Applying Primitive Elements Theory for Procedural Transfer in Soar

Bryan Stearns (stearns@umich.edu)
John Laird (laird@umich.edu)
Mazin Assanie (mazina@umich.edu)
University of Michigan, 2260 Hayward Street
Ann Arbor, MI 48109-2121 USA

Abstract

Detailed transfer of procedural knowledge has been modeled in Actransfer, an extension of ACT-R, by combining the primitive memory operations of productions (PRIMs) with the architecture's procedural learning mechanism (Taatgen, 2013c). This work explores whether these same principles can be applied to the Soar cognitive architecture, which uses different models of working memory and procedural learning. We confirm that these principles can transfer to an unmodified version of Soar. Our research contributes a novel model of skill learning based upon a deeper level of primitive skill composition than described in the PRIM model that is suitable for unbounded working memory architectures, and which yields transfer profiles similar to those revealed in human studies.

Keywords: cognitive transfer; skill acquisition; cognitive training; cognitive architecture; ACT-R; Soar.

Introduction

For decades, cognitive architectures (Newell, 1990) have been proposed as unified theories for achieving the general capabilities found in the human mind. Transfer of procedural knowledge is one such capability (Taylor, Kuhlmann, & Stone, 2008). The primitive elements theory of cognitive skills, proposed by Niels Taatgen, has recently achieved success in improved modeling of human transfer (Taatgen, 2013a, 2013c). Taatgen implemented his theory in a new cognitive architecture, Actransfer, which extends the ACT-R cognitive architecture (Anderson, 2007) to include this transfer modeling. Taatgen noted that, while his implementation was based upon ACT-R principles, the core ideas could be applied to other theories of cognition (Taatgen, 2013b). The work described here pursues this line of research in the Soar cognitive architecture (Laird, 2012), and briefly compares primitive elements learning in Soar with that of Actransfer and humans performing a common task.

A significant contribution of this work is that we extend Taatgen's theory to include a more general level of learning that supports unbounded, dynamic architectural memory structures and that provides a deeper model of the nature of skill composition.

The following sections first describe Actransfer and the underlying PRIM model before introducing the PROP model, the novel application of these ideas in Soar.

Background

The identical elements model of learning by Thorndike (1922) states that transfer among tasks occurs only inasmuch as there are identical cognitive elements shared in task representation and execution. Singley and Anderson (1987) proposed a more precise definition through the identical productions model, in which complex cognition is controlled by procedural knowledge represented as *if-then* production rules. This representation allows transfer to the extent that different tasks share identical productions.

Singley and Anderson (1985) evaluated the identical productions model using ACT, a precursor to ACT-R. By comparing the model with human performance, they found that in some cases it produced a fairly accurate relative prediction of human data. In other cases, only half the transfer measured in human participants was achieved, indicating that the model was incomplete.

Primitive Elements Theory & PRIMs

Taatgen proposed the primitive elements theory (Taatgen, 2013c) as a modification of the identical productions model of transfer. There are two aspects to the theory. First, primitive elements of transfer are defined not as complete, task-specific productions, but as the individual task-general memory operations used in productions, such as the general action of copying a value from one memory slot to another. Second, the theory outlines a model of human skill acquisition and transfer based on hierarchically composing these primitive operations through practice into task-specific rules, using a procedural learning mechanism. Composed skills will share identical elements if they employ similar memory operations.

Actransfer, the implementation of these ideas, was applied to human transfer experiments (Chein & Morrison, 2010; Elio, 1986; Singley & Anderson, 1987), achieving results that both align with human data and provide deeper theoretical explanations for transfer than earlier models.

Primitive elements theory implemented in Actransfer represents what Taatgen called the *PRIMitive information processing element* (PRIM) model. In this model, PRIMs are the fundamental, innate memory operations that can be composed through practice into any skill. Compositions of PRIM sequences are transferable when shared among rules.

Conditions	Action	Other
Compare Equal Compare Unequal Empty Nonempty	Copy	Load Task-Specifics

Table 1: The six types of PRIMs. Loading task-specifics is a PRIM that loads values into memory slots for use by other operations.

```
(PRIM instruction-example
  retrieval.slot2 <> goal.slot2
  retrieval.slot2 <> nil
==>
  query.slot1 := consts.slot1
  query.slot2 := retrieval.slot2
  action.slot1 := consts.slot2
  action.slot2 := retrieval.slot2
)
```

Figure 1: Example pseudo-code primitives of a rule with two compare conditions and four copy actions. Slots are organized under buffers such as `retrieval` or `action`.

Memory Operations

Actransfer working memory is composed of a set of buffers, each having a fixed number of memory slots. All Actransfer skills (rules) are represented as sequences of six types of memory operations, the classes of PRIMs listed in Table 1. This restricted set of compare and copy actions was chosen to reflect previous ACT-R work in basal ganglia modeling (Stocco, Lebiere, & Anderson, 2010).

While there are six PRIM types, there can be many instances of each corresponding to interactions among different memory slots. This is analogous to how computer programs use a finite set of register operations in an assembly language, such as ADD or LOAD, but can apply these among registers in many ways. In Figure 1, while each action is a copy operation, each is a different primitive because it copies using different slots. When Actransfer was configured with 31 working memory slots, this resulted in 1,693 PRIMs for the combinations of these slots with each type of operation (Taatgen, 2013c). Different rules share PRIMs if the same slot operations are used (regardless of the values in those slots).

Independence from values in memory slots makes a PRIM task-general. Instead of using conditions such as `buffer1.slot1 == "foo"`, the architecture preloads constants into a reserved set of slots, effectively making the condition `buffer1.slot1 == consts.slot1`. Thus, a different rule using different constants, such as `buffer1.slot1 == "bar"`, still employs the same primitive. This generalization allowed much of Taatgen's novel transfer across tasks.

Procedural Learning

An Actransfer agent learns a skill by rehearsing it step-by-step according to declarative knowledge recalled from long-term memory. Practice is gradually converted into procedural knowledge. These declarative instructions describe rules as sequences of PRIMs applied with specific constants, as shown in the bottom row of Figure 2. When the agent lacks applicable procedural knowledge, it recalls a list of instructions that describe a single rule, and then sequentially evaluates each instructed condition and action.

Actransfer employs ACT-R's procedure compilation system to transform this practice into skill. Each PRIM instance is implemented as an innate rule in procedural memory. Each

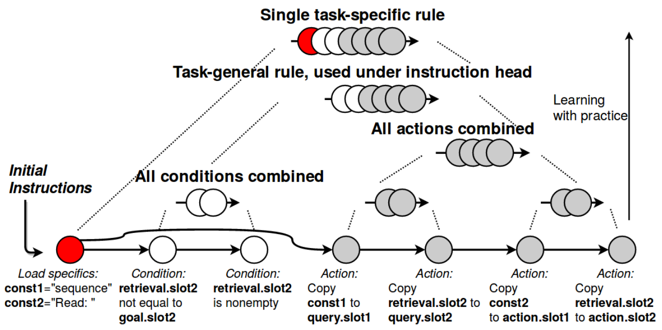
Single task-specific rule
Task-general rule, used under instruction head
Learning with practice
All actions combined
Initial Instructions
All conditions combined
Load specifics: const1="sequence" const2="Read: "
Condition: retrieval.slot2 not equal to goal.slot2
Condition: retrieval.slot2 is nonempty
Action: Copy const1 to query.slot1
Action: Copy retrieval.slot2 to query.slot2
Action: Copy const2 to action.slot1
Action: Copy retrieval.slot2 to action.slot2

Figure 2: Hierarchical clustering of PRIMs, adapted from Taatgen (2013c). Task-general conditions are shown in white, task-general actions are in gray, and the instruction head, which includes loading task-specific constants, is shown in red. All instructions begin with loading task-specific constants into memory. ACT-R production compilation combines repeated sequences of task-general condition and action PRIMs, until finally merging with task-specific constants, resulting in a single production. In this example, actions form a query to retrieve the next item in a sequence from declarative memory, while printing the current number to output.

Actransfer decision corresponds to firing a single rule. Whenever two different rules are fired in consecutive decision cycles, the architecture attempts to combine them into a new rule that can perform the work of both in a single decision. Such rules are not used initially the next time the same operations are practiced, but the more the original rules are practiced in sequence, the more likely it is that the combining rule will be used in their place. Once this replacement occurs, the new rule fires alongside other instructed rules, and the combination process repeats. As skills are practiced in this manner, procedure compilation learns an effective binary hierarchical clustering of skill elements, as shown in Figure 2. Compiled operations perform instructions in parallel rather than serially, decreasing execution time with practice and clustering. The final step of learning incorporates any task-specific constants into a single generated rule. With enough practice, all instructions are converted into such procedural knowledge, so that instruction recall becomes unnecessary.

Any intermediate compilations between the original PRIM sequence and the complete task-specific rule can be used for transfer, as the time to learn a new rule is less when portions of its instructions have already been compiled. The PRIM model thus predicts improved performance with repeated practice based both on incremental composition of operations and on reuse of such compositions.

PROPs - Primitive Skill Elements in Soar

Soar's working memory is not a fixed set of slots, but is an unbounded directed cyclic semantic graph rooted in a *state* ID, as in Figure 3, with each possible attribute path through the graph referencing a unique memory element. It would seem impossible to use primitive elements in Soar, since an unbounded set of memory locations would define infinitely many PRIMs. The solution to applying these concepts to Soar lies in recognizing that Soar's information processing

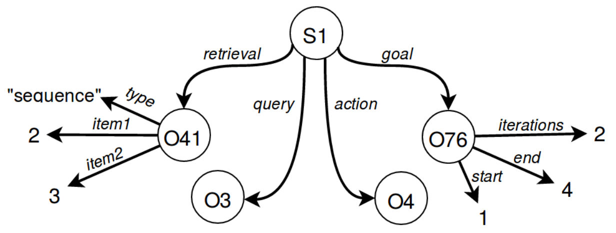

Figure 3: Example Soar working memory. Working memory is a directed graph rooted in a single state ID (shown as S1). Nodes can have any number of edges each pointing to a single value. Values can be more IDs or literals such as numbers or strings.

also differs from that of Actransfer.

Operators

A Soar decision cycle corresponds to the selection of a single *operator*, an architectural construct created, selected, and applied by rules to guide decision making. Selection of a single operator can involve firing several rules in parallel and/or serial, and rules can use wildcard variables to match a potentially dynamic state and shape of working memory.

An important distinction exists between an operator and its abstract definition. The set of rules describing an operator form the abstract definition, and an individual operator is the application of this definition in a decision to specific memory elements. Because Soar rules use variables, an abstract definition can be applied through an unbounded number of operators corresponding with the possible applications of the rule logic to states of the unbounded working memory graph.

We define the set of primitive memory operations within Soar decisions as the *PRimitive OPerator processing elements* (PROPs) that correspond to the most primitive operators from which all Soar agent processing can be composed. PROPs are defined through a fixed set of innate rules corresponding to each type of PROP listed in Table 2, which in turn correspond to the basic conditions and actions usable in Soar rules. While all Soar information processing can be composed from operators of these 22 abstract types, the number of possible PROPs is unbounded, since each corresponds to specific memory elements. For example, in Figure 4 there are four copy action operators, and these would be implemented through the same defining rules, but they remain distinct operators because they use different memory elements.

PROPs can only be applied if given specific memory references as arguments. The *memory referencing* PROP is what allows true support for unbounded memory by tracing a declaratively-known path through the memory graph and supplying the located element as an argument to another operator.[1] The number of memory references required to define each type of PROP is also shown in Table 2.

PROPs correspond to PRIMs as the primitive elements of decision making in Soar, but unlike PRIMs they are not innate. Initially, memory references must be reconstructed to

[1]This includes referencing any task-specific constants, and thus there is no separate primitive for that operation.

Conditions	**Actions**	**Preference Actions**	**Other**
Equal (2)	Copy (2)	Acceptable(1)	*Memory referencing*
Unequal (2)	Remove (1)	Indifferent (1)	
Exists (1)	Add ID (1)	Better (2)	
Not Exists (1)		Worse (2)	
Type Equal (2)		Best (1)	
Greater (2)		Worst (1)	
Greater/Equal (2)		Reject (1)	
Less (2)		Require (1)	
Less/Equal (2)		Prohibit (1)	

Table 2: Table of PROP types. In parentheses after each PROP is the number of memory element arguments required to apply the abstract definition into an operator.

```
pp {PROP-instruction-example
  (s1.retrieval.item2 <> s1.goal.end)
  (s1.retrieval.item2)
-->
  (s1.query.type := s1.consts.slot1)
  (s1.query.item1 := s1.retrieval.item2)
  (s1.action.out1 := s1.consts.slot2)
  (s1.action.out2 := s1.retrieval.item2)
}
```

Figure 4: PROPS instruction logic mirroring Figure 2. Because each primitive is self-contained, the full path from S1 must be specified for each working memory element.

redefine a PROP any time it is used. However, with practice, procedural knowledge to use memory elements is learned (see below), providing rules similar to the innate PRIMs of Actransfer. Thus, primitive memory access skills are based upon references actually used by the agent rather than the space of all possible memory operations. PRIMs in Actransfer may be considered a special case of the PROP model in which the working memory graph elements are arranged to match Actransfer slots and the agent is already trained in their use.

Procedural Learning

Soar's procedural learning mechanism also creates rules by combining the results of decision cycles, and can be used to compose skill elements hierarchically as in Figure 2. However, Soar does not compose rules that subsume pairs of sequential decisions, but instead summarizes any number of decisions and rules that are used to resolve a subgoal impasse.

An impasse is an event that arises when normal decision making cannot proceed, such as when no operator is available for selection or no procedural knowledge carries out a selected operator. When an impasse arises, a new substate is automatically created in working memory. Operators are selected in the substate to resolve the impasse. When the results of this work allow processing in the original state to resume, Soar automatically creates rules that summarize the rule firings and decision making that led to resolving the impasse. This learning process is called *chunking*. In similar future situations, the learned rules (*chunks*[2]) fire to avoid the impasse, replacing the substate processing. Soar chunks do not take effect gradually with practice as do ACT-R rule compositions, but fire whenever their conditions are met.

The PROP model is implemented through standard Soar rules that can be loaded into any Soar agent. As with Actransfer, declarative instructions describing the skills being taught are initially loaded into the agent's long-term memory, and are rehearsed during agent operation whenever the agent has no known operators to select. However, where Actransfer employed an architectural modification to automatically recall instructions when no decisions could be made, this behavior comes naturally in Soar through agent reactions to impasses. A Soar agent is also not restricted to only respond to an impasse with instruction practice, but could choose from available strategies according to the situation at hand. The PROPs agent by default begins instruction practice by recalling and following instructions within the new substate. Further impasses during instruction evaluation allow the agent to compile pairs of instructed procedures through chunking.

The amount of practice taken by a PROPs or Actransfer agent before compiling procedures into higher skills can affect whether those skills transfer across tasks (Anderson, 1982). Consider two rules, one composed of primitives A, B, C and the other of B, C, D. Ideally, the skill (BC) is learned that reduces training time for both rules, as opposed to (AB) and (CD), which cannot be shared. Actransfer does not replace element pairs with their combination until experience determines that they co-occur often across tasks. Once a combination replaces the original components, use of that rule prevents the architecture from further sampling co-occurrence of the old component rules in that context.

For Soar to combine skills based on co-occurrence, a declarative representation of experience is used to mediate the chunking process. The skill hierarchy of Figure 2, which implicitly reflects the ACT-R learning approach, is represented literally in the PROPs agent's long-term declarative memory, along with declarative measures of how often instruction items are experienced together. When two elements in this hierarchy co-occur beyond some threshold, T, the agent chunks them into a new skill element.[3] This co-occurrence reasoning is not integral to the PROP model, and would be unnecessary if Soar defined gradual confidence-based chunking.

Levels of Skill Composition

Through chunking, a PROPs agent learns three different types of knowledge that vary in complexity and provide speedup in different ways.

The first level of learning is of the use of memory through practice in applying PROPs from their abstract definitions, and is unique to this model. This processing is chunked into procedural knowledge when PROPs are compiled into new skill elements (the first level of composition in Figure 2).

[2]Unrelated to ACT-R *chunks*.

[3]Different metrics can be easily substituted for linear co-occurrence, but this simple measure works sufficiently well here.

The next type of learning is the normal hierarchical skill compilation, which is also the core of learning in Actransfer. Gradually improved performance comes from repeatedly composing instructed decisions into fewer, more task-specific rules through chunking, and from transferring such knowledge from previous compositions.

The third, outer-most level of learning is that which achieves independence from declarative instruction look-up. As with Actransfer, once general conditions and actions in an instruction set are merged as far as possible, a final learning step summarizes the instruction set into a task-specific rule that is executed when needed without fetching or evaluating instructions (the final stage shown in Figure 2).

By comparing these stages with the corresponding mechanisms of Actransfer, we can predict that a PROPs agent performing level-one learning should have a steeper initial performance curve as it learns to use its memory references. However, the main learning profiles of the architectures should be similar, including the amounts of transfer they provide, since they share the same core level-two learning process. We can also predict that the more aggressive nature of Soar chunking compared to gradual ACT-R rule compilation should result in the PROPs agent displaying a slightly more discrete and complete independence from instructions upon completion of its third level of learning.

Evaluation

Testing these predictions, we gave a PROPs agent declarative instructions to perform in a simulation of the Elio (1986) study that measured human transfer. Instruction logic and memory organization copied an Actransfer simulation by Taatgen (2013c), so that both model implementations learned to compose equivalent sequences of memory processing.

We ran two experiments. First, we tasked the agent to learn from scratch all three levels of knowledge described. Then, we repeated the experiment, but bypassed level-one learning by initially supplying the agent with all procedural knowledge necessary for memory referencing, mirroring Actransfer's use of innate PRIMs. We varied the learning threshold T, the number of times two skill elements must be seen together to be chunked into a new skill, and found values in the range of 16 to 24 provide comparable behavior to human and Actransfer results. $T = 16$ is used in data shown below.

Because the PROP model is implemented through rules rather than architectural modifications, maintaining instruction co-occurrence knowledge requires agent decision cycles, and this manifests as performance overhead during training. For a better comparison of the models, this overhead is omitted from this evaluation, as it is a reflection of implementation rather than part of the theory. All data are averaged over 8 samples, as in Taatgen's originally reported results.

Actransfer performance was originally reported in simulated time, but decision cycles are shown here to allow a more meaningful comparison across architectures. [4]

[4]Actransfer follows ACT-R in assuming 50 ms per decision by

Step	Calculation	Operation Type
Particulate rating	Solid $\times$ ($\text{lime}_4 - \text{lime}_2$)	Component
Mineral rating	greater of (algea/2)(solid/3)	Component
Index 1	Particulate + Mineral	Integrative
Marine hazard	($\text{toxin}_{\text{max}} + \text{toxin}_{\text{min}}$)/2	Component
Index 2	Index1/Marine	Integrative
Overall quality	Index2 − Mineral	Integrative

(a) Example procedure. Component steps only reference inputs. Integrative steps require remembering results of previous calculations.

SOLID	ALGAE	LIME	TOXIN
6	2	3	4
		5	8
		1	7
		9	2

(b) Participants look up hypothetical water sample data from among ten values provided per trial. For lime or toxin values, procedure instructions either specify the row index to look up or instruct to find the max or min value.

Figure 5: The Elio task

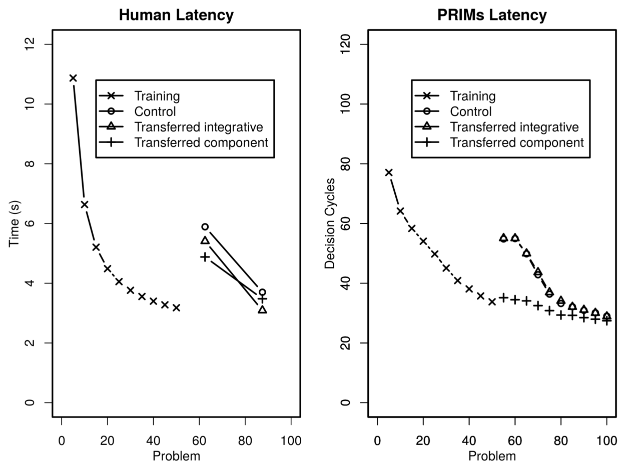

Figure 6: Human and Actransfer performance for component steps. Data for problems 1-50 show training performance. Data for problems 50-100 show performance for each of the three transfer conditions. Actransfer data were generated using supplementary materials from Taatgen (2013c).

The Elio task involved calculating hypothetical pollution rates based on water samples. Subjects repeatedly performed mental calculations using given input values. In the human study, subjects were trained in an initial procedure until they achieved perfect recall, and then tasked with performing it 50 times on various inputs (see Figure 5). Following this, subjects were assigned to 50 trials of one of three transfer conditions: transferred integrative, transferred component, and a control. The first two of these shared different types of calculations with the training, but the control did not. A basic ACT-like identical productions model would predict transfer from the training procedure to procedures that shared calculations, but would not predict transfer to the control. Yet transfer to the control was evident in the human results, as shown in Figure 6 through the faster initial performance of the transfer tasks compared to the training. Elio's transfer condition data measure the mean performances from the first and last 25 trials per subject. Depicted human training data shows Elio's power-law fit to human performance. In the original study, results for component and integrative calculations were reported separately. Only performance on component steps is shown here for brevity, as integrative results are comparable.

Control transfer is also reflected in the Actransfer agent. The *transferred component* procedure shows much additional transfer as well. This is because it shares component calculations with the training, allowing classic identical productions transfer in addition to primitive skill composition transfer.

We first ran our PROPs agent on the Elio task performing full learning of knowledge levels one through three. We then ran the agent with level-one procedural knowledge predefined for all relevant *memory referencing* PROPs. Performance for both experiments is shown in Figure 7.

Four results stand out. First, as expected, the initial transfer condition performances in the PROP model indicate the same rates of transfer as in the Actransfer agent. Second, unlike Actransfer, in both experiments the PROPs agent performance sharply converges to maximal performance, which is just under ten decision cycles. This is due to the discrete nature of Soar chunking, particularly with level-three learning, when independence from instructions is permanently achieved in a single chunking step. Third, the PROPs agent that only performs level-two and level-three learning roughly

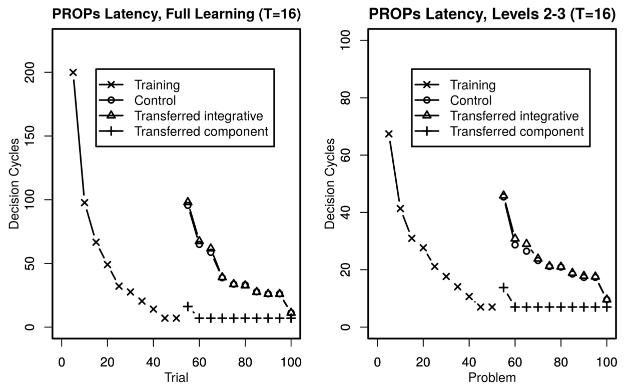

Figure 7: PROPs agent performance of the Elio task, measured in decision cycles. Hierarchy management overheads are not included. Left: Learning all levels. Right: Learning with predefined PROPs.

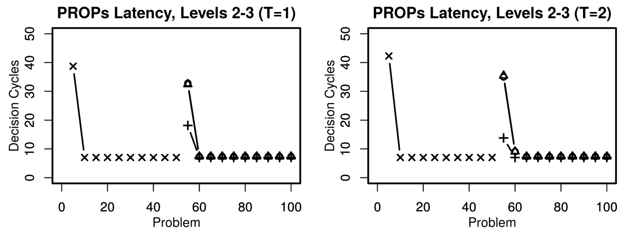

Figure 8: The progression from $T = 1$ to $T = 2$.

default, with additional time for operations such as long-term memory retrievals. We similarly assume 50 ms per Soar decision.

shows the same power-law performance as Actransfer, as is expected since they perform similar learning processes. Finally, one notes that if modeling simulated time using 50 ms per decision cycle, the PROPs agent with full learning of levels one through three performs at similar simulated time scales to the human model, beginning at 10 s, with the exception of then converging to maximal performance as discussed, ending at about 0.5 s.

Learning threshold T controls the sharpness of the learning curve, as well as the transferability of composed skill elements. A threshold of $T = 1$ causes near-instant skill acquisition, but makes blind combinations that might not be transferable. Through an analysis of the number of chunks transferred across procedures with varying T (not shown), we found that even a threshold as low as $T = 2$ allowed sufficient co-occurrence sampling for achieving near-optimal transfer in this task. Figure 8 demonstrates the improved initial latency of the *transferred component* case, which is not further improved with the higher threshold of Figure 7.

In summary, our experimentation indicates that the PROP model not only provides the same transfer as the PRIM model, but that deeper learning with memory references to suit dynamic memory also aligns with human performance.

Discussion

Primitive elements theory distinguishes among three types of skills: innate, task-general, and task-specific (Taatgen, 2013c). Innate skills are single primitives, task-general skills are combinations of primitives, and task-specific skills are the combinations of general skills with specific constants. These correspond to the three levels of *learning* in the PROP model. We theorize that level-one memory management knowledge would be learned (possibly developmentally) by human subjects prior to participating in the Elio task. Actransfer by contrast assumes a fixed configuration in which memory slots and their use are innate.

Actransfer's fixed set of PRIMs is useful in that they *must* in some respect be shared across any use of the architecture, just as registers must be used in any normal processor logic. In that model, the number of innate PRIMs expands combinatorially with working memory capacity, though only a subset might be used. In the PROP model, however, while transfer likewise depends on using a common set of memory references, PROPs only reflect skill elements used in practice.

ACT-R's and Soar's procedural learning mechanisms differ in many ways, yet provide similar models of learning with practice. However, as Soar chunking does not currently define gradual skill acquisition under uncertainty. To support such learning requires decision-making overheads that are not part of the PROP model, suggesting that architectural support for gradual confidence-based chunking provides a better fit to this sort of learning, and might be worth pursuing in Soar.

We have shown that despite differences between ACT-R and Soar models of working memory and learning, the primitive elements theory can be implemented in both to achieve similar results. In so doing, we introduced the PROP model for information processing in unbounded memory spaces through memory reference learning. The PROP model builds upon Taatgen's original PRIM model to provide a deeper and more general theory for the acquisition of cognitive skills.

Acknowldgments

The work described here was supported by the Office of Naval Research under grant number N00014-15-1-2058. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the ONR or the U.S. Government.

References

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, *89*(4), 369 - 406.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York, NY: Oxford University Press.

Chein, J. M., & Morrison, A. B. (2010). Expanding the mind's workspace: Training and transfer effects with a complex working memory span task. *Psychonomic Bulletin & Review*, *17*(2), 193–199.

Elio, R. (1986). Representation of similar well-learned cognitive procedures. *Cognitive Science*, *10*(1), 41 - 73.

Laird, J. E. (2012). *The soar cognitive architecture*. Cambridge, MA: MIT Press.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Singley, M. K., & Anderson, J. R. (1985). The transfer of text-editing skill. *International Journal of Man-Machine Studies*, *22*(4), 403 - 423.

Singley, M. K., & Anderson, J. R. (1987). A keystroke analysis of learning and transfer in text editing. *Human-Computer Interacttion*, *3*(3), 223–274.

Stocco, A., Lebiere, C., & Anderson, J. R. (2010). Conditional routing of information to the cortex: A model of the basal ganglias role in cognitive coordination. *Psychological Review*, *117*(2), 541 - 574.

Taatgen, N. A. (2013a). Diminishing return in transfer: A PRIM model of the Frensch (1991) arithmetic experiment. In *International conference on cognitive modeling.*

Taatgen, N. A. (2013b). *The gap between architecture and model: Strategies for executive control* (Tech. Rep. No. FS-13-03). AAAI.

Taatgen, N. A. (2013c). The nature and transfer of cognitive skills. *Psychological Review*, *120*(3), 439–471.

Taylor, M. E., Kuhlmann, G., & Stone, P. (2008). Transfer learning and intelligence: An argument and approach. In *Proceedings of the first conference on artificial general intelligence (AGI)* (Vol. 171, pp. 326–337). IOS Press.

Thorndike, E. L. (1922). The effect of changed data upon reasoning. *Journal of Experimental Psychology*, *5*(1), 33.