Running head: The nature and transfer of cognitive skills

The Nature and Transfer of Cognitive Skills

Niels A. Taatgen

University of Groningen

Word count: 25,256

Corresponding Author:
Niels Taatgen
University of Groningen
Institute of Artificial Intelligence
Nijenborgh 9
9747 AG Groningen
Netherlands
Email: n.a.taatgen@rug.nl

Author Note

This research was supported by ERC StG 283597 MULTITASK from the European Research Council. I would like to thank Stefani Nellen, Marieke van Vugt and Fokie Cnossen for comments on earlier drafts of this article.

Correspondence concerning this article should be addressed to Niels Taatgen, University of Groningen, Institute of Artificial Intelligence, Nijenborgh 9, 9747 AG Groningen, Netherlands, Email: n.a.taatgen@rug.nl

Supplemental materials

Detailed documentation of the models, and code to run the models can be downloaded from: http://www.ai.rug.nl/~niels/actransfer/supplemental.html

Abstract

This paper presents the Primitive Elements theory of cognitive skills. The central idea is that skills are broken down into primitive information processing elements that move and compare single pieces of information regardless of the specific content of this information. Several of these primitive elements are necessary for even a single step in a task. A learning process therefore combines the elements in increasingly larger, but still context-independent units. If there is overlap between tasks, this means the larger units learned for one task can be reused for the other task, producing transfer. The theory makes it possible to construct detailed process models of two classic transfer studies in the literature: a study of transfer in text editors and one in arithmetic. I will show that the approach produces better fits of the amount of transfer than Anderson and Singley's identical productions model. The theory also offers explanations for far transfer, in which the two tasks have no surface characteristics in common, which I demonstrate with two models in the domain of cognitive control, where training on either task-switching or working memory control led to an improvement of performance on other control tasks. The theory can therefore help evaluate the effectiveness of cognitive training that has the goal to improve general cognitive abilities.

Keywords: Cognitive transfer, Skill acquisition, Cognitive Control, Cognitive Architecture, Cognitive Training

The Nature and Transfer of Cognitive Skills

The goal of this article is to develop a theory and model of how cognitive skills are acquired, and how transfer between skills can be explained. The ability to learn and carry out complex cognitive skills is critical to human intelligence, yet is still poorly understood. There are many reasons to believe skills are not independent of each other, but are closely interrelated, and build upon each other. It is, however, hard to characterize this interrelationship precisely. The representation of skills presented in this article is a proposal for such an account, and thereby offers explanations for situations in which skills overlap.

With a few exceptions, the representation of choice to model skill acquisition is the production rule (Anderson, 1982; Newell, 1990). Typically, the knowledge necessary for a particular task is encoded as a set of rules, each with a number of conditions that match the current state of (working) memory and information from the senses, and a number of actions that modify working memory or initiate motor output. Even models that do not employ production rules as such either use a similar condition-action paradigm (e.g., Botvinick & Plaut, 2004), or abstract away the complexities involved (e.g., the role of the algorithm in instance theory, Logan, 1988). Despite their past successes in explaining complex human behavior, production rules have a number of disadvantages. A first problem is that productions rules are fairly complex representations. This presents a challenge when we try to answer the question of how these representations can be learned, and how they are represented in the brain. A second problem is that production rules are usually highly specific for a particular task, making it hard to characterize how skills are interrelated. The high specificity of rules can be attributed to two separate aspects. The first is that a production rule typically specifies multiple elementary comparisons and elementary actions. The second is that production rules incorporate specific knowledge elements. For example, production rules for doing multi-column addition may refer to columns, to the number one as something that has to be carried from one column to the next, to the addition of two digits and to shifting attention one column to the left. A set of rules to represent multi-column subtraction would therefore refer to a slightly different set of specifics, and multi-column multiplication would be different still. There are common elements between the three, in fact, iterative procedures occur in many other tasks, as we will see later on, but the specifics are different. How and when, then, does transfer occur?

In this article I propose the *primitive elements theory*, a way to break down production rules into their smallest possible elements, some of which are specific to the task but most of which are general. The resulting task-general elements control the flow of information in the mind, regardless of the specific content. When these smallest units are used to carry out tasks, a learning process combines them and eventually builds the productions rules that are typically used to implement task models. However, on the way the learning mechanism produces partial task-general rules that can also be used by other tasks. This generates the potential for transfer.

A Computer Analogy

There is always a danger in drawing analogies between human cognition and computers, but in this case it may help to improve understanding of the model I propose. At the lowest level, a computer program is built out of *machine code*. Machine code consists of a fixed set of instructions, each of which perform primitive actions within the computational system. The function these instructions have to perform is to move around information between the different components in the computer. These components can be external, like peripheral devices and memory, or internal to the CPU, like the arithmetic unit and internal registers that store intermediate information (accumulators), or serve a function in task control (program counters, the instruction register, a pointer to the stack, etc.). With a small and finite set of instructions, computers are capable of implementing any algorithm. One of the challenges of CPU design is to formulate a good instruction set, which should neither be too large or small, and be comprised of instructions that are neither too simple or too complex.

The primitive elements model has similarities to the machine code model. It assumes that cognition has several specialized units for perception, output and memory, which have to be coordinated by atomic "machine level" instructions. Similar to the CPU, there is also a need to store intermediate results, that is, a working memory, and to keep track of task control. The chief conceptual difference, though, is that the instruction set of a CPU is fixed, but the set of primitive skills within the cognitive system evolves through a learning process. Starting from the smallest possible skills, combinations are built that recur often, and are therefore useful in many different tasks. This will be the basis for explaining transfer: training on a particular task evolves the available set of operators towards that task, and if those operators and their partial predecessors are also useful for a new task, there will be transfer.

Theories and Models of Skill Acquisition and Transfer

The dominant idea in models of skill acquisition is that the starting point of learning a skill is a set of problem-solving strategies, often called weak methods, which are very general, but also very inefficient. The process of skill acquisition involves some form of specialization that creates efficient knowledge specific for a particular task. Different theories have proposed different mechanisms to accomplish this, but they all share this basic principle (Anderson, 1982, 1987; Laird, Rosenbloom & Newell, 1986; Logan, 1988; Taatgen & Anderson, 2002; Taatgen & Lee, 2003). Models based on these theories have been very successful in explaining many aspects of skill acquisition, but their limitation is that they divide knowledge into two categories: a fixed set of general strategies, and learned, task-specific strategies. This division raises a number of questions. The first concerns the origin of the general strategies. Anderson (1982) assumed they are innate. Newell (1990) expressed uncertainty about the status of general strategies, but did not offer mechanisms that explain how they can be learned. A second question is whether this division is accurate, because if it

were, the consequence would be that all learned skills are independent of each other. This is at odds both with the intuition that skills build upon each other, and, as we will see later in this article, with experimental evidence.

The idea that skilled knowledge consists of general strategies and task-specific knowledge is not unique to production system theories. Several neural network models of complex tasks employ the same paradigm, albeit in different terms. For example, models of learning the past tense (Rumelhart & McClelland, 1986) and the balance scale task (McClelland, 1995) employ task-specific networks with task-general learning mechanisms (associative learning and back-propagation, respectively). An exception, to some extent, is the model of routine sequential action by Botvinick and Plaut (2004), which can model how actions related to making tea can intrude during making coffee. Their model selects actions mainly based on the current context instead of a particular goal.

The division of knowledge into fixed task-general and learned task-specific also influences how we think about transfer. Already the term transfer suggests that we have to take something from one task and "transfer" it to another task. An assumption about transfer is that it is a rather limited phenomenon. The supposedly limited role of transfer has a long tradition, going back to Thorndike (1922, Thorndike & Woodworth, 1901). Thorndike was critical of the *doctrine of formal discipline*: the idea that by learning Latin and Mathematics the brain is trained like a muscle, enabling it to do a variety of other things unrelated to the topics that were studied. The idea of formal discipline has a long tradition, and was first formulated by Plato in *de Republica*. Thorndike introduced the *theory of identical elements* as a replacement. According to this theory, transfer between two tasks only occurs insofar as knowledge elements are identical. For example, the only reason why it is easier to learn French after Latin, is that many words in Latin are similar in French. Thorndike demonstrated limited transfer in several experiments, for example in mathematics (Thorndike, 1922, but see Singley and Anderson, 1989, for criticism on Thorndike's experiments).

Singley and Anderson (1985) developed a modern version of the theory. They argued that Thorndike's theory lacked precision: what exactly is an element of knowledge, and when are two elements truly identical? They proposed the production rule as the element of transfer, and used the number of identical productions between two tasks as a measure of potential transfer. As a demonstration of this approach, they examined transfer between text editors. In one of their experiments, subjects had to learn to edit text using one of three editors, and then switched to a different text editor. The experiment, which we will discuss in detail later on, demonstrated substantial transfer between text editors. At approximately the same time, Kieras and Bovair (1986) outlined a similar theory of transfer with production rules, showing that the amount of extra time needed to learn extensions to particular procedures can be predicted by the extra number of productions needed for that extension. In all these cases, transfer was a matter of taking task-specific knowledge from one task and using it for another, semantically similar, task.

To predict the exact amount of transfer, Singley and Anderson encoded each of the three editors in a production system model based on a task analysis by Card, Moran and Newell (1983). The three models had several identical production rules, especially those involving the higher level planning of the edits. More specific rules, in particular the specific keystrokes controlling the edits themselves, differed between editors. In addition, the two line editors showed a larger overlap with each other than with the screen editor. To account for transfer, they calculated the percentage of overlap in production rules between two editors, taking into account the frequency in which a particular rule was used. On this basis they were able to account for a large amount of the transfer, but not all of it. In particular, the model was only able to account for half of the transfer between the line editors and the screen editor: while transfer in the data amounted to 62%, the model only showed 33% transfer. The model was clearly missing something.

In addition to this shortcoming, the identical productions model is underspecified in that it does not explain how different levels of generalization can be represented and learned. In all currently popular cognitive theories based on production rules, productions are from the outset either specific to a task, or task-general. Task-specific productions match a particular goal, so in the case of text editors, the choice is between a production that applies to a specific editor, or one that is used for text editing in general. However, when learning a particular text editor it is impossible to tell in advance whether a particular piece of knowledge is specific to that editor, or can be generalized to other editors.

A further complication of assessing transfer between tasks is that the level of transfer is not necessarily fixed, but may change with level of expertise. For example, it makes intuitive sense that the basic skills acquired in learning Latin may transfer very well to other languages, but as skills becomes more specific their overlap decreases. There is some limited experimental evidence for this. Frensch and Geary (1993) found little transfer from varying amount training on simple additions to multicolumn additions, even though subjects became slightly faster at both. They explained this with by assuming that their subjects were already very skilled at simple additions, so that additional training had very little general benefit. Anderson and Fincham (1994) study with an analogical reasoning task revealed diminishing transfer with practice between different types of problems within the task.

If transfer is so hard to explain through procedural knowledge, maybe we should look at declarative knowledge as the main source of transfer. Many theories are centered on analogical transfer (Forbus, Gentner & Law, 1995; Gentner, 1983; Holland, Holyoak, Nisbett & Thagard, 1986; Hummel & Holyoak, 1997). The idea is that to carry out a new task, knowledge about a similar task is retrieved from memory, and adapted to fit the new situation. Analogy is potentially a general strategy that can become increasingly powerful as the number of available examples in memory increases. If all transfer can be based on analogy, procedural knowledge might not be involved in transfer at all.

One of the problems with declarative transfer through analogy is that there are many studies that show that it is not a particularly dominant strategy in human reasoning. The most infamous example of this is the failure of people to solve an analogy problem in which they are first told a story about a general who captures a fortress by breaking up his army into smaller groups of soldiers to avoid setting off explosives that are only triggered by a large force. Only a few of the subjects in studies by Gick and Holyoak (1980) managed to use this example to solve an analogous problem about a surgeon who wants to use radiation to treat a stomach tumor, but wants to avoid burning healthy tissue. Similarly, Reed, Ernst and Banerji (1974) found no transfer between the Missionaries and Cannibals problem and an analogous Jealous Husbands problem.

However, a study by Day and Goldstone (2011) did find positive transfer between a task in which subjects had to control a ball that oscillated between two elastic bands and a task in which the population of a city had to be stabilized. The tasks were very different on the surface, but they did share the same underlying principle. According to Day and Goldstone, transfer was successful in this case not because subjects made an explicit analogy, but because they could reuse parts of the mental model from the ball task for the population control task.

Another set of studies by Rickard and colleagues (Rickard, Healy & Bourne, 1994; Rickard & Bourne, 1996; Bajic, Kwak & Rickard, 2011) show that people favor the use of facts from memory exactly as they are learned. For example, people do not use the fact that 3+5=8 to calculate the answer to 5+3, but have a separate memory trace for that reverse fact, a model they call the Identical Elements Model.

Even though analogical reasoning may be a partial explanation for transfer, it does not seem to be the full story. Apart from the question regarding the extent to which people use analogical transfer, there is also the issue that it serves a different purpose than transfer in cognitive skills. The goal of analogical transfer is typically to find solutions to novel problems, whereas the goal of this article is to understand how the acquisition of an entirely new skill is helped by existing skills.

A type of transfer that is hard to explain through any current theory of transfer is *far transfer*. Far transfer refers to cases in which the two tasks involved are very dissimilar (Barnett & Ceci, 2002). Although similarity between tasks can be defined in several ways, we will speak of far transfer if the specifics involved are different between tasks (we will characterize this more precisely later on). Several recent studies show transfer between tasks that are certainly not similar. Jaeggi, Buschkuehl, Jonides and Perrig (2008) found that training on the N-back task improves scores on the Raven progressive matrices task. In addition, Karbach and Kray (2009) found that training task-switching improves performance on the Stroop task, a working memory task, and the Raven's test. Other studies also found positive transfer between dissimilar tasks (Chein & Morrison, 2010; Mackey, Hill, Stone & Bunge, 2011), but others did not: several attempts to replicate the Jaeggi et al. study failed

(see Redick et al., 2012), and a large study by Owen et al. (2010) also failed to find transfer despite similarities to earlier studies.

The key proposal in this article is that the smallest elements of skill are smaller than what is typically represented in a production rule. A production rule typically performs several functions at the same time. The theory presented here breaks down these rules into their basic information processing units, and splits them into task-specific information and general information processing patterns. This separation between the specific and the general allows automatic reuse of the general components of a skill. This solves the problem that Singley and Anderson had in accounting of the editor data: by having smaller units, more transfer can be accounted for, and by separating out the specific values, knowledge that is learned in the context of a specific editor can still be used for a different editor.

More generally, the theory can explain cases in which there is transfer between skills that are structurally similar but completely different in content.

Overview of the Article

The first question we will try to answer is: what are the smallest elements of skill, and how are these elements combined into more complex units? Part of this question is how the task-specific elements can be separated from the general items in order to promote maximal reuse of knowledge. This will first be illustrated with a choice reaction task, which will be broken down into eight elementary steps.

The next questions are: how are the appropriate skill elements sequenced, how do learning processes combine elementary steps into bigger units, and how can these be used to explain transfer. I will illustrate this with an example of counting, and show how this transfers to semantic inference.

In the second part of this article, I will go through a real example of transfer between similar tasks: the Singley and Anderson (1985) editor experiment. The models involved are relatively elaborate, but are nevertheless simple in structure. The primitive elements model not only provides a better prediction for the amount of transfer between the editors than Singley and Anderson's original model does, but also reproduces the learning process itself.

In the third part, I will develop methods for task control and hierarchy. Task control is not considered a separate mechanism or module, but instead consists of a set of strategies for handling goals and elements in working memory. I will demonstrate this idea using an arithmetic task developed by Elio (1986). In her experiment, subjects had to memorize a set of formulas, and then perform calculations using these formulas. Subjects then transferred to a phase in which some or all of the formulas were replaced by new ones. Key in this task is that it requires some form of subgoaling. We therefore need provisions in the model for task control, which, I will argue, are not architectural but strategical.

In the fourth and final part, I will show how the model handles far transfer. Both the editor and Elio task show large semantic and structural overlap. Now we explore how sharing

key components of cognitive control between tasks that are superficially very different can explain transfer. More in particular, I will look at the proactive and reactive strategies to cognitive control, and how training proactive control transfers to other tasks in which this type of control is beneficial. A first demonstration of this is an experiment by Chein and Morrison (2010), in which training on a complex working memory task led to a gradual improvement of working memory capacity, but also to an improvement on the Stroop task. A second example is an experiment by Karbach and Kray (2009), in which several days of practice in task switching not only improved performance on a different task-switching task, but also on the Stroop task and a working memory task.

The Smallest Elements of Skill

Beyond the strong ties of the formalism of the production rule to symbolic theories, there is a much broader agreement that some form of condition-action mapping is necessary to explain human sequential behavior (Botvinick & Plaut, 2004, Cooper & Shallice, 2000, O'Reilly & Frank, 2006). Stocco, Lebiere and Anderson (2010) made an effort to bridge the gap between symbolic architectures and a neural implementation with a neural network model of the basal ganglia. This model implemented a symbolic production execution and learning system. The general assumptions of Stocco's model, which is consistent with both the symbolic ACT-R (Anderson, 2007) and connectionist Leabra (O'Reilly & Munakata, 2000, see also Lebiere, O'Reilly, Jilk, Taatgen & Anderson, 2008) theories, are as follows.

Cortical areas have specific functions, such as vision, motor output, declarative memory, working memory, time perception, etc. Each of these systems has particular inputs and outputs. For example, the output of the visual system can be a representation of the object in the fovea, whereas the input consists of oculomotor commands that direct the gaze. On a more cognitive level, the input to declarative memory can be a partial pattern of what we try to recall, after which the output is the completed pattern. All the outputs of the cortical areas are placed in a common workspace (for example, the global workspace proposed by Baars, 1988, and Dehaene, Kerzberg & Changeux, 1998), or, in terms of ACT-R, a set of buffers. In both the ACT-R and Stocco model, the workspace is not a single area, but rather a collection of areas distributed throughout the cortex. According to Stocco's model, the role of the basal ganglia is to match patterns in this workspace, and carry out actions by routing information elements from one place to another in the workspace. Cortical areas connected to these elements of the workspace then carry out the associated actions.

Stocco's model uses a type of production rule that is much more restricted than the rules that are used in most production systems. First, a production can only inspect the fixed-format output of specialized cognitive modules, and is restricted to simple comparisons (equality and inequality). Second, the action of the production itself is restricted to moving information within the workspace. Cortical modules then perform meaningful actions with

that information. Figure 1 shows a global outline of the general architecture inspired by these ideas.

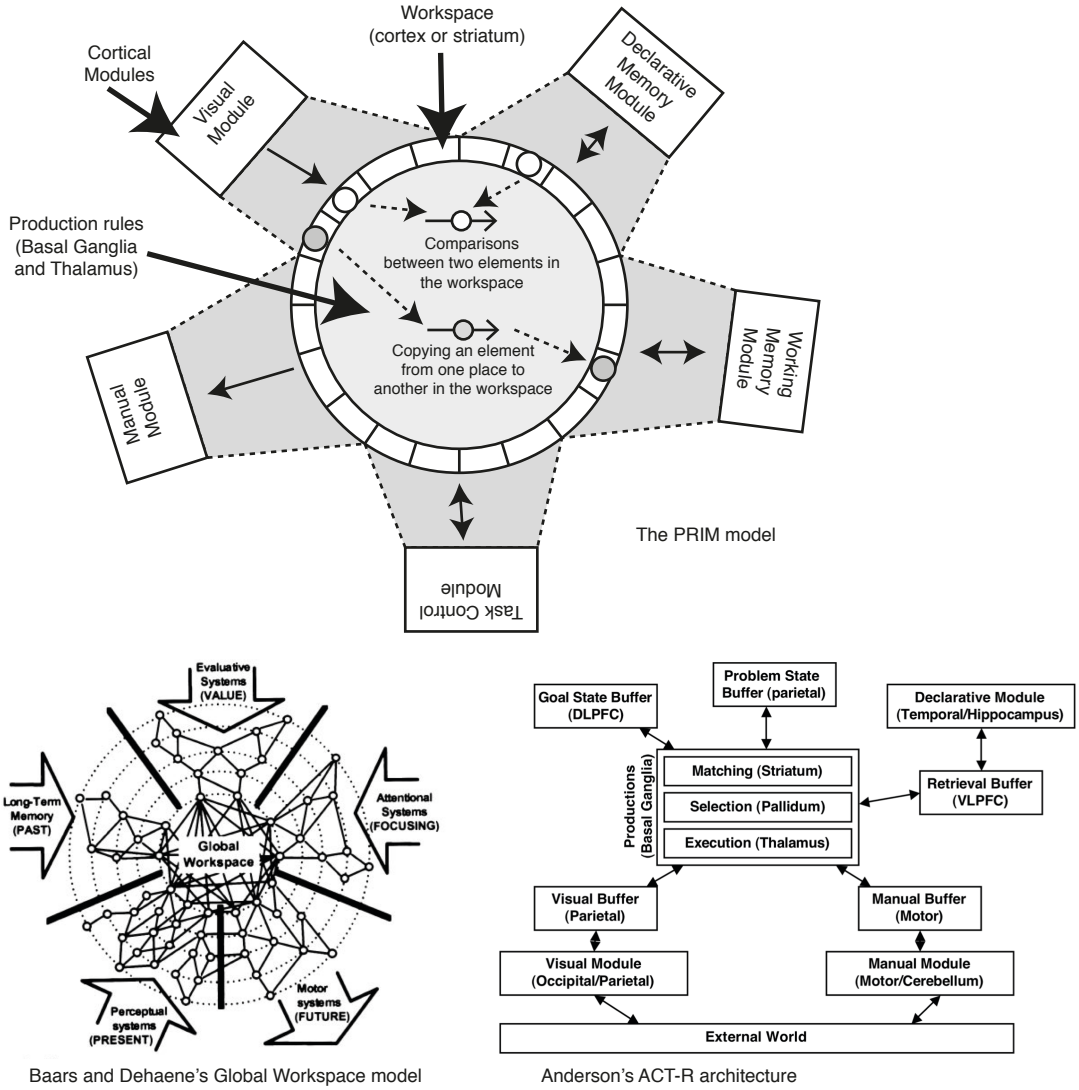

Figure 1. Global outline of the primitive elements model of skills, inspired on the neural network model by Stocco et al. (2010), the Global Workspace model (Dehaene, Kerszberg & Changeux, 1998), and the ACT-R architecture (Anderson, 2007). Specialized modules (the boxes on the outside) provide input to the workspace (the central ring in the Figure), or carry out actions placed in the workspace. The production rules in the center make comparisons between items in the workspace (indicated by circles here), and copy them. For example, a comparison can be made between a visual item and an item retrieved from memory, and another visual item can be copied to working memory. See also Jackendoff (1987) for another version of this general idea.

The Stocco et al. model suggests that the basic elements of cognitive skills are either comparisons between pieces of information in the workspace, or moving (or copying) information within the workspace. I will call these basic elements *Primitive information processing elements* (PRIMs). The basic idea of PRIMs is best explained using an example.

Example: A Choice Reaction Task

In choice reaction tasks, subjects perceive a stimulus, retrieve an appropriate response from memory, and then make that response. For example, they are instructed to press their index finger if they see the letter "A", their middle finger if they see the letter "B", or their ring finger if they see the letter "C". The example in Figure 2 assumes that perception, memory retrieval and output are carried out by dedicated modules that retrieve and deposit their inputs and outputs in a central workspace.

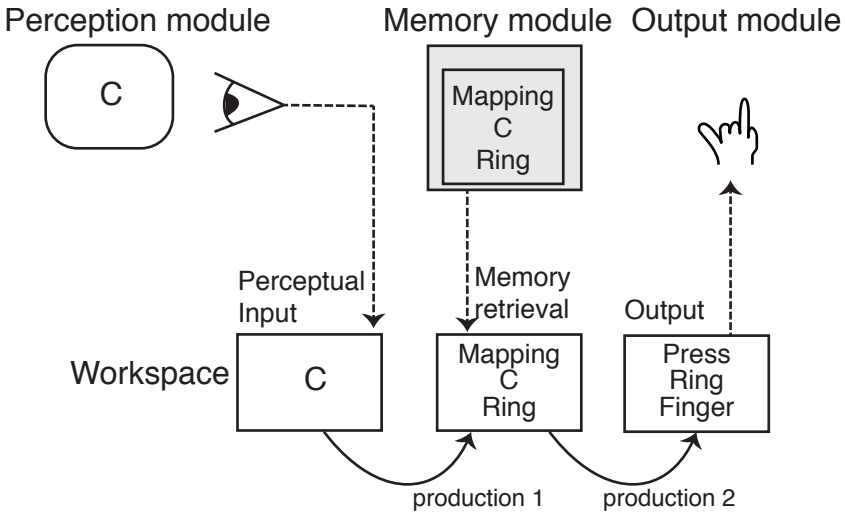

Figure 2. Outline of a choice reaction task. The perception module perceives the letter "C", and retrieves from memory that this letter is associated with the ring finger, and then presses the ring finger. Three cognitive modules (perception, memory and output) carry out specific subtask, but production 1 and production 2 control the flow of information between the modules at appropriate moments.

The two productions necessary for this task have to move the information (the letter "C" in the example) from the perceptual system to memory, and from memory to the output module (the "Ring" finger in the example). If we look at it in more detail, the productions do more than move information. First, they are triggered by the context of this task, and not otherwise. Second, they add information: in the case of production 1, the rule specifies that we are looking for a particular mapping of the letter to a finger, not just any fact in memory that relates to the letter. In the case of production 2, the rule specifies that it is a press action that is to be carried out, and not any other possible motor output involving the ring finger. In this view, production 1 consists of four elementary elements: checking that the task is CRT, checking that there is a perceptual input, moving "mapping" to memory retrieval, and moving the perceptual input to memory retrieval. These are the atomic elements of skill that we are after.

The first production is activated by a visual input, which is its first PRIM (Figure 3). Subsequently, it has a declarative retrieval that consists of two PRIMs: the constant

"mapping" and the visual input. Similarly, the second production is broken down into a match of the declarative retrieval and two action elements.

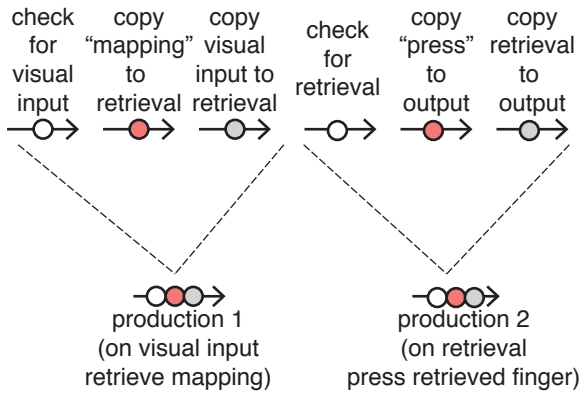

Figure 3. The two productions needed for a the choice reaction task broken down into elementary comparisons and actions.

There is still one problem: there are PRIMs that match or copy specific values, in our example "mapping" and "press". Moreover, production 1 and 2 only fire in the context of the choice reaction task, which means they have to check the current task. PRIMs cannot check values in de the workspace against specific constants, because that would require an infinite number of PRIMs. The solution is to bring specific values into the workspace, and then use general PRIMs to compare these values against other slots in the workspace. For this, we need an additional PRIM that puts these specifics into the workspace (by retrieving them from memory). Figure 4 shows how this is done. In Figure 4, as in Figure 3 and all subsequent figures, we will adapt the convention that colored nodes represent task-specific knowledge, white nodes represent general conditions, and grey nodes general actions.

In addition to showing how the specifics are entered into the workspace, Figure 4 also gives an impression of the learning process. PRIMs are themselves production rules, but lack a control component (i.e., in what order are they carried out). We will discuss this control component in the next section. To graphically preview this distinction we depict the PRIMs as consisting of two components: the elementary production rule (the arrows in the Figure), and a control component (the circles in the Figure). A production compilation process (Anderson, 1982; Taatgen & Anderson, 2002) combines PRIMs that fire in sequence into new larger production rules. For example, copying the specific to the retrieval and copying the visual to the retrieval can be combined into a new production that accomplishes both. This means that instead of only one primitive comparison or action, multiple are carried out in parallel. Because we have separated out the specific elements, many of the new productions that are learned are task-general rules. In Figure 4 these are the rules with only white and grey nodes.

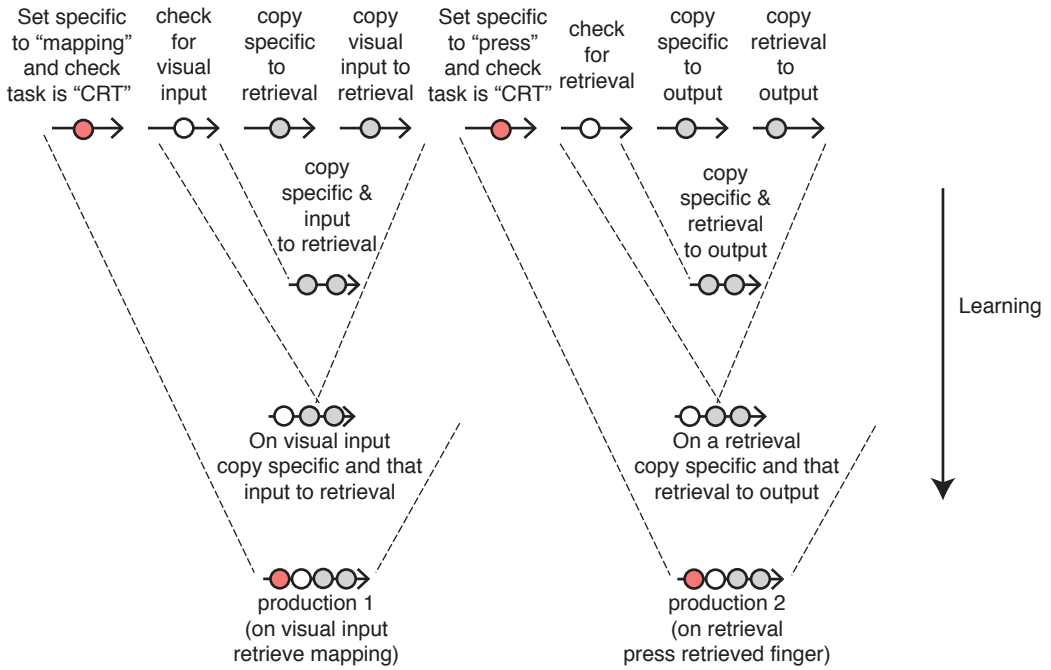

Figure 4. The Choice Reaction Task productions in which specific knowledge is moved to the front of each sequence. The Figure also shows how a learning process produces the task-specific rules, and the intermediate products, each of them incorporating two or three PRIMs.

The advantage of the task-general productions is that they can be used for other tasks. If another task has the same structure as a task learned earlier, it does not need to start the learning process with just PRIMS, but can already use learned productions that incorporate several PRIMs, resulting in a faster learning process, or, in other words, transfer.

The total number of PRIMs is a finite set, and depends on the size and composition of the workspace. More specifically, the workspace is subdivided in a fixed set of buffers (perception, declarative memory, working memory, etc.), each of which has a fixed number of slots to store information. Buffers can be input-only, output-only, or both input and output. The current implementation has 10 input slots, 7 output slots, and 14 both slots. The input slot includes slots that store specific values (e.g., "mapping" and "press") in the workspace.

For each non-output-only slot in the workspace there is a PRIM that checks whether it is empty, and one that checks whether it is not empty. Moreover, for each combination of two non-output-only slots there is a PRIM that checks whether the values in the two slots are the same, and a PRIM that check whether the two values are different. This adds up to (10 + 14) x (10 + 14 - 1) x 2 + (10 +14) x 2 = 1188 elementary comparison PRIMs (the white nodes). Furthermore, for each combination of a non-output-only and non-input-only slot there is a PRIM that copies the value in one slot to the other slot, so (10 + 14) x (14 +7) = 504 elementary action PRIMs (the grey nodes). This means that there are already 1,430,586 possible combinations of two PRIMs, and 805,896,780 combinations of three. These combinations cannot possibly be all useful, but a subset of them is, and it is up to long-term learning to discover and acquire these. Finally, there is one last PRIM, which is the step that checks the task and sets the specifics (represented by the colored nodes).

An observation that can be made regarding productions build out of PRIMs is that they do not seem to refer to variables, or variable binding. This is a consequence of the fact that ACT-R can only match items in buffers, and therefore variable binding means making a connection between two fixed locations , which is exactly what PRIMs do.

Although the size of the workspace can be adjusted in the implementation to accommodate additional modules, in particular perception and motor modules that are represented rather sparsely in the current implementation, the assumption of the theory is that the human global workspace has a fixed size.

Sequencing elementary skills

An aspect of "normal" production rules that PRIMs lack is control. A production rule typically combines a number of comparisons and actions because they perform a meaningful cognitive operation together. Because PRIMs carry out only single information processing steps, some other mechanism is needed to sequence them properly.

In order to accomplish this declarative memory is used to store the sequences of PRIMs. This fits general theories of skill in which a skill is first represented declaratively, and is then gradually proceduralized through training (e.g., Anderson, 1982). It is also analogous to computer programs, because in machine language, instructions in a program are stored in memory, and executing a program means retrieving these instructions from memory one at a time. However, the computer does not learn, and will therefore never become smarter at executing its programs.

The assumption is that knowledge to carry out a particular task is first encoded in declarative memory using what we call *operators*, the counterpart of productions in procedural memory. An operator consists of a number of linked memory items, each of which corresponds to a PRIM: a root item that represents the specific values, a list of conditions and a list of actions. Figure 5 shows one of the two operators for the choice reaction task.

A psychological reason for the split between a declarative operators and a procedural PRIMs is that procedural learning is very slow (e.g., Anderson & Fincham, 1994), and is therefore incapable of accomplishing the human ability to quickly acquire new tasks. Declarative memory, in which learning can be fast or even instantaneous, can therefore serve as a scaffold to build skills. In design as well as skill learning it can make a huge difference whether what we build can be made out of ready-made pieces, or has to be designed completely from scratch.

Each of the items in an operator is retrieved and carried out by a corresponding PRIM, one at a time. First, the root of an operator is retrieved, and any specific items are placed in the workspace (the Figures will refer to them with item1, item2, etc.). If one of the condition PRIMs does not match, the process is aborted, and process proceeds with another operator in memory. A gradual speedup of this process is achieved by the production compilation mechanism (Taatgen & Anderson, 2002). More specifically, this mechanism combines two

consecutive productions into one, while incorporating the contents of any declarative retrieval that happened in between these two productions into the new rule. In the context of carrying out operators, this means that parts of the operator are gradually incorporated into new production rules that are combinations of PRIMs.

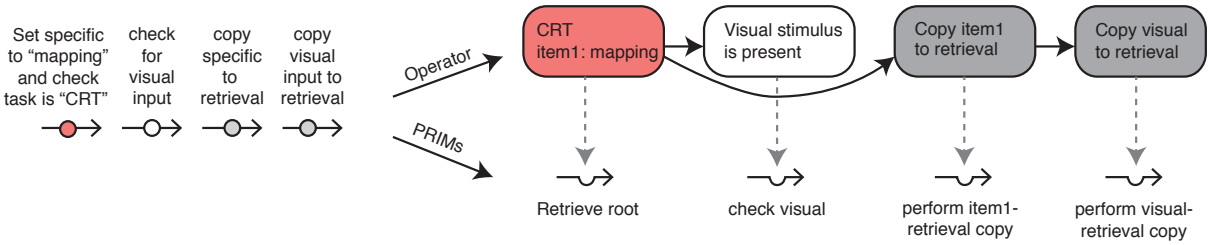

Figure 5. The order in which PRIMs are carried out is determined by operators in declarative memory. To depict this graphically, the original PRIMs are shows as arrows with a "slot" that needs to be "filled" by the operator to control the order in which PRIMs are carried out. The root of the operator (colored) contains the name of the task (CRT in the example), and any specific values ("mapping" in the example), and pointers to the list of conditions and actions. The operators represent the order in which PRIMs have to be carried out, but are otherwise passive structures in memory. The PRIMs, on the other hand, do carry out the actions, but do not know the order.

As learning progresses, fewer memory retrievals are necessary. Figure 6 illustrates this process step-by-step. In the intermediate examples, PRIMs are combined into productions that carry out several PRIMs in one step. Therefore, fewer components of the operator need to be retrieved until only the root of the operator is left. In the final expert stage, a task-specific rule is learned, and memory retrievals are no longer necessary at all.

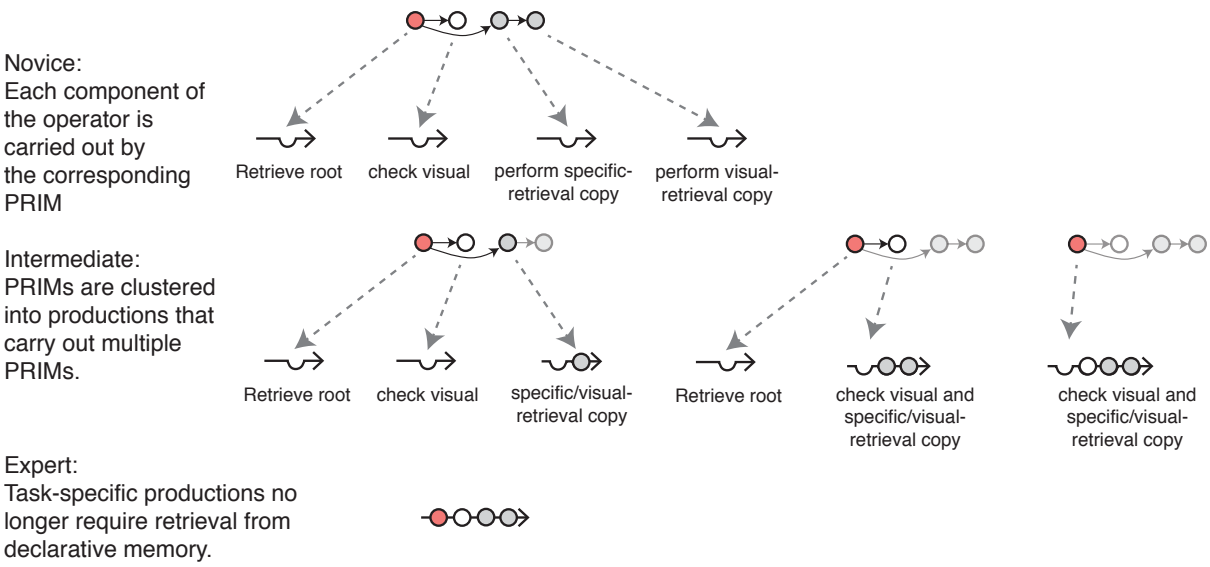

Figure 6. Novice (mirroring Figure 5), three intermediate, and expert stages of acquiring rules. The colored nodes represent task-specific knowledge, white nodes conditions, and grey nodes actions. Arrows with just an empty slot represent PRIMs, arrows with one or more nodes represent learned rules in which part (or all, in the case of the expert rule) of the operator has been incorporated.

Two specific choices were made in the representation of operators. The first is to split the conditions and the actions into separate lists, and not put them conditions and actions in one list. Splitting up the two will, as we will see, promote separate productions for testing groups of conditions and carrying out groups of actions, which in turn promotes transfer, and this is necessary to capture the transfer found in the experiments discussed later. The second choice was to arrange control of PRIMs through lists. The alternative would have been to represent them as a set, and not care about the order in which they are tested or carried out, improving overall flexibility. However, increased flexibility has a price: additional resources have to be devoted to keep track of which conditions and actions have already been accessed, and assessing the overlap between operators, which we will discuss shortly, would become harder.

There is no hard size constraint on the number of conditions and actions in an operator. However, module actions, such as declarative retrieval of facts, perception or motor output only take place between operators, creating natural boundaries between them.

Building Models

All models discussed in this article contain the following components. The first is the complete set of all possible PRIMs: 1188 condition PRIMs, 504 action PRIMs, and one PRIM to retrieve the root of an operator. Because these are always the same, they are predefined in the system. The second is a set of operators for each of the tasks, specified by the modeler and specific to that model. Each operator consists of a root item, and lists of conditions and actions. Root items are always unique for a tasks, but the conditions and actions can be shared, even partially, within and between tasks. The learning process is simulated by repeated execution of a particular task. Production compilation produces within task speedup by combining productions. Transfer between tasks is achieved by training the model on one task, and then testing it on a transfer task. If the training task produced intermediate task-general rules that are also useful for the transfer task, the model predicts transfer between the two tasks. In the discussion of the models, I often call operator components PRIMs, even though they are only supply the control components for the PRIMs.

The choice of a list representation for the condition and action components has one unfortunate consequence: the order of the conditions or actions has a huge impact on the amount of transfer. If two operators have C1 and C2 as conditions then there will be transfer if they appear in the same order, but not if the order is reversed in one of the operators. To reduce this lack of constraint, and not give the modeler arbitrary control over the amount of transfer, the implementation of the model tries to match the condition and action lists of new operators with existing condition and action lists. It then tries to reorder the lists to maximize the overlap. The reasoning behind this is that this is what people will try to do as well. Although they do not have detailed conscious access to the PRIMs, it is likely that the

process of constructing a strategy for a task tries to reuse as much existing knowledge as possible.

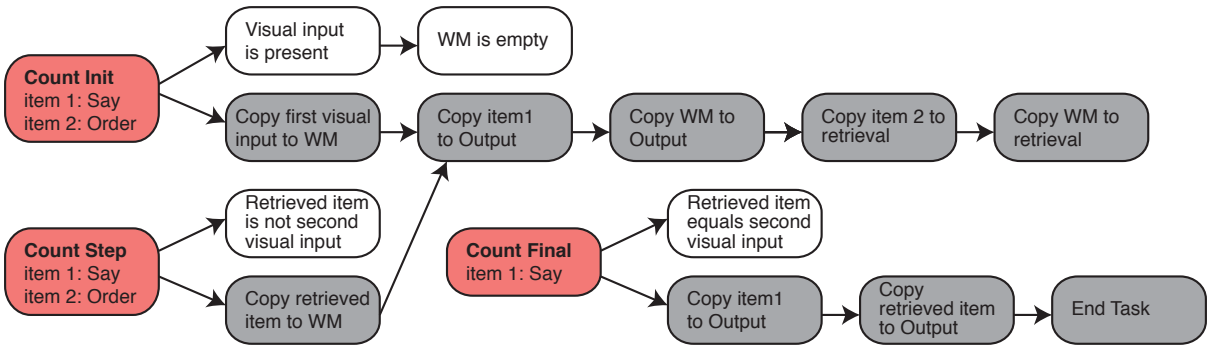

Figure 7. The three operators to do the counting task. Each of the three operators has a colored root node that identifies the task and the specific items. Each of the connected white nodes refers to a condition PRIM, and each of the grey nodes to an action PRIM. Note that the "Init count" and the "Count step" operator have a partial overlap in their action PRIMs.

An example of transfer: from Counting to Semantic reasoning

We will now look at the example of counting, and how counting can transfer to another iterative task. Counting in this example means: counting from a starting number to an end number, for example, count from 2 to 5. This model consists of three operators: one that initiates the count, one that proceeds through the counting steps until the final number is reached, and a third operator that finalizes the count. Figure 7 shows a representation of these three operators. In discussing this and future examples, we will mainly look at the operators because they are the representations that are different in each model, while the PRIMs are always the same.

As was already suggested in Figure 1, the workspace is divided into areas that connect to the different modules, and each area is further subdivided into a number of separate slots. In the counting example, we will need two input slots: one to represent the starting number, and one for the end number. We need one slot in working memory to store the counter and two slots to specify the output. Finally, we need three slots for declarative memory, because we will need to retrieve the order of numbers from memory, and these are represented in triplets such as "order, 2, 3", "order, 3, 4", etc.

Counting is now carried out as follows. The model starts with an empty working memory, and a visual input consisting of the start and end number. It will first retrieve one of the three root operators. Let us assume it immediately retrieves the right operator: "Init count", which consists of two specific items ("Say" because that is will be part of the output, and "Order", because it will later retrieve a fact of that type from memory), and a reference to the first element in the condition and action lists. The condition list consists of two PRIMs that are checked in order: is there a visual input, and is working memory empty (both true). After checking the conditions, the action PRIMs will be carried out one at a time: the first

visual (so the starting number) is copied to working memory (so we keep a counter), item 1 ("Say") is copied to the output, along with the contents of working memory (so the model says the first number), then the second item (which is "Order") is copied to declarative memory retrieval, and finally the number in working memory is copied to declarative memory retrieval. If the starting number is 2, this means we effectively ask declarative memory to complete the pattern "Order, 2, ...".

The second operator, Count Step, is triggered by a successful retrieval of an order fact (so the next number) from memory in which the retrieved number is not yet equal to the end number (the second item in the visual input). It then stores this number in working memory, and proceeds as in the first operator: say the number, retrieve the next number. Because these steps are identical between the two operators, the same action PRIMs are used. The third and final operator, Count final, has as condition that the retrieved number equals the end number. It the says that number and ends the task. It is important to stress that PRIMs only move around information within the workspace. It is up to the different modules, vision, declarative and output, to do the actual work.

The counting procedure is initially very slow, because every item in the operator has to be retrieved and executed separately. Fortunately, the production compilation process will, like in Figure 6, learn productions that can carry out multiple PRIMs in one step. Eventually, this learning process will produce three task-specific rules that no longer require the operator.

The task-general rules that it has learned during that process can be useful for other tasks, though. What the count model essentially performs is a form of iteration. Iteration is common in many tasks, even though these may not share the same content. One task with the same iterative structure as counting is a task of semantic reasoning, in which questions need

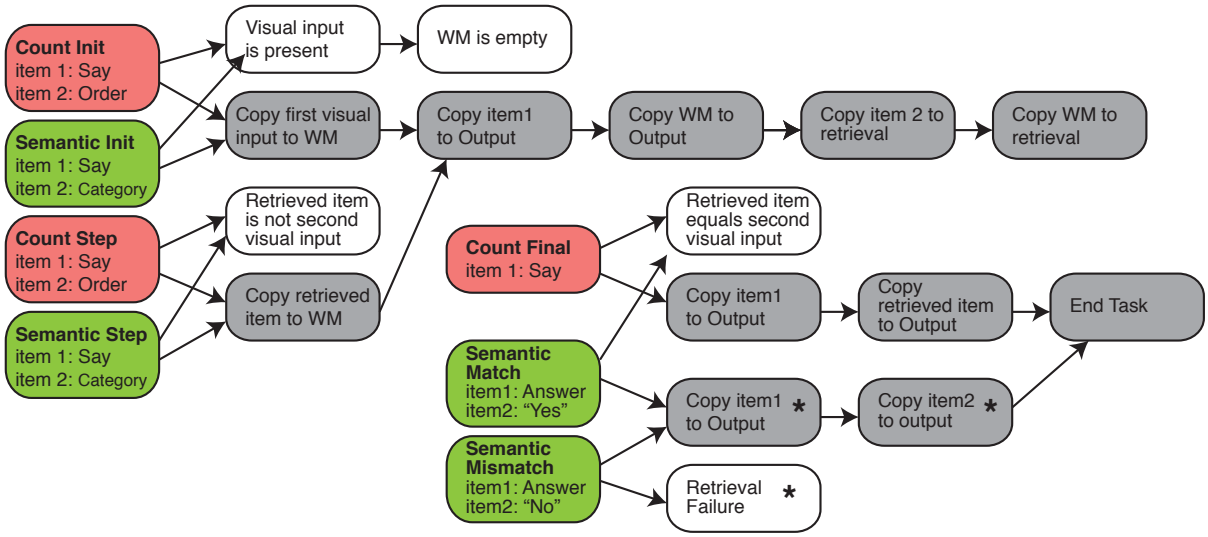

Figure 8. Memory structure of the operators after the semantic task has been added to the operators for the counting task (Figure 7). Addition of the semantic task require four task-specific root nodes (green), and three additional condition and action nodes (marked by *).

to be answered like “Is a canary an animal?”. This question can be answered by iterating through a semantic network of living things. In the example, first “A canary is a bird” has to be retrieved, followed by “A bird is an animal.” This is a bit like counting, with canary as the starting point, and animal as the end point. The only difference is that there is no guaranteed end to the iteration, because if we ask “Is a shark a mammal?” the iteration ends at the root of the semantic tree. To accommodate this, the model should respond with “yes” when it retrieves the end point, but should respond with “no” if any retrieval fails. Figure 8 shows the operators necessary for this task when they are added to memory after the count task.

The first two operators in the semantic reasoning model are structurally the same as in the count model, even though they operate on different information. Instead of iterating through numbers (“2”, “3”, “4”), they now iterate through category relationships between animals (“canary”, “bird”, “animal”). Only the last two operators are different, in that the model says “yes” when it iterates to the category it is looking for, while it says “no” after a retrieval failure. If the model has to learn the semantic task after first learning the counting task, there will be transfer, because productions learned in the counting task can be reused for the semantic task. Figure 9 illustrates this for the first operator in both models.

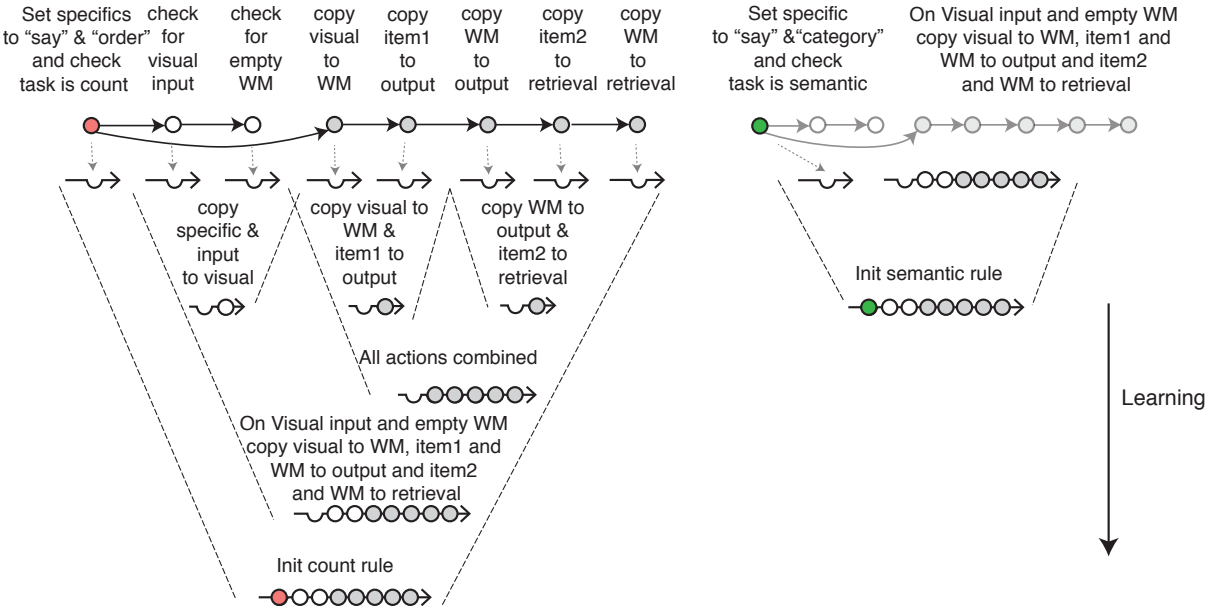

Figure 9. Illustration of transfer between counting and semantic. Left: illustration of learning the Init Count rule. Right: Learning the Init Semantic rule after first learning the Init Count rule. Note that the penultimate rule in the learning of each rule is the same (shown as one blank slot, two whites, and five grays). Therefore, the learning of the Init Count rule shortcuts the process of learning (i.e., positively transfers to) the Init Semantic rule.

When the model has to learn counting from just the PRIMs, it has to go through an elaborate learning process, illustrated on the left side of Figure 9, before it finally learns the “count init” rule. As partial products, it learns several intermediate rules. Because the first operators in counting and in semantic are so similar, the semantics model starts at an

intermediate level (cf. Figure 6), and only needs to do one compilation step to become an expert.

Details of the Simulation

The description of the primitive elements model as outlined in the previous sections does not contain many details on the specific mechanics of the system. The model is implemented in a system called Actransfer, which is an extension and partial modification of the ACT-R architecture (Anderson, 2007), and therefore inherits many of the mechanics of that architecture. Actransfer constrains ACT-R, because it has a fixed set of initial productions (i.e., PRIMs) that the modeler cannot and should not extend. In addition to the PRIMs, a small fixed set of productions coordinates module activity. It makes a few modifications that are necessary for the implementation and are not important on the level of the theory, and it makes some modifications that are detailed below. A full description and downloadable models are available in the supplementary materials. Because Actransfer already has all the PRIMs predefined, a model of a particular task just contains the declarative operators to do the task. Actransfer does not limit the expressive power of ACT-R, because any well-formed production can be expressed as a set of PRIMs.

The workspace and connected modules

The cortical modules and the workspace are inherited directly from ACT-R, with a few modifications. As is already shown in Figure 1, each module connects to a number of slots in the workspace (this is called a buffer in ACT-R).

In the current implementation of Actransfer, perception and motor control have been simplified to an input module and an output module. The input module has a fixed number of slots, allowing it to represent multiple items, or multiple aspects of a particular stimulus. For example, in the count model we used the first slot to represent the starting number, and the second slot the end number. But in later models we will use multiple slots to represent different properties of an object, for example in a model of the Stroop task a slot will be used for the identity of the word, and another slot for the color of the ink. Output consists of three slots, which can be used to represent compound actions, for example to *press* a particular *key*, or to *say* a particular *word*. Furthermore, there are four slots for working memory, four slots for declarative memory, six slots to store the specific items, and four slots for task control. The simplification of perception and output, and the fact that more cognitive modules probably have to be included (e.g., time perception, Taatgen, van Rijn & Anderson, 2007), means that this workspace has to be expanded in the future, along with the necessary PRIMs. However, the workspace should in theory remain fixed, even though its precise composition still has to be established.

Production Compilation

The key mechanism in the model is production compilation: combining two rules into one. As Figure 9 illustrates, multiple combinations are required to build a task-specific rule

from the elementary productions. To ensure a gradual learning process, a new production initially receives a very low utility that only gradually builds up every time the production is reproduced. This ensures that only combinations of rules that recur often end up as rules with a high enough utility to actually be used. This utility learning process is a form of reinforcement learning and follows the standard mechanism that is part of ACT-R.

Declarative Memory

The operators are stored in declarative memory, from which they have to be retrieved to be carried out. The process of finding an operator of which all conditions are satisfied can be very slow, because in the worst case all non-matching operators are retrieved and tested first. The assumption of the model is that the ACT-R mechanism of spreading activation helps in finding the right operator. The idea is that the contents of the workspace spread activation to all items in declarative memory with which it has associations. If associations have the right values, they can help retrieve the right operator at the right time. For the purposes of the models discussed in this article, I have chosen to take a short-cut, and have automatically defined positive associative strengths between elements in the workspace and operators with conditions that match the workspace. This ensures that the model will retrieve the right operator most of the time (not all the time, because there is a noise component in the activation). This short-cut needs to be replaced by something more principled in the future. For example, the way in which operators are learned poses some linear structure on the order in which they are carried out, so a particular operator can spread activation to the most likely next candidate (Cooper & Shallice, 2000). Other than that, the standard ACT-R mechanisms for activation are used (in particular decay that will become important in working memory tasks).

Task Control

The workspace has four slots for task control. In the models in this article, each of these slots has a particular function assigned to it, but the general assumption is that the cognitive system is free to use the control slots in different ways, and that it is indeed possible that individuals differ in the way they handle task control. In the examples we have seen up to now the only task control slot that was implicitly used was one to represent the task itself (CRT, counting, semantic). The model implementation assumes that only operators relevant to the current task are retrieved, and therefore does not require explicit checks, but the current task can be inspected and modified. The three other slots are used to store the current internal control state, the parent goal, and a pointer into declarative memory to maintain the context of the current task. I will discuss each of these in the context of particular models.

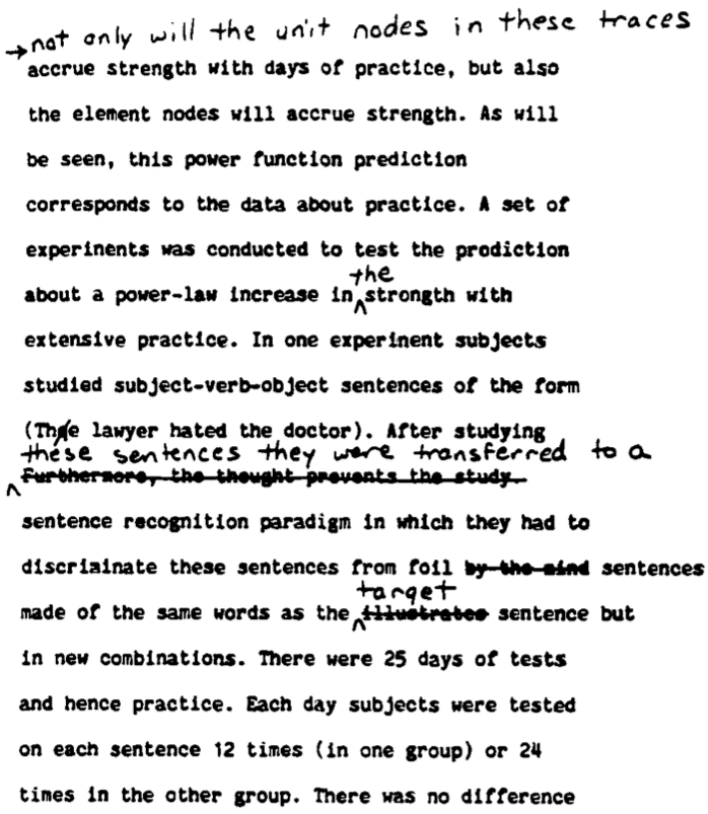

Figure 10. Sample page in the editor experiment: text with to-be-made edits. From Singley & Anderson (1985). Copyright 1985 Academic Press. Reprinted with permission.

Transfer in Text Editing

The Singley and Anderson (1985) experiment was a six-day experiment in which subjects worked with a text editor for three hours per day. Subjects were subdivided into six conditions, a typing control condition, which we will ignore for our purposes here, and five conditions in which subjects used different sequences of text editors. The three editors that were used were ED and EDT, two line-based editors, and Emacs, a screen-based editor. The main difference between the two types of editors is that the screen-based editor is similar to editors that are still commonly used (including Emacs itself), in which part of the text is displayed on a page on the screen. Editing typically involves moving around the cursor on the screen, inserting text by typing, and deleting text by pressing the delete key. The process can be sped up by using particular key combinations for moving, selecting and deleting larger chunks of text. Line editors are an older type of editor in which only a single line of text is shown. Specific commands have to be entered to manipulate the current line of text, or to move to a different line of text. There is no concept of the current position of the cursor, instead particular commands are needed to modify a particular part of a line (for example, in ED the command “s/is/was/p” replaces “is” by “was” in the current line).

In the one-editor condition, subjects used the Emacs text editor on all six days. In the two-editor conditions, subjects used either ED or EDT for four days, and then switched to Emacs on the last two days. In the three-editor conditions, subjects started with one of the

two line editors (ED or EDT), then switch to the other line editor after two days, and ended with two days of Emacs. We will denote the five conditions as follows: Emacs-Emacs-Emacs, ED-ED-Emacs, EDT-EDT-Emacs, ED-EDT-Emacs and EDT-ED-Emacs.

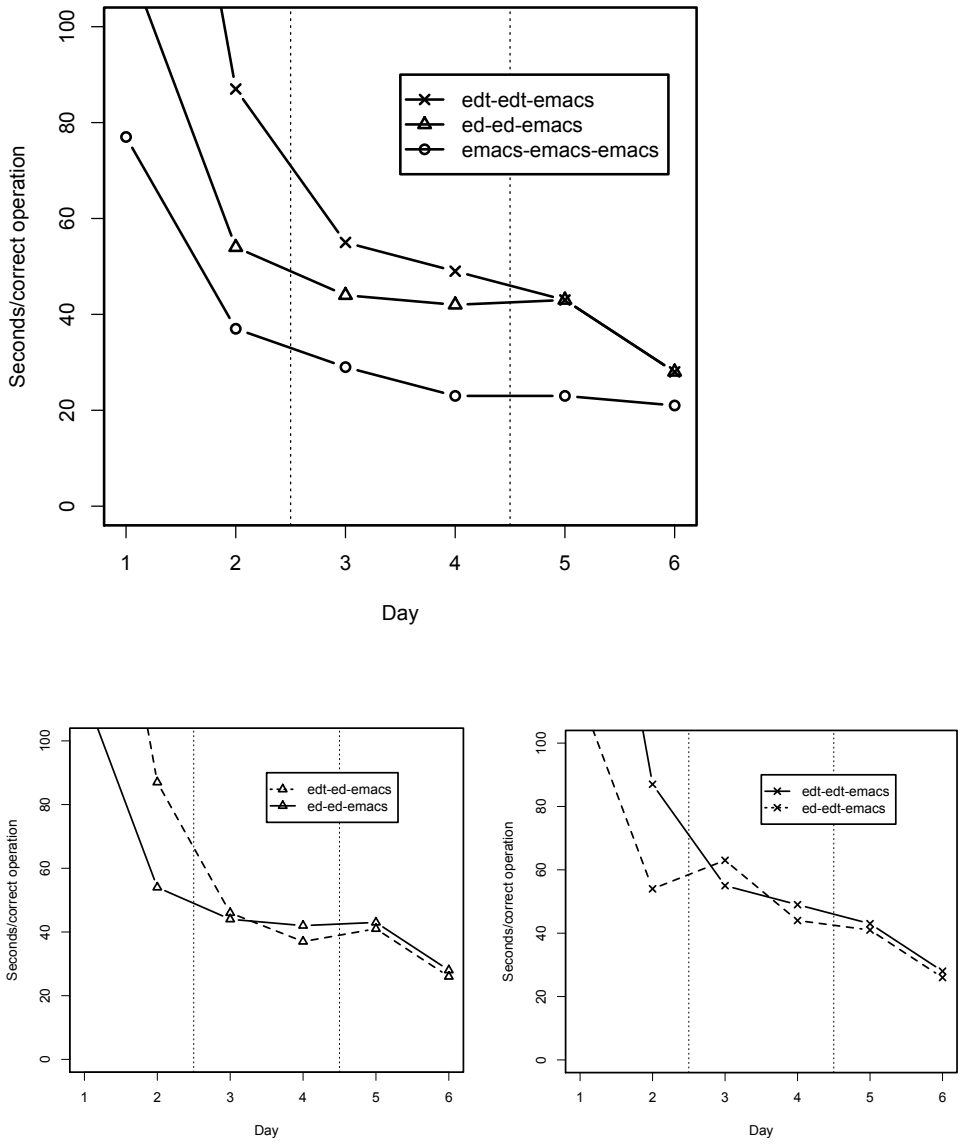

Figure 11. Results of the Singley and Anderson (1985) experiment. The day one scores are in some cases not plotted to not obscure the more subtle effects, and to make it easier to compare the figure to the model results presented later. The dashed lines are three-editor conditions, while the solid lines are two- and one-editor conditions.

Subjects had no prior experience in text editing, so on each day, they received instructions (or a refresher) for the editor that they were going to use that day. The set of instructions was not the complete set for that particular editor, but just enough to be able to perform the editing tasks that they had to do in the experiment. In the editing task itself subjects received printed pages of text (see Figure 10 for an example). Each page consisted of

18 lines, and contained 6 marked edits that the subjects had to perform using the text editor for that day. Edits were either insertions, deletions or replacements, and had to be performed on either a character, word, string or whole line of text.

The overall results of the experiment are shown in Figure 11. A general result is that Emacs was the fastest editor, followed by ED and finally EDT. Subjects became faster at text editing as they gained more experience with a text editor, but there were also clear signs of transfer. One piece of evidence for transfer is that using Emacs on day 1 (in the Emacs-Emacs-Emacs condition) was much slower than any of the other conditions on day 5, where subjects also used Emacs for the first time. Having used the other editors clearly provided an advantage. The second piece of evidence is even stronger: transfer among the line editors. On day 3 of the ED-EDT-Emacs and EDT-ED-Emacs conditions, subjects started using a different line editor. However, in both cases they were much faster at using the new editor than subjects that started with that editor on day 1. The effect of transfer is so strong that switching to a different line editor only produced a small decrement in performance compared to subjects who had used that editor all along, and by day 4 the difference had disappeared all together. Before looking at more detailed analyses of the data, I will first discuss the models of the three editors.

The Primitive Elements Model of Text Editing

The approach in this, and all primitive elements (PRIM) models, is that each task has its own specific model. ED, EDT and Emacs therefore each receive a separate set of task operators. As a consequence, all the root nodes of the operators are specific to a particular editor, but editors may and do share the task-general condition and action PRIMs. Text editing is a complex task, so requires a fairly elaborate model. Fortunately, in this particular experiment subjects received minimal instructions that were just sufficient to carry out the task, therefore largely avoiding the possibility of multiple strategies. Nevertheless, I made a few simplifying assumptions:

1. The assumption that task instructions are memorized perfectly. Even though the instructions were presented and rehearsed quite extensively in the experiment, it is likely that subjects have forgotten or misinterpreted some of them. We dealt with this issue in a different model (Taatgen, Huss, Dickison & Anderson, 2008), and have chosen not to add this to the present model. As we will see, the model therefore initially outperforms the subjects.
2. The perception and action parts of the model are fairly coarse-grained, for example looking up and interpreting the next edit that needs to be made, typing a whole word, and finding the cursor and reading the word directly after the cursor. This means the model does not learn within these actions, and assumes they take a fixed amount of time. This is not a large problem, as Singley and Anderson established

that there was only a small amount of learning in typing, and the majority of the learning was in all other aspects of the task.

3. Only half of the tasks in the experiment were modeled. In the experiment, letters, word, strings and whole lines could be inserted, deleted or replaced. The model can only insert, delete or replace words and whole lines. The added complexity of handling characters and strings would only increase the number of actions and operators, but would not change the way the model operates.

The structure of the models closely follow the task analyses by Card, Moran and Newell (1983) of text editing that were also used by Singley and Anderson (1985) in their analysis of the task. However, adjustments had to be made to stay within the constraints of the ACT-R theory: for example, no explicit subgoaling was used. Instead one of the task control slots, the *control state*, was used to keep track of the state of the model. This state represents the current subtask the model is pursuing, for example looking up the next edit, moving to the right line in the editor, or performing the edit. Furthermore, the model read parts of the next edit to be made at the moment that they were needed instead of memorizing the full edit ahead of time. (i.e., the model will look at the instruction to determine the line to edit. Once it is at the right line it will look at the instruction again to determine what needs to be done.)

The Emacs model's strategy to do a particular edit is relatively straightforward. First, it looks up what the next edit is, and remembers what line the edit is in. It then moves the cursor down until it is at the line in which the edit has to be made. When it is on the right line, there are two options: either it is a word edit or a line edit. If it is a word edit, the model searches for that word and moves the cursor forward word by word until it has found it. Then, depending on the type of edit, it deletes the current word and/or inserts a new word. If it is a line edit, it will delete the line and/or type a new line.

The overall structure of the ED and EDT editors is similar to Emacs, but slightly more complex. In the restricted command set of Emacs there is only one method to move the cursor to the next line, but the line editors each have a unique additional method to skip multiple lines. Line editors require an additional working memory slot to keep a representation of where in the text the current line is, while in Emacs this can be inferred perceptually by inspecting the screen. Because of this, the two line editors are more similar to each other than to Emacs. Let us examine a few examples of where the editors overlap and differ.

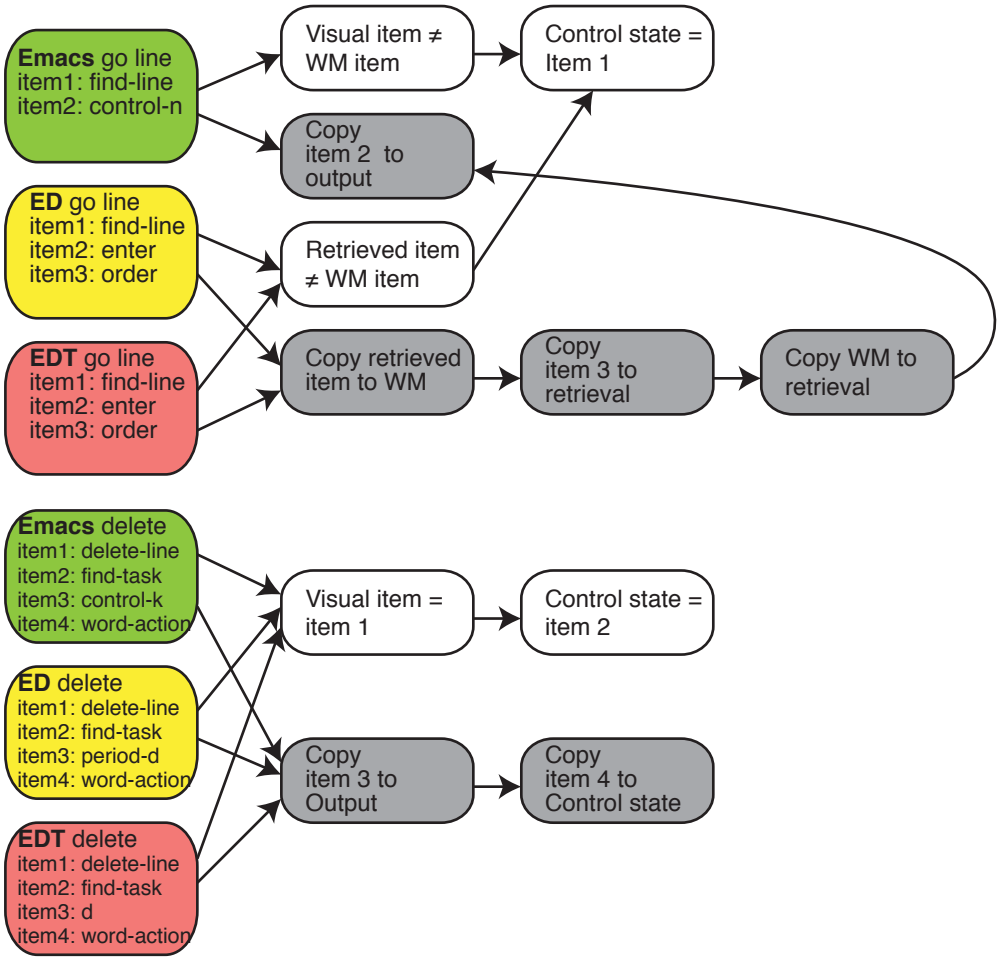

Figure 12. Top: operators to move the cursor (in the case of emacs), or the current line (in the case of ED and EDT) to the line in which the next edit has to be made. Bottom: operators to delete the current line. Note that each of these operators set multiple specifics, they are referred to as "Specific item x" in the relevant conditions and actions.

In order to make the next edit, the right line to edit has to be found first. In Emacs, the page to be edited is on the screen, so moving to the right line involves repeated pressing of control-n. Checking whether the line has been reached is visual: the current location (a visual item) has to be checked against the target (an item in working memory). In both line editors the text is not visible, so the model adapts the strategy of representing both the current line and the target line mentally. The target is represented in working memory, and the current line is updated through an iteration process that is similar to the counting process, that is, by retrieving the next position from declarative memory. In addition, all editors have to check that stage of carrying out an edit is to move to the right line. They keep track of the stage in the control state. In this case, the state has to be "find-line". The second condition makes this check.

This example shows that good transfer is expected between the ED and EDT editors because they can fully share the condition and action PRIMs. The overlap with Emacs is only limited to one PRIM in the condition and one in the action. This overlap is only small, but

still more than nothing, which would be the prediction of the identical productions model. The second example in Figure 12 shows full transfer between all three editors in the case of line deletion. These operators for each of the editors are applicable when the model is looking at the instructions and sees that the edit is a delete-line (a visual check), and the current stage is to find out what the task is (check that the control state is "find-task"). For all three editors the operator is the same. If both conditions are satisfied, a manual output is initiated to carry out a particular keystroke that is different for each editor. In this case the model predicts full transfer, despite the difference in the particular keystroke that has to be carried out. It is not completely clear whether the identical productions model would consider the resulting productions identical.

Even if there is "full transfer" between models, the model for a specific editor still has to learn the task-specific rules. In the delete-line case, the model will learn a production for Emacs that presses *control-k*, one for ED that presses *period-d*, and one for EDT that presses *d*. In the case of full transfer this is only one compilation step as opposed to many (similar to the counting and semantic example in Figure 9).

The full model contains 23, 26, and 19 operators for the ED, EDT, and Emacs model, respectively. To get a sense of the extent of overlap between the three editors, I plotted operator representations for the three editors, except that labels for individual nodes have been removed (Figure 13). The graph layout was obtained through a graph-plotting algorithm by Fruchterman and Reingold (1991), which tries to optimize the layout such that the edges are as short as possible. Colored nodes are again task-specific nodes, and the white and gray nodes task-general nodes. The location of the task-specific nodes in the graph gives a rough sense of overlap between the three tasks. The fact that the central ED and EDT nodes are closer to each other than to the Emacs node is a sign that there is more overlap between those two editors. Closer inspection reveals that many of the colored ED and EDT nodes point to the same lists of grey and white lists of nodes, while the green Emacs nodes have more lists of their own. This should also play out in the amount of transfer the model predicts.

Results of the Model and Comparison to Experimental Data

The model simulation is exactly like the real experiment: the model is given all the operators for all three editors, and then has to perform editing task for six simulated days using the appropriate editor (depending on the condition). At the beginning of each day the activations of the operators for the editor of the current day are refreshed to simulate instructions at the beginning of each day. The underlying ACT-R architecture provides the appropriate execution times for the models, producing predicted solution times. The amount of transfer is mainly affected by the overlap on combinations of PRIMs together with the speed (and amount) of learning. The precise parameters can be found in the supplementary materials. Figure 14 shows the model fit of the experimental data.

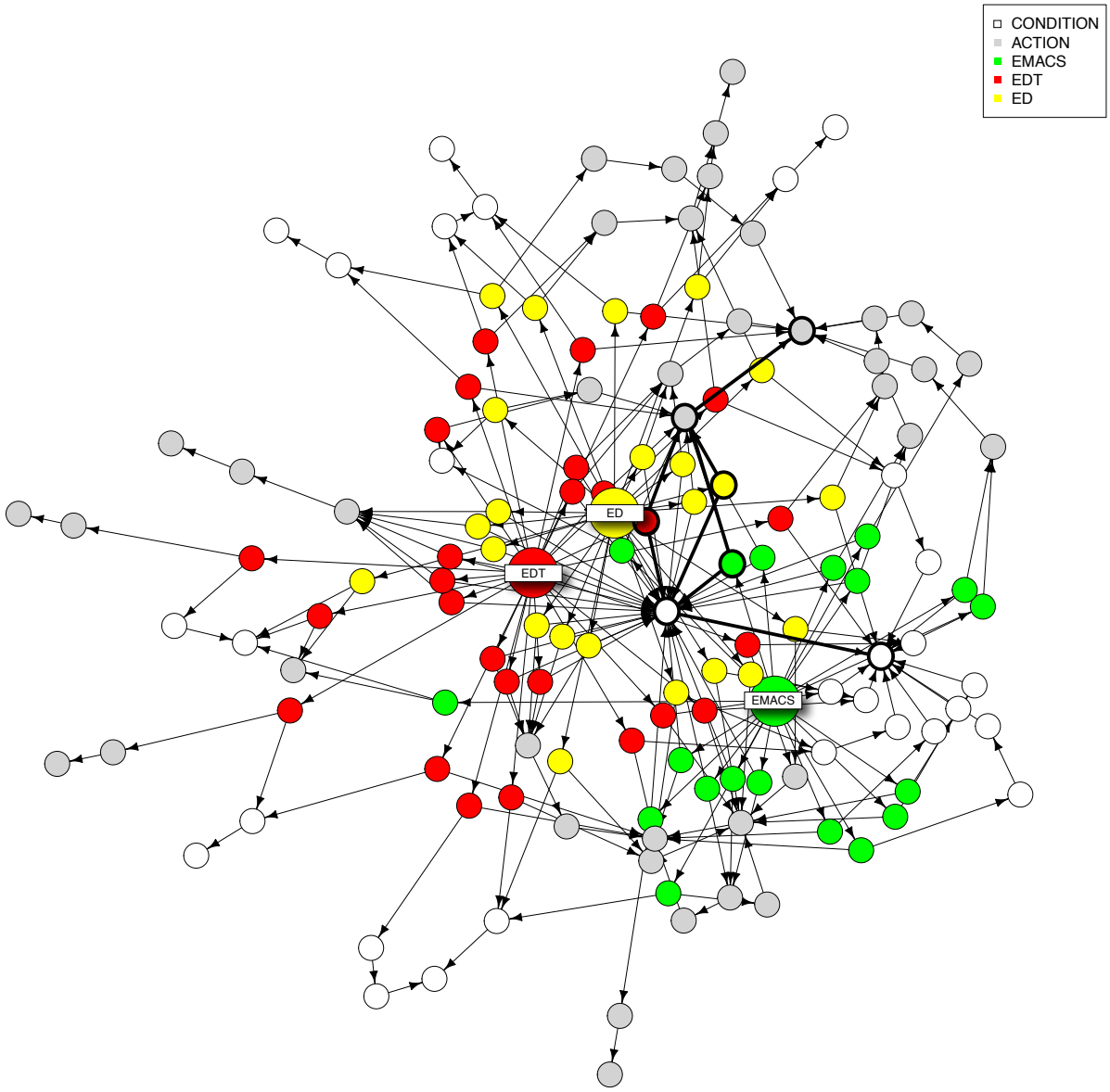

Figure 13. Operators of the editor model. Colored vertices are roots of operators for specific editors, while white vertices are task-independent condition PRIMs, and gray vertices task-independent action PRIMs. The ED and EDT "clouds" of operators (yellow and red nodes) overlap more with each other than with the Emacs cloud (green nodes). The thicker lines represents the seven nodes and their connections from the bottom graph of Figure 12.

A first observation of the model outcome is that it reproduces the characteristic power law of learning. This is most obvious in the Emacs-Emacs-Emacs condition, where the editor remains constant throughout the experiment. The learning effect is mainly caused by the ACT-R mechanisms of production compilation and utility learning, which combine PRIMs into larger productions that speed up processing.

What is more important is that the model exhibits clear evidence of transfer. There is strong transfer between line editors and Emacs, given that on day 5 performance is better for all the two- and three-editor conditions compared to Emacs on day 1. But transfer is even stronger among the line editors. When the model switches from ED to EDT on day 3, its

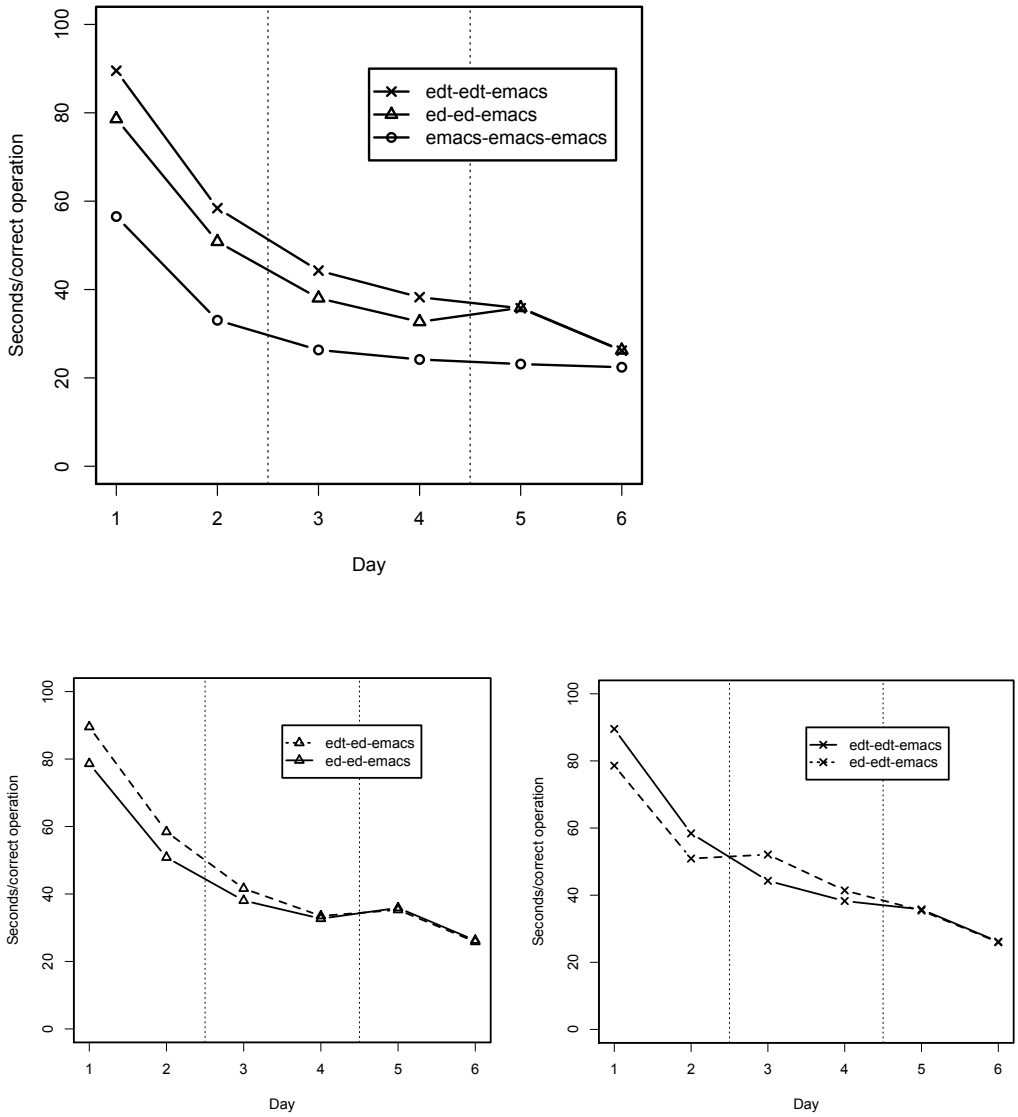

Figure 14. Results of the model of the editor experiment

performance is hardly inferior to the condition that started with EDT, and on day 4 performance is indistinguishable. The same is true for the reverse case when switching from EDT to ED.

We can see that the model exhibits transfer, but we still have to establish whether the amount of transfer is similar to the data, and how this is related to the identical productions model of Singley and Anderson. To assess transfer, Singley and Anderson (1988) used a score developed by Katona (1940). This score compares performance of a subject who transfers to a particular editor on day *n* (after working with another editor on days 1 to *n* - 1) to day 1 performance of subjects who start with that editor. For example, if subjects start working with ED on day 1, they need 115 s per edit. After working with EDT for two days, they only need

46 s per edit when they start using ED on day 3, a difference of 69 s. Subjects that used ED throughout day 1-3, however, need 44 s per edit on day 3, so their improvement is slightly better, 71 s. Expressed as percentage transfer this amounts to (69/71)*100 = 97% transfer. More precisely:

If $M_{lrn}(1)$ is performance on that editor on day 1, $M_{trans}(n)$ is performance on day n after using a different editor on day 1 to $n-1$, $M_{lrn}(n)$ is performance on day n after using the same editor on day 1 to $n-1$, then percentage of transfer is given by the following equation:

$$\%\text{transfer} = \frac{M_{\text{lrn}}(1) - M_{\text{trans}}(n)}{M_{\text{lrn}}(1) - M_{\text{lrn}}(n)} * 100$$

In order to factor out the effect of subjects becoming better typists (subjects were students from a secretarial school, so were already proficient typists), and also to pinpoint more clearly at what stages of an edit transfer was most pronounced, Singley and Anderson performed a keystroke analysis that allowed them to isolate two stages of an edit: to move to the line in which the edit had to be made (including looking up the next edit in the manuscript), and making the edit within the line. Each of these two stages was further subdivided into a planning component and an execution component. The execution component was defined as the time between the first and last keystroke minus any interkeystroke pauses of more than 2 s. The planning component was defined as the remainder of the time spent in that stage. They found that most of the learning took place in the respective planning phases of the edit, and focussed their transfer analysis on those. Given that the model does not improve at all at typing, this also presents the best opportunity for a comparison. Table 1 shows this comparison, with the percentage transfer found in the experiment, the transfer accounted for by Singley and Anderson, and transfer accounted for by the model.

Overall, transfer between the line editors (ED and EDT), was higher in editing a line than moving to a line, probably because moving multiple lines involved quite different steps for each of these editors. The editing of the line itself showed almost perfect transfer, which can be attributed to the fact that the two editors were indeed identical in that respect, except for the particular keystrokes that needed to be carried out. Transfer from line editors to Emacs was much lower, but still considerable, with no difference between the two components. As the table shows, the model predicts more transfer than identical productions, and therefore provides a better fit of the experimental data. In particular, identical productions poorly fits the transfer to Emacs.

There is another crucial difference between the models: the identical productions model looks at the number of productions that two editors share as proportion of the total number of productions, which determines the transfer proportion. The PRIM model simulates the process of learning, so transfer can be determined by comparing simulated response times to

Table 1. Transfer between text editors in the Singley and Anderson experiment, the identical productions model, and the PRIM model.

Component	Training Editor	Transfer Editor	Human data	Identical Productions	PRIM Model
Planning to move to a line	ED	EDT (on day 3)	87%	68%	81%
	EDT	ED (on day 3)	91%	75%	90%
	ED/EDT	Emacs (on day 5)	61%	39%	66%
Planning to edit a line	ED	EDT (on day 3)	105%	90%	90%
	EDT	ED (on day 3)	99%	85%	92%
	ED/EDT	Emacs (on day 5)	62%	27%	57%

response times from the experiment, which is a much more direct comparison. In the PRIM model, particular editors never share task-specific rules, but do use each others task-general rules. This means that transfer is possible in cases where the general pattern is the same but the specifics are different. For example, a different keystroke for a particular function only means that a constant in the respective root PRIMs needs to be different (e.g., for delete-line: period-d in ED, d in EDT, and control-k in Emacs). Therefore, mastery of any editor enables learning the delete-line production for a new editor in just one step. This is especially important in the case of transfer to Emacs, where few productions are identical to line editor productions, but many share components (Figure 14). But transfer goes beyond particular actions. Each time multiple arrows point to one of the white or gray nodes in Figure 14, there is potential for transfer. If a particular operator shares a condition with another operator, it benefits from transfer even if the action is different, and vice versa.

The learning model has another advantage: it can predict transfer with different amounts of training. The Anderson and Singley experiment has a number of potential (but probably not critical) confounds. First, the transfer between line editors is assessed on day 3, while the transfer between the line editors and Emacs is assessed on day 5. It is possible that with more training, the learned skill becomes more specialized, decreasing its potential for transfer (Anderson & Fincham, 1994, Frensch & Geary, 1993). Another problem is that subjects receive different numbers of repetitions between conditions, because sessions last exactly three hours, and editors require different amounts of time per edit. So, if we compare subjects' performance on Emacs on day 5 after four days of EDT, they have had much fewer repetitions than subjects who used Emacs all along. The model allows us to experiment with different amounts of training. To test this, the model was run with different amounts of blocks of practice, each block consisting of 240 edits. The model was run for one through four blocks of practice on ED, after which is switched to one of the other editors (Emacs of EDT).

Learning was switched off in the transfer blocks to make the comparison as precise as possible. Transfer performance was compared to runs in which the model just performed Emacs or EDT for several blocks, again with learning switched off in the last block. The amount of transfer was calculated using the Katona equation, and is plotted in Figure 15. Consistent with earlier research, the model predicts that transfer will decrease as the amount of training increases. The model's explanation for this is that it gradually transitions from learning new general combinations of PRIMs that do produce transfer, to task-specific productions that do not. The simulation also confirms the conclusion that there is more transfer between line editors than between line editors and Emacs, although the difference may not be as large as the data in Table 1 suggest.

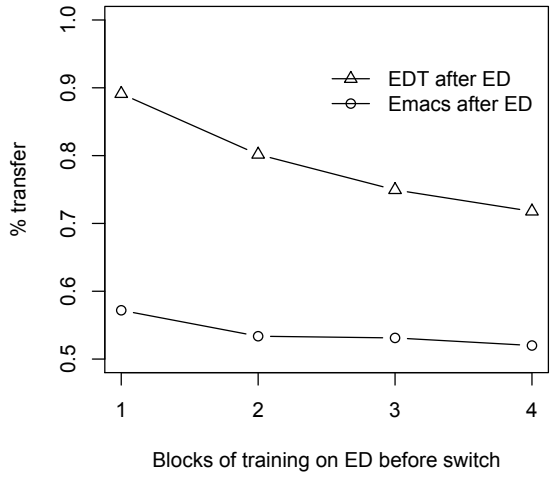

Figure 15. Amount of transfer from ED to either EDT or Emacs after training ED for 1 to 4 days.

To summarize: the PRIM model provides a superior fit over the identical productions model, but also has a major conceptual advantage. It simulates the full learning process, and provides concrete predictions of the time needed to make each edit. It resolves the problem of determining the level of generality of individual rules beyond the specific task context, an issue that the identical productions model sidesteps. One thing that we have not developed in much detail yet is task control. In the next model we will develop the notion of switching goals, and in yet later models we will introduce working memory control.

Transfer in Arithmetic: Setting Subgoals

The general workspace of the PRIM model has four slots reserved for task control. One of these slots is used to identify the current task, and makes sure only relevant operators are retrieved. As we have seen in the editor model, a control state, a second slot, can be used to subdivide a task into several stages. Sometimes one or more of these stages are meaningful tasks in themselves. For example, multi-column addition is normally a separate task, but it is

also a subtask in multi-column multiplication. It is therefore unlikely that multi-column addition as part of multiplication needs its own set of operators that have as additional check whether it is in the "addition" stage of the multiplication. A different option is to temporarily change the task to multi-column addition, and change it back after the addition has completed, a process normally called subgoaling or subtasking.

A strategy to achieve a limited form of subtasking is to change the current-task slot to the subtask, and the restore the main goal after the subtask finalizes. For this, it is necessary to store the main task for the duration of the subtask. This type of subtasking has long been considered as an architectural mechanism (the goal stack), but the current consensus is that it is a cognitive strategy (Altman & Trafton, 2002; Anderson & Douglass, 2001). The PRIM model treats it as such as well, because the strategy is carried out by a particular combination of elementary productions that are not particular to that purpose. Subtasking enables a more direct type of transfer if particular separable components in a larger task are truly identical to a separable component in another task.

An experiment by Elio (1986) provides data to test these ideas. Elio taught subjects to carry out a series of calculations. Each of the steps in the calculation either used input that had to be read from the screen, or results from earlier steps that had to be recalled from memory. Figure 16 shows the interface to the experiment, and Table 2 an example of one of the initially learned instructions (Procedure A), and examples of instructions in the various transfer conditions (Procedure B-D).

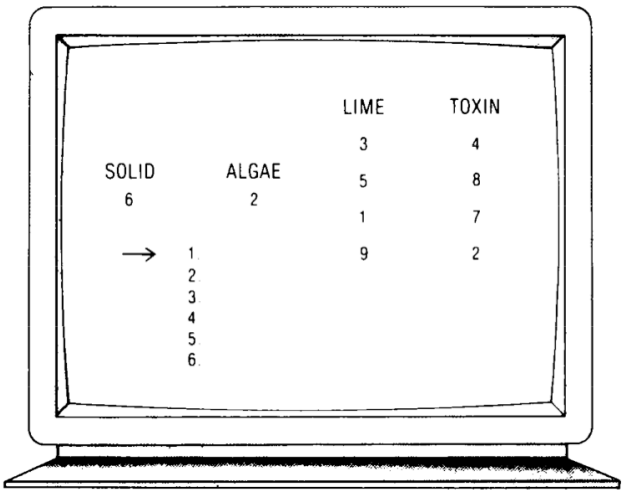

Figure 16. Sample screen in the Elio experiment. From Elio (1986), Copyright 1986 Cognitive Science Society. Reprinted with permission.

In the experiment, subjects had to calculate a hypothetical pollution rate based on a sample of water. For this, they had to memorize the initial procedure A, and were drilled on it until they were perfect at recall. They were then trained on performing the procedure. They were shown a screen like in Figure 16, and had to carry out the memorized calculations and

enter the results one line at a time. Some of the calculations, like the first two in Procedure A, are so-called *component* steps. In these steps the input for the calculations could be read from the screen (with, for example, $lime_2$ referring to the second item in the lime column). Other steps, like the third step in Procedure A, were *integrative* steps in which the arguments of the calculation referred back to the earlier lines. Because the answers disappeared from the screen, subjects had to memorize the results of those steps that were needed later. For example, after carrying out the third step in Procedure A, the Particulate rating could be forgotten, but the Mineral rating still needed to be retained, along with the newly calculated Index 1.

Table 2. Procedures in the Elio (1986) experiment

Procedure A: Initially learned procedure	
Particulate Rating	solid x ($lime_4$ - $lime_2$)
Mineral Rating	greater of (algea/2) (solid/3)
Index 1	Particulate + Mineral
Marine Hazard	($toxin_{max}$ + $toxin_{min}$)/2
Index 2	Index 1/Marine
Overall Quality	Index 2 – Mineral
Procedure B: Transferred component condition	
Particulate Rating	($toxin_{max}$ + $toxin_{min}$)/2
Mineral Rating	solid x ($lime_4$ - $lime_2$)
Marine Hazard	greater of (algea/2) (solid/3)
Index 1	Mineral/Marine
Index 2	Particulate x Index 1
Overall Quality	Index 2 + Index 1
Procedure C: Transferred integrative condition	
Particulate Rating	($lime_{min}$ x 3) + algae
Mineral Rating	lesser of (solid + $lime_1$) (algae + $toxin_3$)
Index 1	Particulate + Mineral
Marine Hazard	solid/$lime_1$
Index 2	Index 1/Marine
Overall Quality	Index 2 – Mineral
Procedure D: Control condition	
Particulate Rating	($lime_{min}$ x 3) + algae
Mineral Rating	lesser of (solid + $lime_1$) (algae + $toxin_3$)
Marine Hazard	solid/$lime_1$
Index 1	Mineral/Marine
Index 2	Particulate x Index 1
Overall Quality	Index 2 + Index 1

Subjects had to solve 50 problems in the training condition, and were then transferred to one of three conditions, illustrated in Table 2. In the transferred component condition (procedure B), the component steps remained the same as in training, but the way they were integrated changed. In the transferred integrative condition, on the other hand, the component calculations changed, but the way they were integrated remained the same. Finally, in the control condition both aspects were different. Subjects also solved 50 problems in the transfer condition.

Elio (1986) reports the results of the experiment by giving the power law parameters of the training phase, and the solution times in the first and second set of 25 problems in the transfer phase, separated in solution times for component steps and integrative steps. From this we can reconstruct the results shown in Figure 17.

The results show the two types of transfer. We see that if a step is identical in the two procedures, it is carried out faster than when the step is different. But there is also transfer from the initial procedure to the transfer procedure in all conditions, even in the control condition in which all calculations are different. The model therefore has to capture both aspects of transfer: the general transfer effect and the specific transfer effect. For the general transfer effect, it will rely on the reuse of general productions, as the previous models, but for the specific transfer it will rely on subtasking.

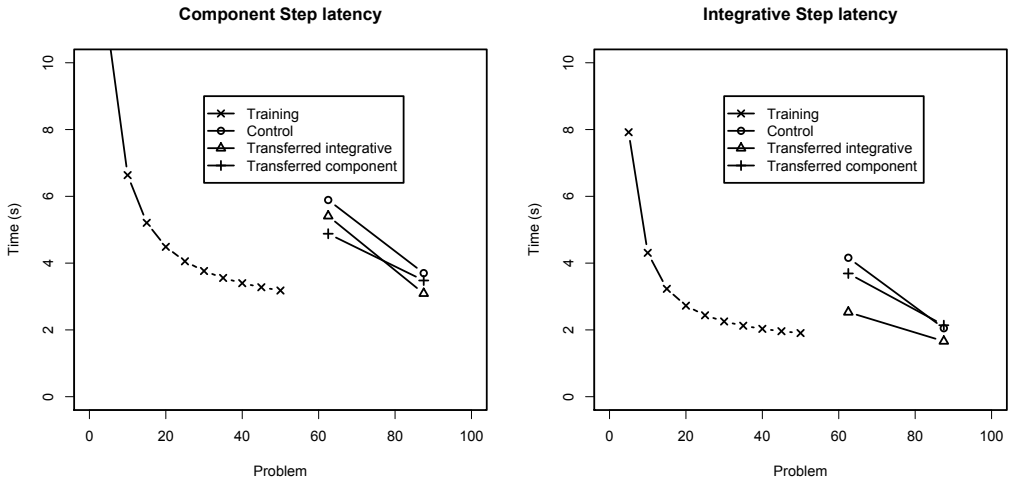

Figure 17. Mean solution times for component and integrative steps in the Elio experiment. Solution times from problem 1 to 50 have been calculated on the basis of the power law parameters from Elio (1986), while the data points for trials 51 to 100 are from her reported data.

The implementation of the model for this experiment poses two challenges that we have not dealt with before. The first is to allow the model to handle the division of the task into subtasks. What we will do is to make each of the lines in the calculation a separate task. Particular procedures then only have to specify the order in which the subtasks have to be carried out. The second challenge is working memory management: memorizing the

appropriate partial result while dropping the ones no longer needed. The Elio model deals with each of the these, but here I will only explain the issue of subtasking, and leave the discussion of working memory for the specific working memory tasks later in the article.

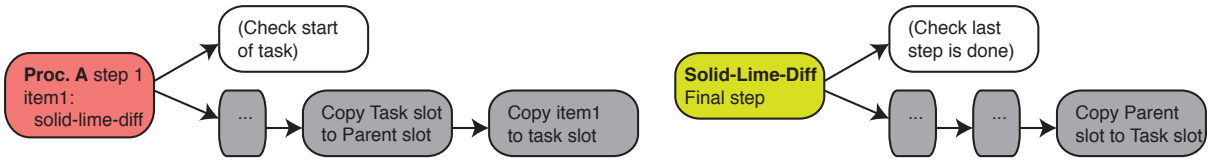

Figure 18. Two operators in the Elio model: one that sets the subtask of calculating "solid-lime-diff", and the a second that moves control back to the main task. Operators have different colors to represent that they are considered as different tasks, even though one is a subtask of the other. Only the condition and action PRIMs that are important for the subtasking are shown.

The solution for subtasking is very simple, and is illustrated by two example operators in Figure 18. At the start of a procedure, for example Procedure A, an initial working memory step is carried out, after which the current task (which is in the Task slot) is copied to one of the other control slots, which is named *Parent slot* in this Figure for clarity. After copying the current task, the task name in the Task slot is replaced with "solid-lime-diff". As a consequence, the model will now start carrying out operators for the Solid-lime-diff subtask. At the end of that task, the contents of the Parent slot are copied back into Task slot, after which the model will resume carrying out operators for Procedure A.

The complete model consists of operators that implement each of the four procedures (A-D), and operators for each of the subtasks (12 in total). There is considerable overlap in PRIMs among the procedures and subtasks, which will produce the general transfer. Moreover, the procedures also use each other's subtasks, which produces the specific transfer.

It is important to note that the choices we have made here for implementing task control are not fixed. A different set of strategies with a different representation may produce a similar result. The way task control is handled in this model is very light, and not sufficient for deeper goal structures. The choices made here are sensible considering earlier empirical work related to ACT-R (Altmann & Trafton, 2001; Anderson & Douglass, 2001), but it is important to stress that task control is not an architectural mechanism, but based on strategies, which are by their nature susceptible to individual differences. Indeed, the assumption that processes of cognitive control are based on skills is the key to explaining transfer in cognitive control.

With these additional aspects implemented, it is now fairly straightforward to model the Elio data. The full model can be found in the supplementary materials. The results are in Figure 19. The models show the same main effects as the experimental outcomes: a transfer effect in all conditions, but more in particular in the steps that are identical between the two procedures. The general transfer is due to learned general productions that handle

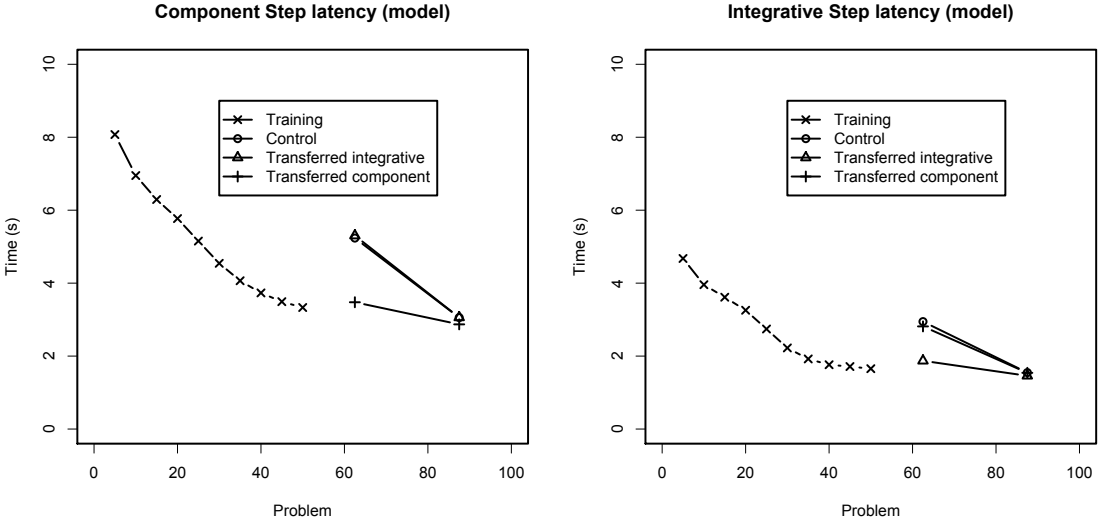

Figure 19. Model fit of the Elio (1986) data.

calculations and overall task control, while specific transfer in the identical steps can be attributed to the fact that the task-specific productions in subtasks can be reused immediately.

Far Transfer in Cognitive Control

The theory of transfer presented here predicts that transfer is possible between tasks that are quite different in content, but that share the same underlying structure. The two examples we have examined in the previous sections are both cases in which the tasks involved are very similar. A number of recent studies have examined far transfer (Barnett & Ceci, 2002) in the context of cognitive control and general intelligence. A remarkable but also controversial study by Jaeggi et al. (2008) showed that training on a complex N-back task (the double N-back task) not only strongly improved performance on N-back itself, but also on scores of fluid intelligence (the Raven progressive matrices task and variants of it). Mackey et al. (2011) trained children on games that either stressed reasoning skills or cognitive speed. Children in the reasoning group significantly improved their fluid intelligence skills, as measured by a test similar to the Raven test, while children in the cognitive speed test improved on the Coding B task from the Wechsler Intelligence Scale for Children IV (in the Coding task, subjects have to translate digits into symbols by identifying the corresponding symbol for a digit provided in a legend).

Unfortunately, these two studies are not straightforward to model. Both studies feature a test of general intelligence, which requires an extensive modeling effort to capture (even though Carpenter, Just & Shell, 1990, modeled the Raven test in a much less constrained modeling environment). Moreover, in the Mackey et al. experiment it was not exactly clear which of the games was effective in producing the improvement. Fortunately, two other successful studies of far transfer did involve tasks of cognitive control (working memory

capacity, Stroop, task switching) that are comparatively easy to model, so I will focus the discussion on them.

Training cognitive control and general intelligence is sometimes compared to training muscles: by regularly exercising them, they become stronger. In the context of working memory capacity this is a particularly attractive idea: by training working memory its capacity increases. Here, I would like to introduce a different idea: cognitive training helps developing our representations and strategies for strong cognitive control, and improves the arsenal of skills available to deal with different control situations. This training aspect is true in sports as well: to be able to excel in for example the javelin, it is not sufficient to have enough muscle power, but also the technical skill to throw it appropriately.

In the two studies that I will model here, the key element of transfer is based on the choice on how tasks are executed: proactive or reactive. This distinction was introduced by Braver, Gray and Burgess (2007). In, for example, the Stroop task, a reactive strategy is to wait for the stimulus, process the stimulus, and then focus on the color of the ink. A proactive strategy is to already prepare for the upcoming stimulus and focus on the color of the ink right away, reducing interference. Proactive control requires planning before a particular event, while reactive control waits for the event before making a decision. This means that in reactive control the outside world controls behavior, while in proactive control behavior is a function of internal preparation and external input. As we will see, proactive control leads to better performance, but also requires more complex strategies. The choice between the easier reactive and harder proactive strategy is likely to be influenced by how hard it is to learn the proactive strategy. If people are trained on proactive strategies, they may therefore tend to favor proactive strategies in new tasks, because they have learned or strengthened task-general knowledge that supports proactive strategies.

Improving Working Memory Capacity with Transfer to Stroop

Chein and Morrison (2010) performed a study in which they trained participants on two complex working memory tasks. It was one of several studies aiming at the improvement of working memory capacity (Klingberg, Forssberg & Westerberg, 2002; Klingberg et al., 2005, see Morrison & Chein, 2011, for an overview). The tasks were originally designed by Kane et al. (2004), who found that complex working memory tasks, as opposed to simple working memory tasks, provide stronger correlations with each other and other measures of cognitive control. Chein and Morrison used the so-called Verbal-CWM (complex working memory) and Spatial-CWM tasks. In the Verbal-CWM task, subjects were presented with a sequence of letters that they had to memorize, with a starting length of four letters. In between each of the letters they had to perform a lexical decision task for four seconds, making it harder to memorize and rehearse the letters. At the end of the sequence, subjects had to recall the whole sequence. If subjects were correct for two sequences in a row, the length of the sequence was increased by one. If they were incorrect for two sequences, the length of the

sequence was decreased by one. On each of the 20 sessions of the experiment, subjects performed 16 trials. The Spatial-CWM task was identical in structure, but spatial positions in a 4x4 grid had to be remembered instead of letters. Similarly, instead of lexical decision, subjects had to give symmetry judgments of presented abstract pictures. In the model of this experiment, only the Verbal-CWM task was modeled. The results from the Spatial-CWM task were, however, very similar to the result of the Verbal-CWM.

Performance on the Stroop task and a reading comprehension task (which I did not model) showed significant improvement between pre- and posttest compared to a control condition in which working memory was not trained. This indicates far transfer. Apart from that, subjects also improved considerably on the working memory tasks themselves. Figure 20 shows the results of training on the Verbal-CWM task, and the pre/posttest comparison for the Stroop task.

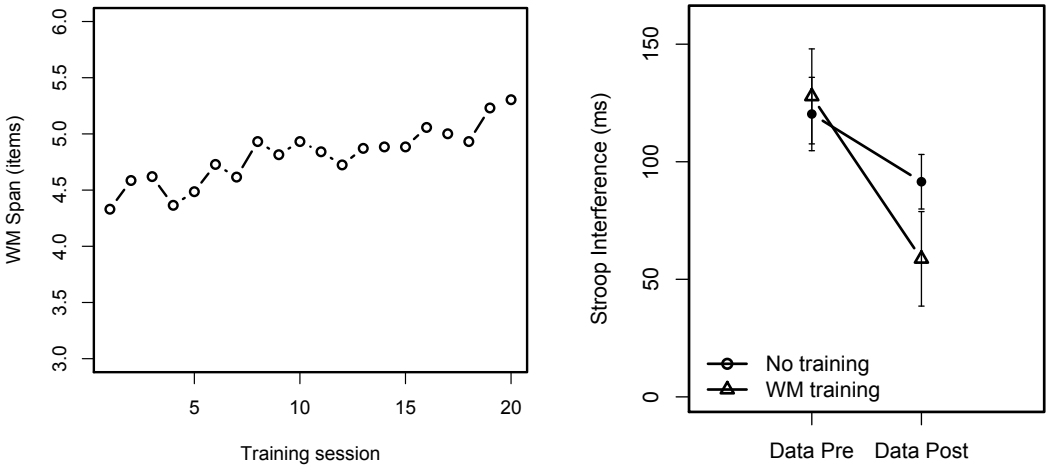

Figure 20. (a) Improvement in the Verbal-CWM task over the 20 training sessions. (b) Reduction in Stroop interference due to WM training compared to a control condition without training.

Modeling Proactive (and Reactive) Control in the Verbal-CWM Task

In the Verbal-CWM task subjects have to juggle two tasks: on the one hand the lexical decision task, and on the other hand memorizing the list of letters. To be successful in remembering long lists of letters, some form of rehearsal is likely to be necessary. From the viewpoint of control this presents two separate issues. The first is how to build a memory representation of the list, because it is not possible to store all items in the workspace. The second is how to maintain this memory representation and protect it from decay. This second issue involves proactive control, because cognitive actions have to be taken to ensure successful future retrieval. It is also the component that will transfer to the Stroop task and improve proactive control in that task. I will explain both in a bit more detail.

Our earlier studies showed that is most likely that only one item can be stored in the workspace (Borst, Taatgen & van Rijn, 2010; Borst, Taatgen, Stocco & van Rijn, 2010; Borst,

Taatgen & van Rijn, 2011). With an immediate capacity of only one item, any additional items have to be kept in declarative memory. The assumption of the model is that, as with task control, there is no fixed method for organizing "extended" working memory, but that this is part of the task strategy. As soon as someone uses a particular working memory strategy, this can be transferred to other tasks if the requirements of those tasks are similar enough. For the verbal-CWM model, I specified a relatively simple strategy by building a list in memory that works as follows:

The first item that needs to be remember is just placed in working memory. When a second or subsequent item has to be remembered, the current contents of working memory is moved to declarative memory, and are replaced with the new item. Moreover, in a second slot a link to the just-removed item is placed. In addition, one of the task control slots keeps a link to the first item (Figure 21).

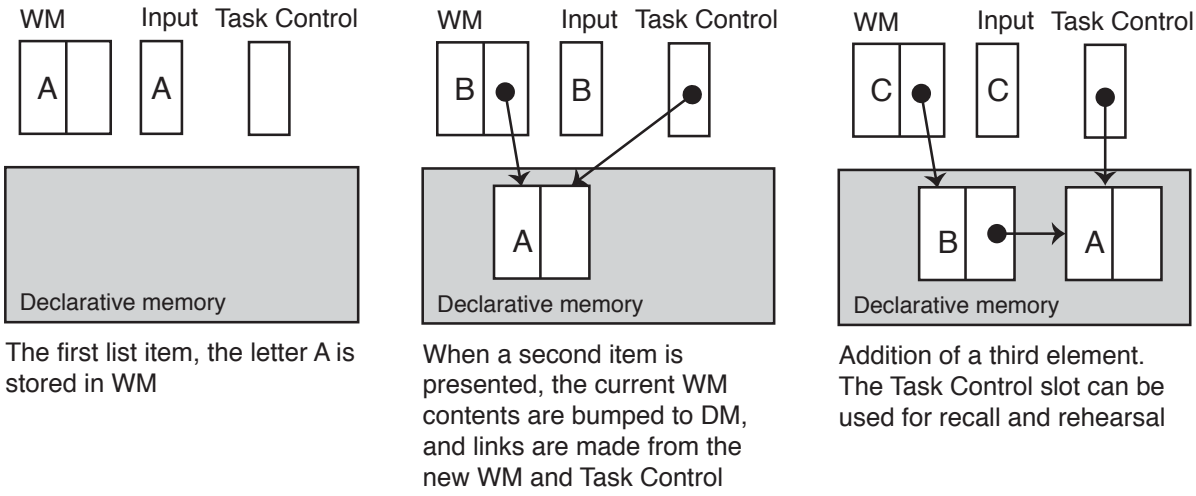

Figure 21. Working Memory Control in the Verbal-CWM task

It is important to stress that this method of remembering items is just a strategy, and not an architectural mechanism. It is accomplished by the same PRIMs that we have already discussed, with the only addition that when something is placed in WM there is an option to create a new WM element (bumping the old into declarative memory), or just modifying the existing element. This method can, however, build other structures in memory (trees, directed graphs, etc.), each of which would require a different combination of PRIMs (see Tenenbaum, Kemp, Griffiths & Goodman, 2011, for an account on how people build different knowledge structures).

A critical property of elements that are bumped into declarative memory is that they are susceptible to decay, and therefore need to be rehearsed. Rehearsal is not a complex strategy in itself, and only needs a simple operator that consists of a few PRIMs. The complexity of rehearsal in the verbal-CWM task is that it has to be initiated at strategic moments during the

task. Rehearsal is, after all, a proactive strategy, because it involves cognitive processes in the service of future demands.

In order to explore the need for internal control, I constructed two models of the Verbal-CWM task, a proactive and a reactive model. The reactive model (9 operators) does not use rehearsal, and therefore also does not need any internal control state, because the external input determines exactly what needs to be done. If a word is presented, a word/non-word decision is asked for, if a single letter is presented, it has to be memorized, and if a recall prompt is given, the sequence of memorized letters has to be reproduced. Whenever there is nothing to be done at a particular moment, the model just waits for the next input.

The proactive model is more complex (14 operators), because it has to integrate rehearsal into the procedure. It uses one of the task control slots that stores the control state to keep track of what it is currently doing: lexical decision, rehearsal or report. I will refer to this as the control state of the model (see Taatgen, 2005, 2007 for a discussion on control states). The model determines the next action on the basis of the control state and the input. The internal state is necessary, because during rehearsal outside stimuli have to be ignored temporarily until rehearsal can be suspended. Also, rehearsal and reporting use the same retrieval strategy, so they have to be discriminated.

In terms of PRIMs, the key characteristic is that the proactive model employs operators that use a combination of checking the input against a particular value, and checking the control state against a particular value, followed by an action in which the control state is changed to a new value (and possibly some more actions). For example, if the control state is to rehearse, and input is still the letter we just added, the model should continue rehearsing, but if a new word appears it should switch the control state to lexical decision. The reactive model, on the other hand, does not use the control state at all, but bases its action only on the current input.

If we compare performance of the two models using the same 20 session training regimen as in the experiment, we see that, not surprisingly, the proactive model captures the improvement in working memory span that was found in the data, while the reactive model does not (Figure 22, left panel). This is a clear indication that subjects, or at least most of the subjects, use a proactive control strategy in the verbal-CWM task.

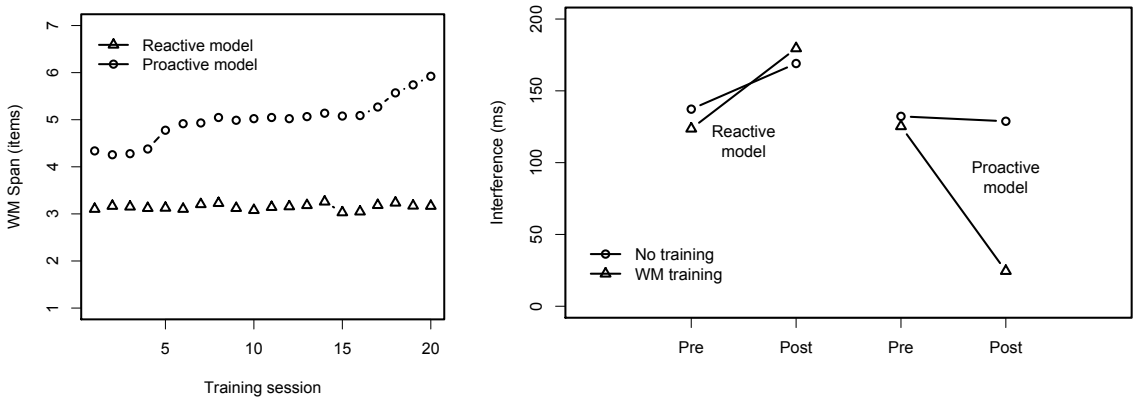

Figure 22. Results of the model of the Chein and Morrison experiment. Improvement in working memory capacity over 20 training sessions (left), and Stroop interference (right) for both the proactive and the reactive model.

Proactive and Reactive Control in the Stroop Task

If the particular combination of PRIMs for proactive control is important to improve on the verbal-CWM, it is likely to also provide the explanation for improvement on the Stroop task. The model for the Stroop task builds upon a model by van Maanen, van Rijn and Taatgen (2012) which accumulates evidence from different sources to make a decision. The approach to model differences in Stroop performance is based on a model by De Pisapia and Braver (2006) where proactive control helps suppressing irrelevant attributes. The Stroop model implements both a proactive and a reactive strategy. The reactive strategy waits for the stimulus, then processes both the identity of the word and the color of the ink, and then retrieves the word corresponding to the color of the ink from memory. In the case of conflicting information, for example the word red in blue letters, both the blue and the red concept receive evidence, which slows down the decision compared to congruent trials where both sources of evidence point in the same direction. The proactive strategy tries to reduce the impact of the irrelevant parts of the stimulus by preparing for the upcoming trial: it focusses on the color of the ink, diminishing the impact of the word identity, thereby reducing interference. In order to achieve this, the proactive strategy alternates between two control states: one in which the model prepares for the upcoming stimulus, and one "neutral" state. Two operators are important concerning this state: one that switches the state from "neutral" to "preparing" while waiting. The other direct visual attention to the color of the ink if a stimulus appears, and the control state has been set to "preparing". In other words: the first operator sets up proactive control, and the second executes it once the stimulus appears. Both of these operators have to compete with reactive operators that just wait during the fixation cross, and just perceives all aspects of the stimulus as soon as it appears. The proactive operators use the same combination of PRIMs as the operator in the verbal-CWM model that initiates the rehearsal. As a consequence, training the verbal-CWM task helps learning and

strengthening the task-general productions that support a proactive strategy. The reactive control strategy does not need a control state at all, and is therefore much more simple.

Because the Stroop model implements both strategies, the model tries each one equally often in the pretest. The pretest by itself is not enough to settle on a strategy, and therefore there is no change between pretest and posttest in the control condition (Figure 22, right panel). If the model is then trained with the proactive strategy for the verbal-CWM task, the task-general productions for the proactive strategy are learned and strengthened by the training task. It will therefore also favor the proactive strategy for the Stroop task, resulting in a reduction of interference. When the model is trained on the verbal-CWM with a reactive strategy, reactive strategies are strengthened, which leads to an increase in Stroop interference. We will see an example of such an increase in the data in the next experiment, where control subjects are trained on a reactive task.

Improving Task Switching with Transfer to Working Memory Capacity and Stroop

A second example of how training proactive control can be transferred to other tasks is a study by Karbach and Kray (2009). Karbach and Kray investigated whether training on a particular task-switching task transferred to different task-switching tasks (near transfer), and also to other tasks of cognitive control and general intelligence (far transfer). More specifically, they looked at age differences in transfer, but because they found transfer in all age groups we will collapse over their results for the purposes of the simulation (the age groups were: 8.1-10.1 years, 18.0-26.3 years and 62.3-76.8 years). In the training task subjects had to switch tasks on every second trial. They received no external cues for the switch, so they had to keep track of this themselves. A block consisted of 17 trials. In each trial, a picture of either one or two cars or planes was presented. The two tasks that the subjects had to alternate between were to respond to whether the picture was a plane or a car, and whether there were one or two items on the card. The same two keys were used for both tasks, and the inter-stimulus interval was 1400 ms. Besides blocks in which subjects had to switch between tasks every two trials, they also had to perform single-task blocks in which all trials required either the car/plane or one/two response. On each of the four days of training they performed eight single-task and twelve task-switching blocks. Apart from this experimental condition there was a control condition in which subjects only performed single-task blocks, and two additional experimental conditions with variations on the experimental condition I will not discuss those, since they produced the same amount of transfer as the standard experimental condition.

Training was preceded by a battery of pretests that was repeated after training in the posttest. This battery consisted of several tasks, of which we will model three in particular: another task-switching task, the Stroop task, and a working memory task. The task-switching task was identical to the training task, except that subjects had to respond to pictures with vegetables or fruit on them that were either small or large. The two tasks were to indicate

whether the picture was a fruit or a vegetable, and to indicate whether the picture was small or large. The Stroop task was similar to the one in the Chein and Morrison (2010) version, except that conflict trials were contrasted with neutral trials instead of consistent trials. The working memory task, also taken out of the collection of tasks from Kane et al. (2004), was a complex working memory task called the count-span task. In it, subjects were presented with a series of pictures with green circles, blue squares and blue circles (approximately 13 objects/picture). Subjects had count the number of blue circles (between 3 and 9), and say the total, after which a next picture would be shown after a blank screen of 500 ms. At the end of a series of pictures, the number of which ranged from 2 to 6, the list of all totals had to be recalled. The measure of performance was the proportion of correctly recalled numbers, so if the subject was only able to recall 3 out of 6 digits this would still be counted as a 50% score.

The results show transfer on all three tasks in the experimental condition: subjects reduce their switch costs, which is the difference in reaction times between trials in which the task remains the same and trials in which the task switched. Moreover, they reduced their Stroop interference and increased their working memory span in the count-span task. In the control condition where subjects only practiced single tasks, they reduced their switch costs somewhat, but did not improve on the other tasks (Figure 23 shows the results along with the model fit).

Although the setup of the experiment is different from the Chein and Morrison study, the explanation for transfer is the same: task switching trains and promotes proactive control, and this transfers to both the Stroop task and the count-span task. We have already seen that the Stroop task and complex working memory tasks benefit from proactive control, but what is the role in task switching? As demonstrated by de Jong (1995), subjects typically employ one of two strategies for task switching: they either prepare for the upcoming stimulus before it appears, or the wait for it to appear before determining the task. If the inter-stimulus interval is long enough, preparing for a stimulus can negate switch costs almost completely. Preparing for a stimulus is a form of proactive control, while waiting for the stimulus is reactive control, just as in the Stroop task. While task-switching training may promote proactive control, the control condition might accomplish the opposite: in the single-task condition the task can be carried out optimally by a reactive strategy. It is possible that single-task training therefore produces negative transfer.

The implementation of the model of task switching closely follows the ACT-R model of Sohn and Anderson (2001). The model that is used in the pre- and posttest (as opposed to the model during training) has a choice between a proactive strategy that uses the inter-stimulus interval to prepare for the upcoming task, or a reactive strategy that waits for the stimulus before determining the task. The proactive strategy requires an operator that matches the perceptual input to a specific value, and an internal control state to a specific value, just as in the proactive strategies for the other tasks, while the reactive strategy does not.

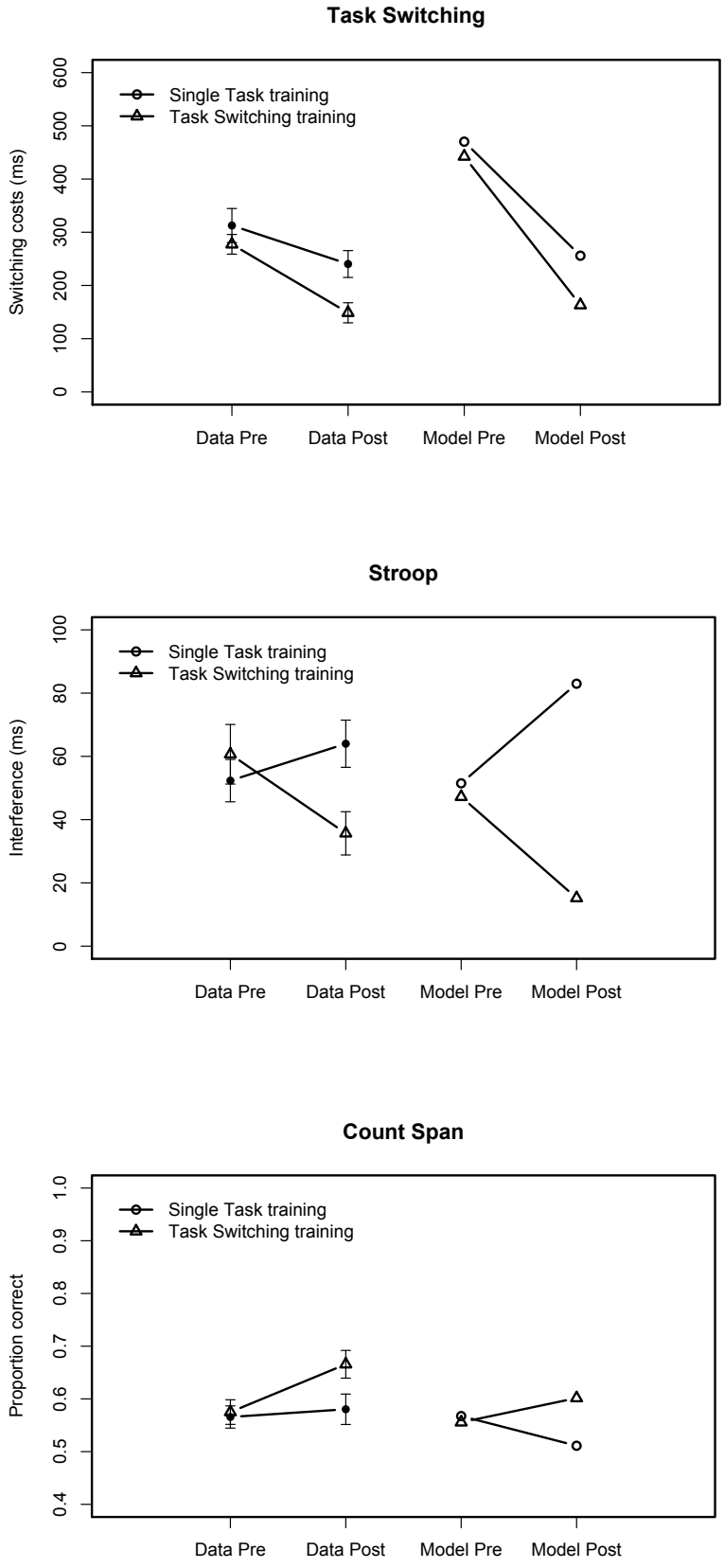

Figure 23. Results of the Karbach and Kray (2009) experiment, and the model fit.

The model of the experiment as a whole follows the same paradigm as the Chein and Morrison model: in the pre- and posttest all the task models have two options: a proactive and a reactive strategy. During the training phase, the model is forced to use a proactive strategy for task switching. The general memory strategy of the model of the count-span task is similar to that of the verbal-CWM task, except that both the proactive and the reactive strategy are incorporated in the model, allowing a choice.

Training on task switching builds and strengthens a rule that combines both external input and an internal state to take actions, while training on single tasks promotes a reactive strategy in which responses are made just on input. This transfers to Stroop, count-span and task switching itself, where either proactive of reactive strategies are favored based on training.

The kind of transfer in this, and also the Chein and Morrison experiment, differs from that of the earlier Editor and Elio models. In those models, transfer was across-the-board in that many of the condition and action sequences were reused, because the differences between the tasks were small. In the models of far transfer, the overlap is focussed on a smaller but critical overlap that promotes the use of a better strategy.

The results of the model (Figure 23) show that people do indeed reduce their switch costs with training, even if it the tasks are different. Although the switch costs are reduced in both conditions, the improvement after training task switching is significantly stronger. More interesting, and more pronounced, are the improvements on far transfer. Stroop interference shows a strong decrease after training on task switching, while showing an increase after training on single tasks.

The count-span task shows a similar pattern of results. In the single-task condition there is no improvement in proportion correct in either the data or the model fit, but in the task-switching condition there is a clear increase in performance. The model's improvement is due to an increase in the use of rehearsal, which is again initiated by a combination of external input and internal state.

In the model training on a reactive control task (i.e., the control condition) does produce negative transfer. It is not clear in the data whether that is also true for subjects. On the Stroop task there is an increase of interference, supporting the model, especially because in the Chein and Morrison experiment even the controls improved on Stroop. However, the predicted decrease in performance on the countspan task is not found in the data.

General Discussion

One of the central debates in the transfer of skill is between Plato's doctrine of formal discipline (see Lehman & Nisbett, 1990, for a more modern defense) and identical element theories (Thorndike, 1922; Singley & Anderson, 1989). The primitive elements theory is an identical elements theory, because transfer is achieved by the overlap in combinations of PRIMs among skills. However, by assuming identical elements that carry no intrinsic content

themselves, the model can explain the kind of transfer that is envisioned by the formal discipline doctrine. It shows the two theories are not necessarily at odds, because the doctrine specifies phenomena of transfer, while identical elements specifies the mechanism. The critical advantage of any identical elements theory is that it, unlike analogical reasoning, requires no explicit "transfer" of knowledge from one task to another. Learning one task automatically produces general skills that are directly useable by other tasks.

The primitive elements theory distinguishes between three types of skills: innate skills that are just single PRIMs, task-general skills that are combinations of PRIMs, and task-specific skills that are combinations of PRIMs with instantiated specifics. A single learning mechanism, production compilation, explicates the transition between these knowledge types. The theory therefore provides an account of how complex knowledge representations used in production system models can be traced back to simple origins. In addition, it offers an explanation why there is such a wide variety in the way people solve problems, because it depends on our individual arsenal of task-general skills.

The primitive elements theory increases the scope of the research agenda of cognitive architectures. The goal of research in cognitive architectures is to go beyond the study of individual tasks, and provide an encompassing theory of the mind (Newell, 1990; Anderson, 2007). However, in practice cognitive architectures have been used chiefly to model specific tasks or experiments, offering often localized explanations for the particulars of those experiments. The integrative part in such models in not completely absent, because models within a particular architecture share the same representations and mechanisms. But this kind of integration remains limited, in the sense that the common components are restricted to aspects of cognition that are assumed to be innate.

As a consequence, a common criticism of the cognitive architecture approach of modeling is that architectures afford too many different models, many of which are wrong. The reason for this is that the architecture offers only constraints that are based on innate properties of the mind. The primitive elements theory offers an additional source of constraint: if we assume that most people have acquired common cognitive strategies (iteration, rehearsal, cognitive control), then a model that uses elements of these strategies can learn the task faster than an alternative model that does not. Though it is not impossible for the alternative model to learn the task, the model that uses common strategies is faster, and it is therefore much more likely that people use a strategy that follows that model.

One way to implement this idea is to create an augmented version of Actransfer in which common strategies are already added in both declarative and procedural form. This can be used to test several alternative models for a new task. Models that show overlap in their instructions with existing strategies should learn faster than models in which everything needs to be learned from scratch, potentially producing a better fit with the experimental data.

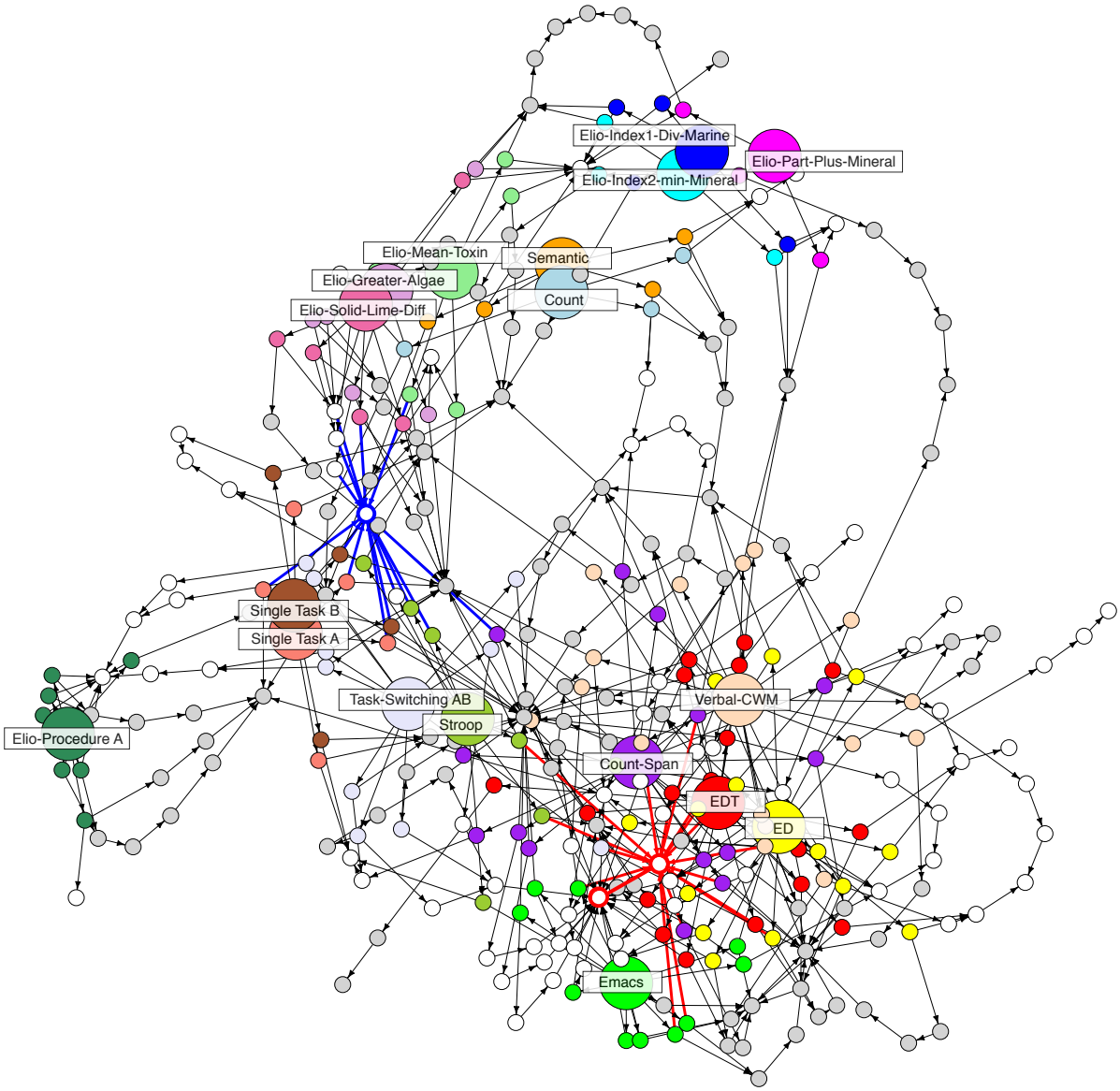

Figure 24. Graph of declarative memory with all tasks discussed in the article. In the case of task-switching and the Elio task only a single copy of similar tasks is included to avoid clutter. Also, some of the operators for the editors were removed for the same reason. The two nodes with the red outline are the conditions that support proactive strategies, while the node with the blue outline supports the reactive strategies.

Common strategies can be identified by careful analyses of tasks, which has been the main approach in this article, but also by a more data-driven method. We can implement a large set of different models, and examine what the common strategic patterns are. As a first illustration of the latter approach, consider the following illustration. If we put all the declarative knowledge for all the tasks discussed in this article together, we arrive at the structure in Figure 24. Red and blue outlines identify the nodes and edges supporting proactive and reactive strategies, respectively. The proactive strategy is not only connected to the working memory, task switching and Stroop tasks, but also to the three editor models, which is not surprising because in text editing the visual stimulus alone is also not enough to

determine the next action. Although we have to be careful in over-interpreting this image, there are many cross-connections among the tasks, and it is interesting to see that the four cognitive control tasks take a central position in the knowledge structure.

Limitations of the Models in this Article

Contrary to the ideas in the previous section, all models in this article start with just the basic PRIMs, whereas we expect people to already have an existing set productions that combine several PRIMs. Without a clear idea about what the starting point of knowledge is, this seemed to be the best approximation of the learning process (note that almost all cognitive models suffer from the problem of prior knowledge in some form). Given the vast possible space of combinations of PRIMs, learning a new task probably involves a number of new combinations, and some for which there are already existing rules. However, the existence of a rule does not mean that it can be used right away. In the example of Figure 9, the penultimate general rule consists of a combination of seven PRIMs. However, this rule competes with more general rules that are higher in the hierarchy, because production compilation does not delete them. The more general rules are applicable in many more situations, so they may win the competition. The current implementation of ACT-R does not include any decay on utility, even though an earlier version did, but in the overall economy of a rule system the frequency of use of a rule should also play a factor in its survival (indeed, in some other cognitive architectures, for example Prodigy, Carbonell, Knoblock & Minton, 1991, the term utility is specifically reserved for that purpose). Even without procedural decay the more general rules may have gained a higher utility because they are useful in more different tasks.

As a consequence, for many of the longer sequences of PRIMs it is not unlikely that rules have to be relearned. This process is faster than learning from the ground up, and this is probably why it was necessary to employ different learning speeds in the different models in this article. In the editor model, the alpha parameter that controls the speed of learning was set to 0.1, in the Elio model to 0.2, in Chein and Morrison model to 0.02, and in the Karbach and Kray model to 0.03. These values reflect the difference in average initial skill level. The more knowledge subjects already have at the start of the task in terms of useful general productions, the higher the learning rate for the model has to be to catch up. In the Elio task, many of the steps involved calculation, something that most people are familiar with and proficient in. The editor task was unfamiliar to the subjects (given that the experiment was carried out in the eighties), but did involve a fairly linear, procedural structure. Typical for Verbal-CWM and task-switching tasks is that they are unusual, in the sense that familiar processes like rehearsal have to be integrated in an unfamiliar constellation, and that different tasks have to be carried out on the same stimuli. This means that the real value of alpha is probably closer to the lower values, concurring with the idea that procedural learning is indeed very slow (Anderson & Fincham, 1994, Anderson, Fincham & Douglass, 1997).

A second reason why different models need different values for other parameters concerns the composition of the workspace. In the models in this article, the perception and motor output parts of the workspace have been reduced to just single sets of slots, which is obviously an oversimplification. If different perceptual and motor modalities are given their own place in the workspace, more PRIMs are needed. As a consequence, the current more limited version can lead to an overestimation of transfer, in particular when different modalities are involved. For example, in all models, both visual shifts of attention and manual outputs were handled by the same output part of the workspace, while they probably should have had their own separate parts. Given that this distinction was not critical in any of the models, changing this aspect would only produce slightly different model fits.

Cognitive Control

The central point this article tries to make about cognitive control is that many improvements in this function are due to skill, and not to an increase in some inherent capacity. The models of working memory, Stroop and task switching were all based on existing ideas and models (Sohn & Anderson, 2001, for task-switching, and van Maanen, van Rijn & Taatgen, 2012, for Stroop; see also Lovett, 2005, for an alternative strategy-based model of Stroop; Daily, Lovett & Reder, 2001 for complex working memory). However, all published models already presumed particular existing control skills, whereas in the PRIM models these strategies still had to be learned, with a choice between simple reactive and complex proactive strategies. The choice between reactive and proactive is not trivial, despite their better performance, proactive strategies are more complex and therefore harder to learn. The learnability of a proactive strategy depends on prior learning of proactive strategies for similar tasks: the general productions learned for the other task make it easier to learn a proactive strategy for a new task if there is sufficient overlap.

In the examples in this article, the difference typically hinged on a few combinations of PRIMs, so the question is justified whether subjects have not already learned all of these combinations earlier in life. This may be true, but this does not mean that they are equally available with respect to utility, as was discussed in the previous section. Typically, a reactive strategy is easier to learn, so if a proactive and a reactive strategy compete the reactive strategy has a higher probability of winning unless the proactive strategy has been trained. In other words, the general gist of transfer of control is correct in the models, even though the specific mechanism is more elaborate. The consequence is, however, that ultimately a different utility mechanism is required (with a reward and a frequency of use aspect) than the one inherited from the current version of ACT-R.

The two models for the verbal-CWM illustrate that reactive strategies are often more simple than proactive strategies: the reactive model does not have to maintain an internal state and has fewer operators. This is also true for many task-switching experiments in which, for example, the location of the stimulus determines the task. In those versions of task-

switching a reactive control strategy is much more simple. In the task-switching paradigm used by Karbach and Kray (2009), internal control is always necessary, and this might be the reason why it is so effective in training proactive control, because the reactive strategy is as complex as the proactive strategy.

Strategies for control can also play a role in other areas of cognitive control, like working memory control. The verbal-CWM improved on its WM performance because it improved its strategy. The choice of a list structure was sufficient for that and the count-span task, but has to be different for tasks in which knowledge in working memory is organized differently. For example, when building up a mental representation of a text, a representation with a more hierarchical or network structure is needed. Even for longer lists, Anderson & Matessa (1997) have shown that subjects tend to break up the list in smaller chunks, also creating a hierarchical representation. The primitive elements theory allows the building of different structures in memory with different strategies. The implication is that a strategy for list rehearsal may have limited transfer to other contexts in which working memory is also necessary but that require a different organization of knowledge. In other words, there is no general capacity that can be increased by working memory training, but there may instead be transfer between working memory tasks if the requirements are sufficiently similar.

Is cognitive control an innate skill, or is it determined for each individual task? The primitive elements theory takes the middle ground between theories that put few constraints on cognitive control, therefore leaving it to individual models to implement control (for example, the EPIC theory, Meyer & Kieras, 1997), and theories that have very strong architectural control (for example, using a goal stack to implement a task hierarchy, Anderson & Douglass, 2001; Altmann & Trafton, 2002). The theory does not impose any specific strategy of control, but allows transfer of any strategy between tasks (through operators), making it unnecessary to reinvent the wheel for each new situation.

Whereas skills probably play a role in individual differences in cognitive control, it is certainly not the complete story. Differences in how well the different components of the cognitive system work also play a role. In particular, many aspects of control depend on the performance characteristics of declarative memory (Daily, Lovett & Reder, 2001), but probably also on other functional modules.

The Origin of Operators

One of the questions that was not answered in this article is the origin of the operators necessary to do all the tasks. Although this is an obligation that no other symbolic model typically resolves, it is an interesting question whether we are closer to a solution. Explicit external instruction, but also reasoning from examples, or analogical reasoning can be a source of operators. Several studies have investigated how instructions and examples can be encoded in declarative memory (see for example Anderson et al., 2004 and Salvucci, in press, for some explorations of that idea). Those models use (complex) general production rules that

interpret external instructions, but without a theory on how the general rules themselves were learned. The primitive elements theory provides this missing link.

One of the assumptions in all the model simulations that were discussed here, is that the models start with a set of basic productions for each of the PRIMs. Every individual probably already has a large collection of productions in which several of these are combined. However, the space of all possible combinations is vast, so individuals probably only have a small subset of all possible general productions at their disposal. Therefore, if a new task can be build out of operators that use sequences of conditions and actions that are already available, then it will be easier than a task that requires the construction of new condition and action sequences (see also Kieras & Polson, 1986). For example, if someone has a rich experience with computers, he will need less time to learn to use a new program (even without explicit instruction) than a computer novice. An exciting new research paradigm that may shed some light on these matters is *rapidly instructed task learning* (Cole, Bagic, Kass & Schneider, 2010; Cole, Etzel, Zacks, Schneider & Braver, 2011). In this experimental paradigm, subjects receive a new instruction every few trials in the experiment. For example, subjects would be given the instruction: "If the answer to 'is it SWEET' is the SAME for both words, press your LEFT INDEX finger", with the uppercase words varying among instructions. They would then have 2-6 seconds before two words would be shown. In our framework this would mean that subjects have 2-6 seconds to assemble the appropriate operators for the particular task (presumably reusing existing lists of conditions and actions), before carrying them out in the subsequent phase. Cole et al. found evidence that if a particular instruction has been practiced, it is retrieved in the anterior prefrontal cortex (aPFC) before it is transferred to the dorsolateral PFC (DLPFC). If an instruction is novel, on the other hand, the new instruction is prepared in the DLPFC and then transmitted to the aPFC. Based on this and other research, Chein and Schneider (2012) proposed a triarchic theory of learning, consisting of a metacognitive system that generates new behavioral routines, a cognitive control network that carries out such routines in a controlled manner, and a representation system for automatic execution. The primitive elements model offers a process theory for the second and third of these systems, and a representation with which the first system can be explored.

Predictive Power and Studies that do not Show Transfer

All experiments we discussed here showed evidence of transfer. The primitive elements theory's central tenet is that transfer can occur between many different tasks. This raises the question how strong the predictive power of the theory is, because there are many studies that do not show transfer between tasks. In that respect the theory still has to further prove itself, because it has not made any predictions yet, but has only modeled existing data. The quality of a prediction of transfer depends on a number of aspects. As in cognitive modeling in general, the models of the individual tasks have to be accurate in themselves, and this is why

I have based the individual models in this article on existing models when they were available. The current system also allows some "wiggle-room" for the modeler to manipulate transfer, because the assignment of particular aspects of the model to slots in the workspace are not prescribed. However, the assumption (that is honored by all models in this article) is that the modeler attempts to maximize overlap. This is already enforced by the implementation in that it automatically reorders lists of PRIMs if that improves overlap with existing lists of PRIMs. However, fixed mappings between workspace slots and particular use of that slot would not be in the spirit of the theory, because individuals may assign different uses to specific slots.

Assuming properly tested models, there is a second aspect influencing the quality of prediction if the skills involved contain condition and action lists that are "common knowledge", skills that most people have already mastered. A research program that tries to identify these common skills, as mentioned earlier in the discussion, could offer some sort of solution to this problem, or studies that compare children to adults, where the assumption is that adults have mastered particular reasoning skills while children have not.

With these caveats in mind, the claim of the PRIM model is that given two skills and validated models of those skills, transfer can be predicted.

In the two successful far transfer studies, the Chein and Morrison model and the Karbach and Kray model, the training tasks that resulted in transfer did involve unusual combinations of operations: inserting rehearsal while there was not enough time, and switching tasks while the stimulus remained the same. Moreover, the explanation for the transfer phenomena consisted of operators that enable proactive control, which led to a performance improvement on the testing tasks. The training tasks themselves were particularly encouraging proactive control: the version of task-switching in the Karbach and Kray study does not afford and easy reactive strategy. The theory predicts that, if the training task would be a more standard task-switching experiment in which the location of the stimulus determines the task, the transfer effects would be much smaller or even absent. The verbal-CWM task used by Chein and Morrison strongly encouraged a proactive strategy, because subjects could only improve their performance with that strategy.

The Karbach and Kray study also showed the possibility of negative transfer, although this was not unambiguous in the data. By training on a reactive strategy, subjects seemed to be more drawn toward reactive strategies on at least the Stroop transfer task. More strong cases of negative transfer, for example the perverse Emacs (Singley & Anderson, 1989, chapter 4), a version of Emacs in which key bindings were changed, require a different explanation. The most likely explanation is that subjects considered perverse Emacs still as Emacs, but with modifications. Instead of building new operators, they probably created new operators that amended Emacs. Negative transfer can therefore be attributed to competition between old and new operators.

Another prediction the model makes is the diminishing return of training, for which there is some evidence in studies of transfer (Anderson & Fincham, 1994, Frensch & Geary, 1993). This prediction shows the extra added value of a learning model, because it would be hard to predict by a static model.

Can the model explain cases in which there is no transfer? Not all of them, but it can shed some light on at least some. Many studies of far transfer envision "brain-training" as if training a muscle. The primitive elements theory states that the main motor of transfer is growth of skills and strategies. Therefore, if the focus of training is on raw capacity transfer might fail. An large-scale example in that category is by Owen et al. (2010). 11,430 subjects were trained in two experimental groups and one control group. In one of the experimental groups, subjects trained on several reasoning, planning, and problem-solving tasks. The second group trained on tasks of short-term memory, attention, visuo-spatial processing and mathematics. The control group answered trivia questions. Transfer was tested on four tasks: a reasoning task and three memory tasks. In the reasoning task, a picture and a sentence were shown, and subjects had to verify the sentence as fast as possible. The reasoning task produced a small but significant effect of transfer when the experimental and control groups were compared. Performance in the other three tasks, two working memory tasks (digits and spatial) and a paired associates task, showed no effects of transfer. The two working memory tasks, however, were simple working memory tasks. They therefore lacked the key conflict between rehearsal and a secondary task that was characteristic for both the digit-span task and the Verbal-CWM task, and that were so important for transfer. Moreover, standard working-memory rehearsal is something that most people have probably already mastered. According to the authors, the tasks were specifically chosen because they are highly sensitive to pharmacological interventions. That means that, in terms of the model, they measure the proper operation of the various modules in the architecture (declarative memory in particular), but do not tap into any new general skills. The lack of transfer is therefore one that the primitive elements theory would predict.

Even Jaeggi et al. (2008) designed their experiment, in which training on the double N-back task produced improvements on the Raven test, with the goal to increase capacity, not to train skills. It is hard to explain why they did, and Redick et al. (2012) did not find effects of transfer, but a first step in answering this question would be to investigate which strategies are trained exactly in the double-N-back task, and whether or not there are alternative strategies that lead to different amounts of transfer (see, for example, Juvina & Taatgen, 2009, for a discussion on different strategies for regular N-back).

Extension To Individual Differences, Development, And Education

One of the problems in researching complex human behavior is that an increase in complexity typically means an even larger increase in individual differences. This has always been a problem for the weak-method based approaches to problem solving, because that

approach would predict much more uniformity. If, however, every individual has their own toolbox of generalized skills, it also means that individuals differ in their optimal solution for a particular problem or task (see also Howes, Lewis & Vera, 2009, who made a similar point in the context of model parameters).

Developing the individual toolbox of general strategies is part of human development, and the primitive elements theory offers opportunities to shed light on how strategies in children develop. Chen and Klahr (1999) taught children the *Control of Variables* strategy of scientific reasoning, and found that with proper instruction children transfer the strategy to other experiments. Van Rijn, van Someren and van der Maas (2003) developed a model of how children learn the balance-scale task, in which a crucial assumption is that children under a certain age cannot properly reason about how two dimensions interact (weight and distance from the fulcrum in the balance-scale task). Once children have mastered this more general skill, they were also able to learn the more advanced strategies for the task. This two-dimension reasoning skill can then be applied in a completely different domain, as shown in a model of reasoning about French determiners (Zondervan & Taatgen, 2003). It is also a skill that is necessary in some of the heuristics for decision making, such as the Take the Best heuristic proposed by Gigerenzer and Goldstein (1996).

A theory of transfer, finally, is important for research in education (Bransford & Schwartz, 1999; Carraher & Schliemann, 2002). If the goal of education is to teach skills that are optimally transferrable, we need to determine what the important skills are, and how they can be taught most effectively. If the premise of the primitive elements theory holds, this means that transfer in education is not necessarily based on content and semantics, but also on the underlying structure of skills.

References

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: an activation-based model. *Cognitive Science, 26*, 39-83.

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

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem situations. *Psychological Review, 94*(2), 192-210.

Anderson, J. R. (2007). How can the human mind occur in the physical universe? New York: Oxford university press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060.

Anderson, J. R., & Douglass, S. (2001). Tower of Hanoi: evidence for the cost of goal retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 27*(6), 1331-1346.

Anderson, J. R., & Fincham, J. M. (1994). Acquisition of procedural skills from examples. *Journal of experimental psychology: learning, memory, and cognition, 20*(6), 1322-1340.

Anderson, J. R., Fincham, J. M., & Douglass, S. (1997). The role of examples and rules in the acquisition of a cognitive skill. *Journal of experimental psychology: learning, memory, and cognition, 23*(4), 932-945.

Anderson, J. R., & Matessa, M. P. (1997). A production system theory of serial memory. *Psychological Review, 104*, 728-748.

Baars, B. J. (1988). *A cognitive theory of consciousness*. Cambridge, MA: Cambridge university press.

Bajic, D., Kwak, J., & Rickard, T. C. (2011). Specificity of learning through memory retrieval practice: The case of addition and subtraction. *Psychonomic Bulletin & Review, 18*(6), 1148-1155. doi: 10.3758/s13423-011-0151-4

Barnett, S. B., & Ceci, S. J. (2002). When and where do we apply what we learn? A taxonomy for far transfer. *Psychological Bulletin, 128*(4), 612-637.

Borst, J. P., Taatgen, N. A., & van Rijn, H. (2010). The Problem State: A Cognitive Bottleneck in Multitasking. *Journal of Experimental Psychology-Learning Memory and Cognition, 36*(2), 363-382. doi: Doi 10.1037/A0018106

Borst, J. P., Taatgen, N. A., & van Rijn, H. (2011). Using a symbolic process model as input for model-based fMRI analysis: Locating the neural correlates of problem state replacements. *Neuroimage, 58*(1), 137-147. doi: Doi 10.1016/J.Neuroimage. 2011.05.084

Borst, J. P., Taatgen, N. A., Stocco, A., & van Rijn, H. (2010). The Neural Correlates of Problem States: Testing fMRI Predictions of a Computational Model of Multitasking. *Plos One, 5*(9). doi: ARTN e12966 DOI 10.1371/journal.pone.0012966

Botvinick, M., & Plaut, D. C. (2004). Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review, 111*(2), 395-429.

Bransford, J. D., & Schwartz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. *Review of Research in Education, 24*, 61-100.

Braver, T. S., Gray, J. R., & Burgess, G. C. (2007). Explaining the many varieties of working memory variation: dual mechanisms of cognitive control. In A. R. A. Conway (Ed.), *Variation in Working Memory*. New York: Oxford University Press.

Carbonell, J. G., Knoblock, C. A., & Minton, S. (1991). PRODIGY: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for Intelligence* (pp. 241-278). Hillsdale, NJ: Lawrence Erlbaum.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum.

Carpenter, P. A., Just, M. A., & Shell, P. (1990). What one intelligence test measures: a theoretical account of the processing in the Raven progressive matrices test. *Psychological Review, 97*(3), 404-431.

Carraher, D., & Schliemann, A. (2002). The Transfer Dilemma. *Journal of the Learning Sciences, 11*(1), 1-24. doi: 10.1207/s15327809jls1101_1

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. doi: 10.3758/pbr.17.2.193

Chein, J. M., & Schneider, W. (2012). The Brain's Learning and Control Architecture. *Current Directions in Psychological Science, 21*(2), 78-84. doi: 10.1177/0963721411434977

Chen, Z., & Klahr, D. (1999). All other things being equal: acquistion and transfer of the control of variables strategy. *Child Development, 70*(5), 1098-1120.

Cole, M. W., Bagic, A., Kass, R., & Schneider, W. (2010). Prefrontal Dynamics Underlying Rapid Instructed Task Learning Reverse with Practice. *Journal of Neuroscience, 30*(42), 14245-14254. doi: 10.1523/jneurosci.1662-10.2010

Cole, M. W., Etzel, J. A., Zacks, J. M., Schneider, W., & Braver, T. S. (2011). Rapid Transfer of Abstract Rules to Novel Contexts in Human Lateral Prefrontal Cortex. *Frontiers in Human Neuroscience, 5*. doi: 10.3389/fnhum.2011.00142

Cooper, R., & Shallice, T. (2000). Contention scheduling and the control of routine activities. *Cognitive Neuropsychology, 17*(4), 297-338.

Daily, L. Z., Lovett, M. C., & Reder, L. M. (2001). Modeling individual differences in working memory performance: A source activation account in ACT-R. *Cognitive Science, 25*, 315-353.

Day, S. B., & Goldstone, R. L. (2011). Analogical transfer from a simulated physical system. *Journa of Experimental Psychology: Learning, Memory, and Cognition, 37*(3), 551-567.

De Jong, R. (1995). The role of preparation in overlapping-task performance. *Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology, 48*(1), 2-25.

De Pisapia, N., & Braver, T. S. (2006). A model of dual control mechanisms through anterior cingulate and prefrontal cortex interactions. *Neurocomputing, 69*(10-12), 1322-1326. doi: 10.1016/j.neucom.2005.12.100

Dehaene, S., Kerszberg, M., & Changeux, J. P. (1998). A neuronal model of a global workspace in effortful cognitive tasks. *Proceedings of the National Academy of Sciences, 95*, 14529-14534.

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

Forbus, K. D., Gentner, D., & Law, K. (1995). MAC/FAC: a model of similarity-based retrieval. *Cognitive Science, 19*(2), 141-205.

Frensch, P. A. & Geary, D. C. (1993). Effects of practice on component processes in complex mental addition. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 19*(2), 433-456.

Fruchterman, T. J. M., & Reingold, E. M. (1991). Graph Drawing by Force-directed Placement. *Software - Practice and Experience, 21*(11), 1129-1164.

Gentner, D. (1983). Structure-mapping: a theoretical framework for analogy. *Cognitive Science, 7*(2), 155-170.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology, 12*, 306-355.

Gigerenzer, G., & Goldstein, D. G. (1996). Reasoning the fast and frugal way: Models of bounded rationality. *Psychological Review, 103*, 650-669.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Menlo Park, CA: Addison-Wesley.

Howes, A., Lewis, R. L., & Vera, A. (2009). Rational adaptation under task and processing constraints: implications for testing theories of cognition and action. *Psychological Review, 116*(4), 717-751.

Hummel, J., & Holyoak, K. (1997). Distributed representations of structure: a theory of analogical access and mapping. *Psychological Review, 103*, 427-466.

Jackendoff, R. (1987). Consciousness and the computational mind. Cambridge, MA: MIT-Press.

Jaeggi, S. M., Buschkuehl, M., Jonides, J., & Perrig, W. J. (2008). Improving fluid intelligence with training on working memory. *Proceedings of the National Academy of Sciences, 105*(19), 6829-6833. doi: 10.1073/pnas.0801268105

Juvina, I. & Taatgen, N.A. (2009). Adding distractors improves performance by boosting top-down control. In N.A. Taatgen & H. van Rijn (Eds.), *Proceedings of the 31st Annual Conference of the Cognitive Science Society* (pp. 353-358). Austin, TX: Cognitive Science Society.

Kane, M. J., Hambrick, D. Z., Tuholski, S. W., Wilhelm, O., Payne, T. W., & Engle, R. W. (2004). The Generality of Working Memory Capacity: A Latent-Variable Approach to Verbal and Visuospatial Memory Span and Reasoning. *Journal of Experimental Psychology: General, 133*(2), 189-217. doi: 10.1037/0096-3445.133.2.189

Karbach, J., & Kray, J. (2009). How useful is executive control training? Age differences in near and far transfer of task-switching training. *Developmental Science, 12*(6), 978-990. doi: Doi 10.1111/J.1467-7687.2009.00846.X

Katona, G. (1940). *Organizing and memorizing*. New York: Columbia University Press.

Kieras, D., & Polson, P. G. (1986). An approach to the formal analysis of user complexity. *International journal of man-machine studies, 22*, 365-394.

Kieras, D. E., & Bovair, S. (1986). The acquisition of procedures from text: a production-system analysis of transfer of training. *Journal of Memory and Language, 25*, 507-524.

Klingberg, T., Fernell, E., Olesen, P. J., Johnson, M., Gustafsson, P., Dahlstrom, K., . . . Westerberg, H. (2005). Computerized training of working memory in children with ADHD--a randomized, controlled trial. *J Am Acad Child Adolesc Psychiatry, 44*(2), 177-186. doi: S0890-8567(09)61427-1 [pii] 10.1097/00004583-200502000-00010

Klingberg, T., Forssberg, H., & Westerberg, H. (2002). Training of Working Memory in Children With ADHD. Journal of Clinical and Experimental Neuropsychology (Neuropsychology, Development and Cognition: Section A), 24(6), 781-791. doi: 10.1076/jcen.24.6.781.8395

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

Lebiere, C., O'Reilly, R., Jilk, D., Taatgen, N. A., & Anderson, J. R. (2008). The SAL integrated cognitive architecture. In A. Samsonovich (Ed.), *Biologically Inspired Cognitive Architectures: Papers from the AAAI 2008 Fall Symposium* (pp. 98-104). Menlo Park, CA: AAAI Press.

Lehman, D. R., & Nisbett, R. E. (1990). A longitudinal study of the effects of undergraduate training on reasoning. *Developmental Psychology, 26*(6), 952-960.

Logan, G. D. (1988). Toward an instance theory of automization. *Psychological Review, 22*, 1-35.

Lovett, M. C. (2005). A strategy-based interpretation of Stroop. *Cognitive Science, 29*, 493-524.

van Maanen, L., van Rijn, H., & Taatgen, N. (2012). RACE/A: An Architectural Account of the Interactions Between Learning, Task Control, and Retrieval Dynamics. *Cognitive Science, 36*(1), 62-101. doi: Doi 10.1111/J.1551-6709.2011.01213.X

Mackey, A. P., Hill, S. S., Stone, S. I., & Bunge, S. A. (2011). Differential effects of reasoning and speed training in children. *Developmental Science, 14*(3), 582-590. doi: 10.1111/j.1467-7687.2010.01005.x

McClelland, J. L. (1995). A connectionist perspective on knowledge and development. In T. J. Simon & G. S. Halford (Eds.), *Developing cognitive competence: New approaches to process modeling* (pp. 157-204). Hillsdale, NJ: Erlbaum.

Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: I. Basic mechanisms. *Psychological Review, 104*(1), 3-65.

Morrison, B. M., & Chein, J. M. (2011). Does working memory training work? The promise and challenges of enhancing cognition by training working memory. *Psychonomic Bulletin & Review, 18*(1), 46-60.

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

O'Reilly, R. C., & Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation, 18*(2), 283-328.

O'Reilly, R. C., & Munakata, Y. (2000). Computational Explorations in Cognitive Neuroscience Cambridge, MA: MIT Press.

Owen, A. M., Hampshire, A., Grahn, J. A., Stenton, R., Dajani, S., Burns, A. S., . . . Ballard, C. G. (2010). Putting brain training to the test. *Nature, 465*(7299), 775-778. doi: 10.1038/nature09042

Redick, T. S., Shipstead, Z., Harrison, T. L., Hicks, K. L., Fried, D. E., Hambrick, D. Z., Kane, M. J., & Engle, R. W. (2012, June 18). No Evidence of Intelligence Improvement After Working Memory Training: A Randomized, Placebo-Controlled Study. *Journal of Experimental Psychology: General*. Advance online publication. doi: 10.1037/a0029082

Reed, S. K., Ernst, G. W., & Banerji, R. (1974). The role of analogy in transfer between similar problem states. *Cognitive Psychology, 6*, 436-450.

Rickard, T. C., & Bourne, L. E. (1996). Some tests of an identical elements model of basic arithmetic skills. *Journal of Experimental Pscychology: Learning, Memory, and Cognition, 22*(5), 1281-1295.

Rickard, T. C., Healy, A. F., & Bourne, L. E. (1994). On the cognitive structure of basic arithmetic skills: operation, order, and symbol transfer effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 20*(5), 1139-1153.

Rijn, H. v., Someren, M. v., & Maas, H. v. d. (2003). Modeling developmental transitions on the balance scale task. *Cognitive Science, 27*, 227-257.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tense of English verbs. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition.* (pp. 216-271). Cambridge, MA: MIT Press.

Salvucci, D. D. (in press). Integration and reuse in cognitive skill acquisition. *Cognitive Science.*

Salvucci, D. D., & Taatgen, N. A. (2008). Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review, 115*(1), 101-130. doi: Doi 10.1037/0033-295x.115.1.101

Salvucci, D. D., & Taatgen, N. A. (2011). *The multitasking mind*. New York: Oxford University Press.

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

Singley, M. K., & Anderson, J. R. (1988). A keystroke analysis of learning and transfer in text editing. *Human Computer Interaction, 3*(3), 223-274.

Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill*. Cambridge, MA: Harvard University Press.

Sohn, M. H., & Anderson, J. R. (2001). Task preparation and task repetition: two-component model of task switching. *Journal of Experimental Psychology: General, 130*(4), 764-778.

Stocco, A., Lebiere, C., & Anderson, J. R. (2010). Conditional routing of information to the cortex: A model of the basal ganglia's role in cognitive coordination. *Psychological Review, 117*(2), 541-574. doi: 10.1037/a0019077

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction.* Cambridge, MA: MIT Press.

Taatgen, N. (2005). Modeling parallelization and flexibility improvements in skill acquisition: From dual tasks to complex dynamic skills. *Cognitive Science, 29*(3), 421-455.

Taatgen, N. A. (2007). The minimal control principle. In W. Gray (Ed.), *Integrated models of cognitive systems* (pp. 368-379). Oxford: Oxford University Press.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say "Broke"? A model of learning the past tense without feedback. *Cognition, 86*(2), 123-155.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology-General, 137*(3), 548-565. doi: Doi 10.1037/0096-3445.137.3.548

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

Taatgen, N. A., van Rijn, H., & Anderson, J. (2007). An integrated theory of prospective time interval estimation: The role of cognition, attention, and learning. *Psychological Review, 114*(3), 577-598. doi: Doi 10.1037/0033-295x.114.3.577

Tenenbaum, J. B., Kemp, C., Griffiths, T. L. & Goodman, N. D. (2011). How to grow a mind: statistics, structure, and abstraction. *Science, 331,* 1279-1285.

Thorndike, E. L. (1922). The effect of changed data upon reasoning. *Journal of Experimental Psychology, 5*, 33-38.

Thorndike, E. L., & Woodworth, R. S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review, 8*, 247-261.

Zondervan, K., & Taatgen, N. A. (2003). The determiners model: a cognitive model of macro development and U-shaped learning in a micro domain. In F. Detje, D. Dörner & H. Schaub (Eds.), *Proceedings of the fifth international conference on cognitive modeling* (pp. 225-230). Bamberg, Germany: Universitätsverlag Bamberg.