involved simultaneous presentation of the two stimuli. Of note, this experiment included instructions to perform both tasks as quickly as possible with no constraints on the serial order of responses. (This last point stands in contrast to the standard procedure in the psychological refractory period paradigm, discussed in the next section.)

The results of Schumacher et al.’s Experiment 1 in the four test sessions are shown in Fig. 4, specifically the average response time for each task in the single-task and dual-task trials. Beyond the general speedup in performance over time for both tasks, the effect of the dual-task condition decreases over the course of the sessions. By the final session, participants have achieved what Schumacher et al. called “virtually perfect time-sharing,” with effectively the same performance in the single- and dual-task conditions.
Recent ACT-R models of this task (Anderson et al., 2005; Salvucci & Taatgen, 2008; Taatgen, 2005) have represented instructions as already-known declarative facts, improving performance over time due to production compilation. The current model’s account of this task is the same in its core learning mechanism—production compilation—but differs in that it encodes instructions aurally, parses them, and encodes them incrementally as declarative facts. These instructions are shown in Table 5. For each task, the first step is encoding the stimulus in either its aural or visual form. Then, the task instructions dictate how to respond for the various stimuli. (This sequential form of testing stimuli also differs from past ACT-R models of this task, to be addressed in the concluding general discussion.) The behaviors for the component tasks are interleaved using threaded cognition (Salvucci & Taatgen, 2008, 2011), the ACT-R mechanism that enables concurrent multitasking. Threaded cognition alternates rule firings between the component tasks, except when there is a resource blockage (e.g., one task trying to access declarative memory, while the other is retrieving declarative information).

The model results are included for comparison to the human data in Fig. 4 ($R = .89$, $RMSE = 0.07$). The learning effect arises from compilation of the instructions with the instruction-following rules, gradually producing more efficient rules to perform the task. Early on, both tasks require declarative retrieval of instructions, and because only one task at a time can retrieve instructions from declarative memory, one task is delayed in the dual-task condition, producing the dual-task effect. Later on, as compilation eliminates these retrievals, the declarative bottleneck disappears along with the dual-task effect. It is important to note that ACT-R does indeed have a cognitive bottleneck, specifically in that procedural resource can only handle one production firing at a time (unlike, e.g., the EPIC architecture: Meyer & Kieras, 1997). However, under the right circumstances, the rules can interleave in such a way that even procedural interference is eliminated. (Interested readers can consult Salvucci & Taatgen, 2011, for detailed diagrams of this interleaving; although the process of instruction listening and following differs between those models and the model described here, the resulting expert models of performance are essentially the same.)
An especially interesting aspect of the modeling account here is the extent to which production compilation can compress and optimize the task instructions. Initially, the aural-vocal task requires up to 16 rule firings from the start of the trial to initiation of the spoken response, and the visual-manual task requires up to 15 rule firings to the initiation of the keystroke response. By the end of the sessions, however, the aural-vocal task requires only two rule firings to encode the aural stimulus and initiate speech; the second production rule is a compilation of up to (depending on the stimulus) 15 original rules (14 of which preceded speech in the early stages). Similarly, the visual-manual task requires only three rule firings to find and encode the stimulus and to initiate the keystroke; the final rule can be a compilation of up to 13 original rules. Thus, production compilation serves two critical purposes in both eliminating declarative retrievals (the primary reason for the decreasing dual-task effect) and collapsing the sequence of rule firings (the primary reason for general speedup).

### 3.5. Dual choice and the PRP effect

Schumacher et al. (2001) followed their demonstration of virtually perfect time-sharing (PTS) above with a demonstration of the so-called psychological refractory period (PRP) effect. A number of earlier studies had used dual-choice tasks to argue for the existence of a cognitive bottleneck in human performance (see Pashler, 1994). Fig. 5 shows a schematic depiction of a typical result in these studies. The two choice tasks are separated in time by a short delay known as the stimulus onset asynchrony (SOA). When examining the response time for each task as a function of SOA, a picture of dual-task interference
emerges: While the first task (Task 1) experiences no effect of SOA, the second task (Task 2) shows a clear effect for short SOAs that decreases, and gradually disappears, for longer SOAs. The effect on the second task has been argued to show the presence of a cognitive bottleneck, because this effect appears even when the two tasks do not overlap with respect to their perceptual or response modalities (like the aural-vocal and visual-manual tasks in the previous section).

Schumacher et al. (2001) argued that the PRP effect arises primarily due to task instructions: Under the PRP paradigm, people are instructed to respond to the first task before the second task, and Schumacher et al. argued that this constraint makes the second task “wait for” the first and thus produces the PRP effect. To demonstrate this idea, they reported a second experiment in which participants performed the dual-choice task in the same way as in the PTS experiment, except with the added response constraint (first task before the second). This simple change of task instructions, even using the same participants as the PTS experiment, resulted in a PRP effect in response times, shown in Fig. 6.

Fig. 5. Schematic of a typical result using the perceptual refractory period (PRP) task paradigm. (Reproduced with permission from Salvucci & Taatgen, 2011.)
To account for the response constraint, the model’s instructions—see Table 6—were altered very slightly from those in the PTS condition. The individual task instructions remained the same, but to perform the dual-task conditions, the instructions state simply to perform the first task (aural-vocal) and then the second task (visual-manual), rather than both concurrently. These instructions give rise to the model results included in Fig. 6. The model’s behavior for each individual task, at all stages of learning, is identical to that in the PTS condition; the critical difference is that here, in the PRP condition, the visual-manual task only begins when the spoken response is initiated by the aural-vocal task model. The PRP curve is slightly longer for the model than the human data but exhibits the same negative slope and same overall pattern as the human data ($R = .97$, $RMSE = 0.04$).

### 3.6. Tracking and choice

The tracking and choice task integrates a choice task like that above with a continuous manual tracking task. In an experiment by Martin-Emerson and Wickens (1992), the tracking component required participants to keep a cursor over a target while the cursor was continually displaced by pseudo-random noise in the horizontal and vertical directions. The experiment included both easy and hard conditions that varied the magnitude of the displacements. Occasionally, a choice task would be introduced in which participants viewed an arrow stimulus and responded with a keystroke denoting the direction of the arrow (left or right). The arrow was displayed at varying offsets (eccentricities) from the center of the target display, and task performance was measured in terms of tracking error and identification response time as a function of stimulus offset.

Fig. 7 shows the human data collected by Martin-Emerson and Wickens (1992). In the upper-left graph, tracking error increased as the arrow stimulus was presented at larger offset distances from the target area. There was also an obvious effect of tracking-task difficulty, with the harder condition evoking larger tracking errors. In the lower-left graph, response time for the arrow stimulus (i.e., the choice task) also increased with
increasing stimulus offset. For this measure, however, there was no effect of tracking-task difficulty.

The model’s instructions for this task appear in Table 7. The tracking instructions are very simple, namely to move the mouse to the target (fighting against the cursor’s displacements). The choice task instructions closely resemble those for the visual-manual task in the dual-choice domain, except that the choice here is a binary choice between only two options. Each task is practiced separately, as in the original experiment. The final instruction directs the model to perform the tracking and choice tasks concurrently.

Over the course of practice trials, the model compiles these instructions into very efficient models in the same way it does for the dual-choice domain. The tracking task eventually requires only three rule firings to find the target, move the mouse, and repeat; the final rule (that repeats the behavior) is a compilation of five original rules that interpret the repeat instruction and set the next iteration in motion. Likewise, the choice task eventually requires four rule firings to find the arrow stimulus, encode it, record the encoded content, and repeat; the final rule in this case is a compilation of eight original rules. The two task processes execute concurrently using threaded cognition (Salvucci & Taatgen, 2008). When an arrow appears, the choice-task model encodes the direction of the arrow, and then begins to move visual attention back to the target as it makes its keystroke response. The ACT-R eye-movement mechanisms (Salvucci, 2001a,b), critical for the equation-solving and menu-selection tasks, are again critical here: the farther the eyes

| Table 6
Instructions for the dual-choice task in the PRP condition |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>To choose-visual-manual…</td>
</tr>
<tr>
<td>Read stimulus [pointing at the stimulus]</td>
</tr>
<tr>
<td>Compare stimulus literal “O – –”</td>
</tr>
<tr>
<td>If success punch right-index</td>
</tr>
<tr>
<td>Compare stimulus literal “– O –”</td>
</tr>
<tr>
<td>If success punch right-middle</td>
</tr>
<tr>
<td>Compare stimulus literal “– – O”</td>
</tr>
<tr>
<td>If success punch right-ring</td>
</tr>
<tr>
<td>Done</td>
</tr>
<tr>
<td>To choose-aural-vocal…</td>
</tr>
<tr>
<td>Listen-for tone</td>
</tr>
<tr>
<td>Compare tone literal low</td>
</tr>
<tr>
<td>If success say “low”</td>
</tr>
<tr>
<td>Compare tone literal middle</td>
</tr>
<tr>
<td>If success say “middle”</td>
</tr>
<tr>
<td>Compare tone literal high</td>
</tr>
<tr>
<td>If success say “high”</td>
</tr>
<tr>
<td>Done</td>
</tr>
<tr>
<td>To do-dual-task…</td>
</tr>
<tr>
<td>Do choose-aural-vocal</td>
</tr>
<tr>
<td>Do choose-visual-manual</td>
</tr>
<tr>
<td>Repeat</td>
</tr>
</tbody>
</table>

| 850  
Fig. 7. Tracking and choice task, human (Martin-Emerson & Wickens, 1992) and model results for tracking error and response time as a function of the offset of the visual stimulus from the tracking target.

Table 7
Instructions for tracking and choice task

<table>
<thead>
<tr>
<th>To track-target…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move-mouse-to target [pointing at the target]</td>
</tr>
<tr>
<td>Repeat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To respond-to-arrow…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read arrow [pointing at the arrow]</td>
</tr>
<tr>
<td>Compare arrow literal “&lt;”</td>
</tr>
<tr>
<td>If success punch left-pinkie</td>
</tr>
<tr>
<td>If failure punch left-middle</td>
</tr>
<tr>
<td>Repeat</td>
</tr>
</tbody>
</table>
must move to encode the arrow and then return to the tracking target, the longer the response time, and the greater the tracking error (due to increased time away from this task). The model’s results for tracking error ($R = .97$, $RMSE = 2.29$) and response time ($R = .69$, $RMSE = 0.10$), included in Fig. 7, exhibit the main effect of stimulus offset on both measures (albeit with a smaller slope), the effect of tracking-task difficulty on tracking error, and the lack of an effect of tracking-task difficulty on response time.

### 3.7. Driving and phone dialing

The driving and phone-dialing task derives from a common activity on today’s roadways, namely the use of mobile phones while driving. In one study (Salvucci, 2001b), drivers dialed a mounted phone using one of two modalities: manual dialing using standard phone buttons, and voice dialing using spoken commands. For each modality, drivers dialed full seven-digit phone numbers as well as “speed” codes associated with a full number (e.g., pressing “2” or saying “home” to call home). The dialing modality crossed with the length of dialing thus yielded four conditions: full-manual, speed-manual, full-voice, and speed-voice.

Fig. 8 shows the core results of the study. The upper-left graph shows the total dialing time for each of the four conditions while driving and as a single task (i.e., not while driving). The inclusion of driving led to slightly longer dialing times overall, but in fact drivers were able to interleave driving and dialing efficiently such that the additional dialing time in the driving condition was relatively small (roughly 16% on average). The lower-left graph shows lateral velocity—the average absolute side-to-side velocity of the vehicle—as a measure of vehicle stability while driving. Full-manual dialing led to the largest lateral velocities above the baseline (no secondary task) condition, and speed-manual dialing also resulted in larger velocities than baseline. The velocities for voice dialing, however, did not differ significantly from baseline; this result was especially noteworthy given that full-voice dialing yielded the largest total dialing times, and thus driver performance was affected most by the modality of dialing rather than the total dialing time.

The driving and dialing task provides an interesting test for cognitive models in two ways. First, the task is very much a real-world task and thus offers some generalization beyond the psychology laboratory. Second, accounting for this task requires integration of a known well-practiced skill—driving—with a newly instructed dialing task (Although the participants had presumably dialed a phone before, both the phone itself and the methods of dialing the phone were, in large part, new to participants.). The driving component skill was taken from an existing ACT-R model of driving (Salvucci, 2006) and added to the larger model’s set of component skills. Driving was initiated via the simple instruction “start drive.” The model received instructions for the various dialing task conditions, as illustrated in Table 8. The final “dial” goal performs the same task as the experiment participants: listening for the actual dialing instruction while driving, and then initiating the dialing task condition associated with that instruction.

When the task is run in simulation, threaded cognition allows the two tasks to interleave execution. The simulation results, which are shown in Fig. 8, exhibit a good
qualitative and quantitative fit for dialing time ($R = .98$, $RMSE = 1.14$) and lateral velocity ($R = .97$, $RMSE = 0.04$). Like human drivers, the model can interleave the tasks efficiently enough that dialing times increase only slightly while driving. For manual dialing, the need for visual guidance of button presses pulls the model’s visual processing away
from the roadway for short periods of time, negatively affecting its steering performance and producing larger lateral velocities. For voice dialing, the rule firings that initiate the listening and speaking processes are interleaved with driving, but this interference is small enough that lateral velocities remains at baseline level. (The model produces a similarly good fit, $R = .99$, $RMSE = 0.29$, for another measure of steering performance, lateral deviation of the vehicle from lane center.)

3.8. Summary

The model provides a good account of behavior in all seven tasks. There remains room for improvement for particular tasks, most notably for equation solving and menu selection, for which the model captures the qualitative pattern of results but does less well in capturing the quantitative values. Nonetheless, the fact that a single model of skill acquisition can account for behavior in such a variety of tasks—with only the task instructions varying among tasks—demonstrates the promise of the proposed model in providing a unified approach to skill acquisition.

The task simulations illustrate how integration and reuse play an important role in skill acquisition. Integration arises in the task simulations in several forms. The sequencing of task steps arises in a straightforward way by sequentially following the instructions, one after another. More interestingly, production compilation helps to fuse together the task steps by eliminating the general instruction-following rules between component skills: Early on, the model requires the general rules to transition from one component skill to the next, whereas later on, the compiled rules from one skill incorporates a direct link to the next skill and immediately instantiates that skill. Moreover, the successful composition of skills includes the passing of shared information from one skill to another through the use of indexicals, which dictate how information is stored in the problem-state buffer by perceptual skills and later retrieved from the buffer by other component skills.

Skill reuse is also successfully achieved in the task simulations. It is not obvious from the results thus far that this need be true—for instance, the single model could in principle have included only disjoint rules for all tasks such that all rules fire only for a single task. To address this point, consider the reuse of the model’s rules as graphed in Fig. 9. The figure shows, for all production rules in the model, how many rules are used (fire at least once) for just one task, two different tasks, and so on. (The model includes additional rules that do not fire for any of the tasks but were originally included for completeness.) A total of 45 rules (41%) fire for more than one task, and 14 rules (13%) fire for all seven tasks. The rules that fire for the majority of tasks are those associated with the most commonly needed skills: instruction listening and following, rehearsing information, and reading visual information. Some skills, like comparing objects or listening for an aural stimulus, are not ubiquitous but are at least active for two or three tasks. Most important, in contrast to the vast majority of cognitive models targeting a single-task domain, the current model reuses large portions of its skill knowledge and transfers the knowledge among multiple tasks.
4. General discussion

In the learning of new tasks, skill acquisition can in many ways be better characterized as skill composition—the composition of already-known component skills in novel ways to enable the performance of new skills and tasks. As such, an important key to understanding skill acquisition comes in understanding the processes of integration and reuse of component skills. This article proposes a cognitive model of skill acquisition that aims to elucidate how integration and reuse can be realized in the cognitive system.

The integration of knowledge during skill acquisition has many facets, and the proposed model helps to demonstrate how integration can be grounded in the cognitive system. One critical challenge is how information can be shared, or passed, among component skills that acquire new information and/or process acquired information. The proposed model suggests that information sharing can be realized by means of indexicals that point to information stored as problem state—task-specific information associated with the brain’s parietal areas (Borst et al., 2010). Indexicals themselves (e.g., a term such as “target”) can be acquired from the task context—for instance, from an experimenter’s verbal instructions. When a skill acquires the information associated with an indexical (e.g., a word corresponding to “target”), the skill uses the indexical as a reference to store the information in the problem state. Then, when another skill requires this information, it can reference the same indexical and thus retrieve the information from the problem state.

Another critical challenge of integration arises in how integration may adapt over time or with practice. The model suggests that two learning mechanisms are central to this process: production compilation, which combines rules to form efficient task-specific rules; and utility learning, which, with practice, gradually favors the newly formed compiled rules. During this process, the indexicals that reference problem-state information become embedded into the compiled rules—in essence, defining new chunk representations (specifically new chunk types with indexicals as new slot names). Thus, not only is the proposed model acquiring new procedural skills but it is also building new declarative representations to encode the information acquired for newly learned tasks.

Fig. 9. Histogram representing the number of rules in the model that fire for the given number of tasks.
The reuse of component skills requires that both the integration process and the skills themselves generalize to different task contexts. The model demonstrates that this is possible even across disparate task domains. The model’s reliance on indexicals helps to achieve this, because it can adapt flexibly with new (procedural and declarative) representations depending on the task instructions. This sense of skill reuse is closely related to Singley and Anderson’s (1989) theory of cognitive transfer, in which transfer between skills can be expressed in terms of the overlap (reuse) of production rules between those skills. A consequence of this overlap is that practice and improvement of one component skill should transfer to other tasks that utilize the same skill (e.g., Anderson et al., 2011; Lee & Anderson, 2001). The model here relates most closely to “vertical transfer” (Haskell, 2001) that combines prerequisite skills to enact higher level knowledge. However, the model does not specify the manner or context in which the component skills themselves were learned, as has been done in some previous work on transfer (like Singley & Anderson, 1989). Nevertheless, the model has implications for transfer in that it demonstrates how the use of indexicals to store and access information can help achieve flexible integration with other skills as is critical for transfer.

The model has some important limitations that should be noted. With regard to scope, because it focuses on instruction following, the model does not currently address other means of skill acquisition such as analogy (Gentner, 1983; Holyoak & Thagard, 1996) and discovery (Dunbar, 1993; VanLehn, 1991). The model also does not yet address the issue of individual variability. As its account broadens to these areas, the model’s foundation in a cognitive architecture should provide guidance in future development. For example, there have been key developments in understanding and modeling individual variability in a cognitive architecture (e.g., Lovett, Daily & Reder, 2000), and the ideas therein could be carried over to the current model in an attempt to predict behavioral variability and validate the predictions with data from a similar variety of tasks. As another example, the ideas behind previous Soar models of explanation-based learning (Huffman & Laird, 1995) and learning from exploration (Howes & Young, 1997) could be incorporated into this model to extend its reach to other forms of learning.

With regard to the account of instruction following, one limitation centers on the representation of instructions and its implications for skill acquisition. In particular, the instructions used here are overspecified in the sense that they provide an unambiguous sequence of steps for the model to follow; in a more realistic experimental situation, the instructions may not provide an exact specification of behavior, but rather require the human participant to infer steps and fill in the gaps in the instructions. For example, realistic instructions analogous to those in Table 1 (for the paired-associates task) may not direct the participant to remember the mapping from word to number, or to repeat the trial, and yet the participant would be able to figure out the missing steps (either through learning or by asking the experimenter). For the equation-solving instructions in Table 3, the instruction “type-and-check” is likely overspecified in that natural instructions might only direct the participant to type the result; checking the result after typing would come at the discretion of the participant. Thus, whereas the model follows the instructions to the letter, human participants could adapt the specified behavior to include missing and/or
desired steps. The complexity and difficulty of such adaptation could help to explain potential differences in task performance that arise from differing sets of instructions (as in Bibby & Payne, 1993).

The overspecification of instructions is a limitation in another way, in that the instructions define a single prescribed strategy for performing the task. In contrast, most tasks have a rich space of possible strategies that could lead to successful performance. Some recent models of task performance (e.g., Brumby, Howes & Salvucci, 2007; Howes, Lewis & Vera, 2009) have focused on this issue by using models to explore the strategy space and determine the most likely strategies based on each strategy’s potential payoff. In this approach, instructions can be viewed as constraints on task execution, and behaviors can be viewed as instances of strategies that fall within these constraints. This previous work explored the constrained strategy space by generating many possible models that satisfy the constraints. Extending the current model to explore strategies would require a very different approach in which the model itself evolved and tested strategies under the psychological limitations of the cognitive architecture, likely with higher level metacognitive strategies that govern exploration (e.g., Sun, Zhang & Matthews, 2006).

The current model’s inflexibility of processing could also be extended through the use of context. Recent work (Taatgen et al., 2008) has demonstrated the benefit of “context instructions” that include when to apply a step and what the step accomplished. The critical difference with context instructions is that they specify a partial order of task steps (vs. a complete predetermined sequence). Then, when unexpected events occur—like a change in the visual stimuli or the failed completion of the previous step—the person can adapt in a flexible way and continue using the next step that satisfies the current conditions. The model currently has no way to adapt its behavior in the face of unexpected events, and a future model would benefit greatly from incorporation of context information.

These limitations notwithstanding, the proposed model’s unified account of skill learning serves as a step toward Newell’s (1990) grand vision of a unified account of cognition and behavior. In fact, the model attempts to benefit all three aspects of Newell’s (1973) prognosis for unified theories: (a) building “complete processing models” that include perceptual, motor, and cognitive processes (as provided by the ACT-R architecture); (b) addressing complex tasks, including the process of encoding and interpreting instructions; and (c) developing a single model that generalizes to a diverse set of tasks. Skill acquisition is certainly one of the central components of human cognition, and there is much to be gained by placing skill acquisition in the context of a unified theory.

Acknowledgments

This study is supported by Office of Naval Research grant N00014-09-1-0096. I thank John Anderson, Andrew Howes, Richard Young, and Niels Taatgen for numerous insightful comments and helpful suggestions.
Notes

1. The canonical implementation of ACT-R is available at <http://act-r.psy.cmu.edu>. The modified version of ACT-R used in this article is available at <http://cog.cs.drexel.edu/act-r>.
3. For several years, the publicly distributed implementation of ACT-R has included an optional ability to specify variables for slot names. The current modifications to ACT-R include this feature as a standard feature—a natural extension of the elimination of chunk types.
5. The two paired-associates models appear in Unit 4 of the ACT-R tutorial: <http://act-r.psy.cmu.edu/actr6/>.
6. I thank Richard Young for his helpful suggestions in framing this argument.

References