A Computational Theory of Working Memory:

Speed, Parallelism, Activation, and Noise

A Thesis

Presented to

The Academic Faculty

by

Michael Dwyer Byrne

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy in Psychology

Georgia Institute of Technology

August 1996

A Computational Theory of Working Memory:
Speed, Parallelism, Activation, and Noise

Approved:

_____
Susan Bovair, Committe Chair

_____
Kurt P. Eiselt

_____
Alex Kirlik

_____
Nancy Nersessian

_____
Timothy A. Salthouse

_____
Tony Simon

Date approved: _____

DEDICATION

For the person who made this effort possible, providing more love, support, and friendship than I could possibly have deserved: my fianceé, Vicky Coon.

PREFACE

Getting a Ph.D. is, in some senses, the culmination of an academic journey that begins in one form or another when we are still children. As with any other journey, there have been twists and turns along the way and, as is common in long journeys, there are many people who have helped make the journey possible. The twists and turns leading to this dissertation started to become more clearly visible while I was an undergraduate and starting to learn the field. One of the things that drew my interest at the time was the study of errors, which I still believe are not nearly as well-understood as they ought to be. I began to think harder about a particular error which I decided was probably working memory-related. That work led to my Master's thesis and my use of CAPS. It also got me interested in issues of working memory. I decided that CAPS was ultimately not the way for me to go, and I began exploring other ideas about working memory. This dissertation is how those other ideas have played out.

The list of people to whom I am indebted is a large one. Since the formatting requirements for this document have already made it longer than it needs to be, a few more pages shouldn't hurt anything. I will do my best to acknowledge everyone, but I am sure to leave out someone. If you are one of those people, I apologize. Yes, even the preface probably has mistakes.

First and foremost, I would like to thank my parents for providing me with all the support over the years for my intellectual curiosity. I'd especially like to thank my father for all the times he told me he was proud of me and that I'm a good kid. Even if you weren't entirely honest all of those times, Dad, thanks.

There are a number of teachers before I got to college that I feel really made a difference in my intellectual growth. This includes Mr. Meneely (sorry if I blew the

spelling) in 4th and 6th grade, Mr. Lunde in 7th and 8th grade, Carolyn Rebholz in 9th grade, Curt “Ace” Johnson in 10th and 11th grade, Jewell Lyngaas and Barney Hall in 11th grade, and Mike Herzig and Mr. Weber in 12th grade. A particularly special thanks goes out to Lois Anderson for being more than just my 11th grade Lit. teacher, but for being a friend and coach as well. Anyone reading this thesis owes Lois some thanks, because my writing never improved more over a nine-month period than it did in her class.

The journey gained a great deal of momentum at the University of Michigan while I was an undergraduate. I would like to thank Ed Smith and Paul Bogossian for their excellent Cognitive Science course—I still believe that is the best class I ever took, and look where it got me. I’d also like to thank Jay Elkerton for the HCI class and John Laird for the Cognitive Architectures class. The most important person to thank, though, is not for his classroom instruction, but for being a terrific undergraduate advisor, going far above and beyond the call of duty as an advisor. Dave Kieras, I can’t thank you enough for really getting me started in the field. I don’t know where I would be today were it not for what I learned from you, but I would put good money down that it wouldn’t be finishing a Ph.D. in Cognitive Psychology. Or at least not one like this.

Of course, I owe a great debt to those who went before me in the field, paving the way for the journey. I have been especially influenced by, of course, Herb Simon and Allan Newell’s work—everyone in the field has. I think, though, that the two works that really got me in to the field, despite the fact that I was starting 9th grade when they came out, were Card, Moran, and Newell’s *The Psychology of Human-Computer Interaction* and John Anderson’s *The Architecture of Cognition.* I am privileged and honored to be going to work with John Anderson this fall; John, your work has always had a great influence on me. I also owe a great debt to Alan Baddeley for all the work he has done in furthering the study of working memory.

Of course, the influences that seem the strongest are the closest and most recent. Graduate school is, after all, where one becomes an adult as a researcher. The people to whom I owe the biggest debt of all are the members of my committee who, despite the

craziness that overtook Atlanta for the 1996 Centennial Olympic Games, have done a terrific job of turning around drafts and giving me comments that, as I'm sure they can attest, have greatly improved this dissertation. Individual thanks are in order.

First, I want to thank my advisor, Susan Bovair, for what has to be called Herculean patience with me, being my advisor for the five years I have been here at Georgia Tech. Not just thanks for the academic advising, but for five years of great talks, many of which had little or nothing to do with my academic work, but a great deal to do with friendship. I thank you, and wish you well in whatever it is you choose to pursue.

I owe another terrific debt to Tim Salthouse—this much should be obvious to anyone reading the dissertation. Thank you for letting me co-opt so many of your good ideas, for teaching me most of what I know about cognitive aging, for doing such excellent science, for providing me with data, and for writing "great job" on a draft of this dissertation. From you, that is high praise indeed and I'm not sure I deserve it. If there is ever anything I can do for you, please let me know—I owe you a lot.

Thanks to Tony Simon, modeling man, for reminding me repeatedly that there are other ways of thinking about modeling and how models do (and should) work, and for some poignant career advice. Good luck with whatever direction you go.

Thanks to my buddy, Kurt Eiselt. You probably don't know it, but you had a lot to do with my decision to even come to Tech. You are such a good guy to just chat with, and you do good work to boot. Keep up that good work—even administrators have a little time for research, right?

Once in a while, in trying to soak up the information in a field, it's possible to get in something of a rut in one's perspective. Alex Kirlik deserves a warm thanks for trying to make sure that I could see the rut for what it was. We may not always agree, Alex, but you've made me think a lot about things I would not otherwise have considered, and there's a lot of value in that. I would like to have had more of that thinking included here in the dissertation, but it will just have to wait. Keep a lookout, though, I will get to it.

Finally, thanks to Nancy Nersessian for agreeing to be something of a late addition to the committee and reminding me, as hoped, about the bigger picture. If I ever find the ultimate, perfect, double-chocolate-fudge brownie recipe, I promise I'll make some and send them to you, and the recipe with it. It's the least I can do.

There are faculty here at Georgia Tech not on my committee who deserve acknowledgement as well. Chris Hertzog heads up this list, for teaching me almost everything I know about statistics and letting me TA it with him for two years. A good time was had by some, anyway. I'd also like to thank Jim Foley for the work he let me do with him—a great gain for Mitsubishi will be a great loss for Tech, Jim, but good luck with it. Thanks also to the multimedia project crew of John Stasko, Mark Guzdial, Ashwin and Preetha Ram, and especially Richard Catrambone. It's been good working with all of you, and I do appreciate you giving me the chance to do it. Also here at Tech, members of the support staff merit a huge "thank you," especially Wanda Abbot and Jane Crawford in Psychology and Joan Morton over at GVU.

Thanks also to a bunch of the students at Tech for both personal friendship and professional camaraderie. In no particular order that would be: Bill Collins, Bob King, Alex Kell, Angel and Beth Cabrera, Mike Cox, John Akers, Megan Moloney, Betsy "Betty" Meinz, Peter Batsakes, Jeff Millians, Anthony Francis, Kenny Moorman, Todd Griffith, Marjo Anderson, Sharon Carter, Marcia Crosland, Linda Kneidinger, Stuart Tross, Larry Najjar, Rebecca Snyder, Chris Whaley, Andy Edmonds, Leslie Wilson, and everyone else I can't think of, probably because of the parties involved. (I am not using "parties" to refer to people, by the way.) And, of course, thanks to Dave Rettinger, my friend from Michigan who remains a colleague and friend, though now at Colorado.

I would also be remiss if I didn't thank the folks outside Georgia Tech who made a difference. Certainly the top of this list belongs to Marcel Just and Sashank Varma, the folks who made SPAN possible by first providing CAPS, as well as awesome 11th-hour rescues. Thanks also to the various faculty outside of Tech who didn't have any reason to give me the time of day but did anyway, especially: Bonnie John, Richard Young, Paul

Thagard, Reid Hastie, Peter Polson, Clayton Lewis, and Randy Engle (who is here at Tech now but wasn't when I first asked a favor).

Thanks are also due to the organizations that helped fund my time here in graduate school, which include the National Science Foundation (via a Graduate Fellowship), the Graphics, Visualization, and Usability center (via a Research Assistantship), and the Office of Naval Research (an ONR grant funded the research assistantship). Nothing in this dissertation, of course, represents views of any of these organizations—they just helped me get through the paying rent and eating part of graduate school.

Of course, there are other groups that deserve thanks, and these are the tool makers. Tops on this list is Apple Computer, for making machines in accord with Steve Jobs's vision: insanely great. Thanks for saving me from DOS and UNIX—I could work with them, but because of the Mac I don't have to. Thanks to Bare-Bones Software for BBEdit, Digitool for Macintosh Common Lisp, Apple again for HyperCard, Inspiration Software for Inspiration, Qualcomm for Eudora Pro, Now Software for Now Utilities and Up-to-Date, APS for storage products, Aladdin for Stuffit Expander, Dartmouth for Fetch, Xerox for TextBridge, Peter and Quinn for Internet Config, CE Software for QuicKeys, Skidperfect for Extensions Strip, James Walker for Dialog View, and MetroWerks for CodeWarrior. There are other tools that I use, of course, but these are the exceptional ones that also have acceptable customer relations. Much work was done more easily because of all of them.

And last but certainly not least, Vicky Coon. Yes, it's dedicated to you, but that doesn't mean I shouldn't thank you, too. So thank you—I wouldn't have made it without you.

The completion of one journey often launches the next, and that is certainly the case here. I won't miss being a student, and I am looking forward to moving on with the rest of my life. More topically, I am also looking forward to continuing as a researcher in psychology. Herein lies the end of the student journey, but the beginning of the professional one.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF ILLUSTRATIONS

Page

SUMMARY

The construct of working memory has played a key role in cognitive psychology's understanding of individual differences on a wide range of tasks, especially differences between young and older adults. Despite this importance, working memory has received little attention from computational modelers, especially in regard to the results from the cognitive aging literature. The thesis presents a computational theory of working memory called SPAN, which tries to take into account what is known about the working memory mechanisms of decay, displacement, and especially processing speed. In many ways, SPAN is a production system instantiation of a slowing theory of working memory and aging much like that advocated by Salthouse. The motivation and background for the theory is laid out and the theory described. To test the theory, models of three tasks are presented: the fan effect sentence verification task, the Digit Symbol speed test, and the Computation Span working memory measure. Not only can SPAN produce the kinds of young vs. old differences observed in empirical studies of these tasks, but the SPAN models mirror the quantitative differences as well. The results provide further support for slowing theory and demonstrate that empirical results from the cognitive aging literature are amenable to the formal techniques usually associated with cognitive science. SPAN presents a promising new way of looking at working memory, and prospects for future work in modeling, perception/action, and learning are also discussed.

CHAPTER I.

OVERVIEW

The research described in this thesis concerns working memory. More specifically, it describes a theory of working memory that is instantiated as a production system called SPAN (which stands for Speed, Parallelism, Activation, and Noise). The central claim of the theory is that the primary factor responsible for individual differences in working memory is a difference in the rate of activation of memory items, a kind of processing speed index. This theory is largely based on work in cognitive aging, and results from the cognitive aging literature are simulated to demonstrate the power of the theory. A more detailed view of what appears in each chapter follows.

Chapter 2 provides some of the background context and motivation for the thesis. This chapter is primarily concerned with the relationship between two traditionally separate research areas, cognitive aging and cognitive science. Working memory serves as the common ground between the two disciplines, as it has played a central role in both areas.

Chapter 3 is an examination of the construct most central to the thesis, working memory (WM). In order to have a theory of working memory, it is necessary to have a clear definition of the construct. Chapter 3 is an attempt to provide such a definition by providing a brief sketch of the evolution and current status of the construct.

Of course, for the construct to be a useful one, it has to play an important role in psychology's understanding of human behavior. Working memory is indeed a pervasive piece of our understanding of a wide range of experimental results, and Chapter 4 gives a flavor of this pervasiveness, particularly as it relates to individual differences in cognition.

Since SPAN is a computational theory, it requires a well-specified set of mechanisms. Chapter 5 presents several mechanisms that are believed to be key in understanding the functioning of working memory, and it is upon several of these mechanisms—especially decay, displacement, and speed—that SPAN is based. In Chapter 6, issues related to the computational nature of the theory are discussed. Justification for the use of computational modeling in general and production systems in particular are presented, and several caveats, particularly issues surrounding task analysis, are considered.

The theory itself is presented in considerable detail in Chapter 7. The basic mechanisms and their interactions are described first at a relatively high level, followed by the precise equations used to determine the specific behaviors of SPAN. Other issues related to both the use and implementation of the theory, such as goal management, are also discussed.

The next three chapters describe models of cognitive aging experiments built with SPAN: Chapter 8 presents a model of the fan effect observed by Gerard, Zacks, Hasher, and Radvansky (1991) in sentence verification, Chapter 9 the Digit Symbol speed test (e.g. Salthouse & Coon, 1994), and Chapter 10 the Computation Span working memory measure (e.g. Salthouse & Coon, 1994). For each of these tasks, the task itself and the empirical results are described, followed by a description of the model for the task. The predictions of the simulation model for each task are then compared to the empirical results and the correspondence discussed.

Chapter 11 presents discussion of three issues raised by the three models: the demonstrated sufficiency of the slowing account, the uniqueness of SPAN relative to other production system theories of cognition, namely CAPS, Soar, ACT-R, and EPIC, and the difficulty in modeling errors, a problem not unique to SPAN. This concludes the discussion of work that has been done with SPAN; Chapter 12 describes a number of promising places for further exploration. This includes expanding the number and scope of the tasks modeled and tackling issues in making SPAN into a more general theory of

cognition, namely perception/action and learning. Finally, Chapter 13 presents a summary and discussion of the thesis as a whole.

CHAPTER II.

COGNITIVE SCIENCE, COGNITIVE AGING, AND WORKING MEMORY

This chapter introduces the context and motivation for the remainder of the work. The central problem of interest is working memory. Despite the fact that the construct of working memory has, in one form or another, been with the field of psychology since at least the time of William James, it is not theoretically well-understood. The most powerful theoretical tools psychology has to address the problem are mathematical or computational, such as work in cognitive science. Unfortunately, researchers in cognitive science have not done a good job of taking into account one of the richest sources of data on working memory, that acquired from years of research in cognitive aging. There is, however, common ground between cognitive science and cognitive aging, and this work attempts to find that common ground and capitalize on the strengths of the two fields. This chapter is intended to be a high-level overview of the issues related to spanning the gap between the two subdisciplines; specific data and claims will be described in more detail in subsequent chapters.

1. Cognitive Science

It would be foolish to attempt to include a complete history or even summary of the emerging interdisciplinary effort referred to by the label "cognitive science" in the scope of this document.[1] However, because of the relationship cognitive science has with psychology in general and cognitive aging in particular, some comments are in order. Within psychology, cognitive science is typically practiced by those researchers who

[1] For those interested, Gardner (1985) provides a good starting point.

approach problems in understanding human cognition from a computational, or information-processing, perspective. Not all psychologists who consider themselves cognitive scientists spend their time and energy constructing computational models, but they are typically sympathetic to and supportive of modeling endeavors. It may not be the case that cognitive science is exclusively associated with computation, but within psychology, computational models are most often associated with cognitive science. This is an admittedly narrow caricature of cognitive science within psychology but it captures one of cognitive science's most important features: a focus on computation.

Within the computationally-oriented community, a distinction is typically drawn between between "knowledge" and "architecture." Knowledge is the facts and procedures we all acquire as a result of our experience, i.e. the contents of long-term memory. Architecture, on the other hand, can be thought of as the "fixed processing structure" (Newell, 1990, p. 82) that attempts to apply the knowledge an agent has in pursuit of the agent's goals. In cognitive science, the difference between individuals is typically thought of as a result of differences in the knowledge, and has generated a great deal of interest in novice/expert differences in a wide variety of domains.

However, little attention has been paid to individual differences that are a result of differences in the architecture. This may be because it is assumed that the architecture is uniform across individuals—work by the Soar group, for example, seems to echo this perspective. There are problems with this assumption, though. One difficulty with locating differences between individuals entirely in the knowledge is that knowledge is generally considered to be domain-specific; that is, people have different kinds and amounts of experience in various domains.[2] While this is certainly true, there is a problem—if there are

[2] There are almost certainly fundamental pieces of knowledge, such as basic knowledge about quantity, that cut across a wide variety of domains—the child development literature has long been concerned with determining just what these pieces of knowledge are and how they are acquired. It is unlikely, though, that there are substantial differences in this kind of foundational knowledge between adult members of modern Western society, as lacking such knowledge would make everyday tasks excessively difficult. Thus, this kind of knowledge difference is probably not the source of the general young-old performance differences found in the cognitive aging literature.

systematic performance differences that span a wide variety of tasks, the "difference in knowledge" account is weak. Why would one person be much different than another not just on one task neither of them had ever seen, but on an entire battery of such tasks? Since neither individual should have any domain-specific knowledge about the tasks if they are new tasks, why should they differ? The difference must then lie in the architecture, not in the knowledge. But is there evidence for this kind of general performance difference? Indeed there is. For reasons that are unclear, this evidence has gone largely unnoticed in the cognitive science community.

2. Cognitive Aging

Cognitive aging is the subfield within cognitive psychology that is concerned with studying the effects the normal adult aging process has on cognition. The cognitive aging literature is a rich source of the kind of data that would support the idea of a difference in cognitive architecture: differences on a wide variety of knowledge-lean tasks. This effort was initially rooted in early psychometric studies of intelligence. Based on such studies, a distinction was drawn by researchers between "fluid" and "crystallized" intelligence. Tasks that load on a crystallized intelligence factor require the use of accumulated knowledge, such as vocabulary tests and "general knowledge" questions. Fluid tasks, on the other hand, tend to be novel (or use novel stimuli) to all subjects and require the construction and manipulation of new representations and partial products. An abstract reasoning task would be an example of a fluid task. What is striking about the results from cognitive aging is that the adult aging process appears to have relatively little effect on performance on crystallized tasks, but results in a substantial decrement in a wide range of fluid tasks (for a review, see Salthouse, 1991a). Or, to cast it in other terms, as adults age, their knowledge is unaffected (it may even improve) but there appears to be a degradation in the performance of the architecture. It would be very difficult to make a tenable argument that the differences found in this literature are based on differences in knowledge, since performance on knowledge-intensive crystallized tasks remains steady but performance on

knowledge-lean fluid tasks does not.

The strength of the work in cognitive aging is data. Unfortunately, there has been surprisingly little theory about what it is that produces the widely-observed decrement in performance[3] —cognitive aging work has been primarily, though not exclusively, concerned with describing the various declines (or, in some cases, lack of declines). Much as it is not clear why cognitive scientists are generally not aware of the evidence for architectural differences found in aging, it is not clear why researchers in cognitive aging have by and large not adopted the powerful theoretical tools of cognitive science. What is largely missing in cognitive aging are explanations for the decline in performance at the level of causal mechanisms. Two such explanations have emerged in the field, slowing and inhibition deficits (a great deal more will be said about both of these later), but neither of these explanations have demonstrated an ability to account for the data in a mechanistic way. That is, it has not been demonstrated that the mechanisms of either slowing or inhibition deficits are sufficient (much less necessary) to produce the observed performance differences. It has been formally claimed that they are capable of doing so, but these claims have not been supported with strong mathematical or computational models.[4] Slowing and inhibition deficits are theories in search of sufficiency.

3. Working Memory

Despite their conspicuous lack of overlap, cognitive science and cognitive aging do have some common ground. Since both fields are concerned with cognition, it would be somewhat surprising if they did not. One striking parallel between the two areas is the major distinction each field draws: in cognitive science, this is the knowledge/architecture distinction, and in cognitive aging, it is the crystallized/fluid distinction. Perhaps this

[3] The slowing theories of Salthouse (e.g. 1985, 1996) and Cerella (1985) and the inhibition deficit theory proposed by Hasher and Zacks (1988) are the most noteworthy exceptions.

[4] Some very general models regarding slowing and cognitive performance have been presented (e.g. Salthouse, 1988; Cerella, 1990), but slowing has yet to be shown to be computationally sufficient to account for the quantitative differences on any specific task.

common dichotomy has not been acknowledged because of the differences in terminology, but the two distinctions are strikingly parallel. Crystallized tasks are those that draw on accumulated knowledge, and fluid tasks are those that draw on more basic or general cognitive operations—the architecture.

On the fluid or architectural side of this shared distinction is possibly the largest piece of common ground: the shared construct of working memory (WM). In cognitive science, working memory is typically a feature or function of the architecture, while in cognitive aging it is often considered the "workspace" in which fluid tasks are performed. Thus, working memory is where fluid intelligence or "changes in the architecture" would be realized at a theoretical level in cognitive aging and cognitive science, respectively. Working memory is an important component in many cognitive science theories (though individual differences in working memory are usually not) and the cognitive aging community has collected a small mountain of data regarding WM. There is still a gap here, since computational theories have not been applied to the working memory data from the cognitive aging literature. Cognitive science and cognitive aging have, as yet, failed to make use of the knowledge found outside their own traditional disciplinary boundaries. The goal of the remainder of this paper is to demonstrate that cognitive science and cognitive aging can be informed by one another and that this union can be fruitful for both sides. This will be accomplished by developing a cognitive science-style computational theory of working memory and applying that theory to some of the data from experiments in cognitive aging. The computational theory is based on a construct central to a great deal of work in cognitive aging, that of slowing. The theory formalizes the notion of slowing and can then provide demonstrations of computational sufficiency for basic phenomena found in cognitive aging. Since slowing has been shown to be closely related to the decline in working memory associated with normal aging, a demonstration of the effectiveness of the theory in accounting for differences in working memory capacity would provide a strong link between cognitive science and cognitive aging.

It is hoped that this will firmly establish the common ground between the two

fields, the shared interest in working memory. Further, exploring this common ground can help strengthen theory in both areas. This exploration should help cognitive science by expanding the scope of the theories to an area in which there is extensive empirical data; at the same time, it can help cognitive aging by providing formalization and specificity in what has been a theoretically weak research endeavor.

CHAPTER III.

WHAT IS WORKING MEMORY?

The psychological construct of working memory has a considerable history, though the term “working memory” has not always been used by researchers to describe it. Probably one of the most cited works in cognitive psychology is Miller’s (1956) seminal paper, which describes the “magic number” seven (plus or minus two). The human information-processing system was described as having an information “bandwidth” of roughly seven “chunks” of information. This bandwidth of capacity for immediate storage was (and often still is) referred to as “short-term memory.” Though Miller was not detailed in his specification of what a “chunk” is, it is generally considered to be one meaningful unit of information. Thus, while the letter string “runspot” is seven letters long, for most English speakers it can be grouped into two meaningful chunks, “run” and “spot.” The “plus or minus two” was included to reflect the fact that the capacity of the limited bandwidth differs between individuals. One of the reasons this paper is so well-referenced is because the “chunk” construct has remained a central one in cognitive psychology and has gone more or less unchanged in the last forty years.

The next way station along the history of working memory is the “modal model” of Atkinson and Shiffrin (1968), a view of the human memory system which is still taught to introductory psychology students. In the modal model, there are essentially three components in the human memory system: perceptual buffers, the long-term store, and the short-term store (STS). In the modal model, the STS was a key component as it was thought to be the location information had to occupy in order to be actively processed.

The STS was conceptualized as more or less a set of “bins,” where the number of

bins is fixed for each person, though the number of bins was presumed to differ between individuals. When all the bins were full, one bin would have to be emptied in order for something new to come into the STS—a kind of interference or displacement. The size of each bin was, of course, one chunk. Items were retained in the bins via rehearsal, and those items displaced from a bin decayed over time. Though the modal model is generally considered dated, the decay and displacement ideas presented by Atkinson and Shiffrin are powerful ideas that remain in cognitive psychology today.

The next major point along the evolution of the construct came along with the introduction of the term "working memory" by Baddeley and Hitch in 1974, further refined and explicated in Baddeley's seminal book *Working Memory* (1986). Where the modal model posited the role of immediate memory as one of only storage, Baddelely's conceptualization includes processing as well. That is, working memory consists of not only the information currently being processed (as in the STS of the modal model), but also the processing itself. This view was based largely on dual-task studies wherein one task was to remember a string of digits or letters and the second was another cognitive task, such as simple reasoning or sentence understanding. It was consistently found that the storage demands of the memory task interfered with the time course of the processing task—that is, that storage essentially displaced processing to some degree. Thus, it was posited that the same system, working memory, was responsible for both storage and processing.The relationship between long-term memory and working memory is not considered in much detail in this conceptualization.

Baddeley (1986) presents a number of other aspects of his conceptualization involving separate working memory systems for phonological/verbal information (called the "articulatory loop") and another for visual/spatial information (the "visuospatial sketchpad"), coordinated by a "central executive." While Baddeley's notion of working memory being responsible for both storage and processing has gained widespread acceptance in cognitive psychology, the subsystem architecture remains somewhat contentious. One reason for this may be the lack of specificity in the description of the

system. Certainly, a great deal of research has been devoted to examining the properties of the slave systems (especially the articulatory loop), but very little work has been done by Baddeley or his colleagues on what the central executive actually is, much less what its properties are.

Somewhat orthogonal to Baddeley's work is the perspective offered by Anderson's (1976, 1983, 1993) ACT family of theories. In the ACT view, working memory is simply the "active" portion of long-term memory (LTM), which is a network of chunks and a set of production rules. More active chunks are matched faster by the productions, and so inactive chunks are effectively unusable because it would take too long to match them. Activation management in the more recent instantiations of ACT (ACT-R in particular) is relatively complex and it is not entirely clear how the ACT view of working memory addresses the processing aspect so important in Baddeley's view. On the other hand, this view of working memory makes the relationship between long-term memory and working memory clear; working memory is simply a subset of long-term memory.

But if WM is defined by LTM, why bother separating the two—what is the fundamental difference? The essential issue here is search control. It is generally accepted that human long-term memory is vast. If this is the case, having every memory element in a large network simultaneously active at meaningful levels would make determining which piece of information to use at any particular time computationally intractable. The amount of search or pattern-matching that is necessary at any given time is dramatically reduced by having only a portion of LTM active. Of course, this means that a piece of information that is needed may be unavailable (or simply take too long to access)—an experience which almost everyone has had.

Probably the most integrated working memory construct to date is the one outlined by Just and Carpenter (1992), which is similar to but more detailed than the one presented by Cowan (1988). In the Just and Carpenter view, working memory involves both storage and processing as well as the activated portion of long-term memory. In addition, working memory includes the current goal context and any partial products. Both goals and partial

products are task instance- or time-specific, and may not be stored in LTM. Partial products, such as carries in arithmetic (Hitch, 1978) and syntactic bindings during sentence comprehension (MacDonald, Just, & Carpenter, 1993) are typically a function of the specific task instance—the particular arithmetic problem or sentence being comprehended. It is important to keep track of such information for a short time, but since such information is not relevant to other problems, there is no reason for it to be part of LTM. Goals are similar to partial products in that they are manipulated (in this case, set and satisfied) according to the status of a particular problem. One might argue that some partial products and goals could be stored in LTM, particularly in the case of a person who often solves similar or identical problems. While that may well occur, the point of inclusion of partial products and goals in WM is that if a partial product or goal is not stored in LTM, that element would not be covered by the "activated LTM" view of working memory.

In the Just and Carpenter conceptualization of working memory, all these aspects (activated LTM, storage of partial products and goals, and processing) are tied together by activation, which is thought of as a limited resource. A maximum amount of activation is postulated to be available at any given time, which creates limitations in the amount that can be simultaneously stored/processed, and individuals are assumed to differ in the maximum amount of activation.

This provides a good starting definition of working memory as including activated LTM, goals, partial products, as well as processing and having some kind of boundary on total activity that varies between individuals. How that boundary arises is the subject of later sections; the next important issue to cover is the importance of the boundary itself.

CHAPTER IV.

PERVASIVENESS OF WORKING MEMORY

As has been alluded to, differences in working memory capacity have been invoked to explain differences in performance on a variety of tasks, both differences between individuals in the same age group and across age groups. While I will not present a detailed review of each study cited, this brief survey should serve to point out the convergence found in the recent literature on the importance of working memory.

This is particularly true with respect to the cognitive aging literature, where differences between adults of various ages are reduced to differences in working memory capacity. There are typically two ways in which this argument is made in the aging literature. The more methodologically sound way of doing this is to actually measure working memory capacity and use it as a statistical control in hierarchical regression. Typically, results show that most of the task variance associated with age is accounted for by controlling for working memory capacity. A more common (but probably less sound) approach is to simply attribute performance differences to working memory differences by pointing out the working memory demands of the task. For this argument to be convincing, there needs to be general agreement that working memory capacity does, in fact, decline with age. This general agreement exists with considerable justification. Several recent studies have used one or more standard measures of working memory in young/old or continuous age samples and found evidence for an age-related decline,[5] such as Morris, Gick, and Craik (1988), Gick, Craik, and Morris (1988), Dobbs and Rule

[5] "Age-related" will be used to refer to changes associated with normal aging in adults unless explicitly noted otherwise; in particular, this does not refer to changes associated with development from childhood to adulthood.

(1989), Foos and Wright (1992), Van der Linden, Brédart, and Beerten (1994), though there are many others. Probably the most convincing evidence for the decline is provided by the elegant meta-analysis of the aging literature conducted by Verhaeghen, Marcoen, and Goossens (1993). They combed the aging literature and discovered that, on average, working memory declines are large and universal regardless of what method is used to assess working memory.

Now, if working memory is as central to human cognition as its place in the theories would suggest, it should be the case that as people age, their performance should decline on a wide variety of tasks. In addition, even within an age group, substantial individual differences should be present to the extent that there is variability in working memory capacity within that age group. This is exactly what has been found in a wide variety of domains.

One of the areas in which this has been most prominent is language. Individual differences in working memory within young populations have been linked to performance differences in comprehension (e.g. Daneman & Carpenter, 1980; Turner & Engle, 1989; Engle, Cantor, & Carullo, 1992), production (Daneman & Green, 1986), memory for discourse (Goldman & Varma, 1996), verbal fluency (Daneman, 1991) and processing syntax (Just & Carpenter, 1992; MacDonald, Just, & Carpenter, 1993). Working memory has also been implicated as the source of differences between young and old in text comprehension (Davis & Ball 1989), text recall (Morrow, Lierer, & Altieri, 1992), narrative processing (Morrow, Altieri, & Lierer, 1992), dual-task speech processing (Tun, Wingfield, & Stine 1991), handling syntactic complexity (Norman, Kemper, & Kynette, 1992), resolving anaphoric reference (Light & Capps, 1986) and segmentation comprehension (Stine, Wingfield, & Myers, 1990).

Of course, if language alone showed working memory-related performance, the case for working memory as the global architectural difference between people would not be particularly strong. Fortunately, there are more general measures. One of the most famous tests designed to measure Spearman's *g*, or general intelligence, is the Raven's

Progressive Matrices family of tests (e. g. Raven, Court, & Raven, 1983), which is a spatial matrix reasoning test. Just, Carpenter, and Shell (1990) provides evidence that performance differences within a young age sample on the Raven's is tied to working memory capacity. Clearer relationships between matrix reasoning and working memory capacity have been found in young vs. old designs (Salthouse, 1993; Babcock, 1994). These kinds of results have been replicated in other kinds of reasoning tasks as well. One of the best examples of this is Kyllonen's work, in which he has employed factor analytic techniques to demonstrate that the working memory and reasoning factors that emerge are essentially the same factor (Kyllonen & Christal, 1990; Kyllonen, 1995). Salthouse has also demonstrated age and working memory related differences on a variety of reasoning tasks including integrative reasoning, verbal reasoning, paper folding, analogies, cube assembly, and more (see, among others, Salthouse, Mitchell, Skovrokek, & Babcock, 1989; Salthouse, 1991b, 1992b, 1992c). This provides very convincing evidence that working memory is a general difference in cognitive architecture.

But there is still more evidence that working memory is an important source of individual differences. One of the classic memory experiments, which has been used to support spreading activation theories of memory (e.g. Anderson, 1983) is the fan effect (e.g. Anderson, 1974), or the amount that recognition responses are slowed by the interconnectedness of the terms involved. Subjects in this paradigm study a number of simple sentences such as "the doctor is in the park" and "the hippie is in the theater." Subjects are then shown sentences involving the same actors and locations that appeared in the study set and asked whether they had studied the sentences or not. Response time is consistently (e.g. Anderson 1974, 1983; Anderson & Reder, 1980) a function of the number of sentences in which the actor and location appear—as the total number of appearances goes up, so does the response time. Again there is evidence that working memory capacity is related to amount of fan effect observed (Cantor & Engle, 1993) and evidence that older adults are more susceptible to fan than younger adults (Gerard, Hasher, Zacks, & Radvansky, 1991).

There are a number of other domains in which working memory differences have been shown. One such area is procedural errors—lower-capacity young adults are more prone to such errors than higher-capacity adults (e.g. Just, Carpenter, & Hemphill, 1996; Byrne & Bovair, in press). In arithmetic, differences in error rates come out both in young populations (e.g. Hitch, 1978) and in old vs. young designs (Campbell & Charness, 1990). Skill acquisition has been shown to slow down with increased task-related memory load (Strayer & Kramer, 1990; Reber & Kotovsky, 1992). Working memory differences have been shown to account for age-related decline in procedural assembly tasks (Morrell & D. C. Park, 1993) and spatial memory tasks (Cherry & D. C. Park, 1993). This could continue, but the fundamental point should be clear: the list of tasks on which working memory differences relate to performance differences is clearly large and encompasses a wide variety of tasks. This implies that there are systematic differences in cognitive architecture between persons and that differences in working memory capacity is likely to be a source of those differences. With this amount of evidence, it is extremely difficult to argue against the position that working memory is a key factor in understanding general individual differences in human cognition.

CHAPTER V.

MECHANISMS OF WORKING MEMORY

Now that the construct of working memory has been outlined and its importance highlighted, the next important step in understanding working memory is understanding the mechanisms that yield the observed phenomena. A number of mechanisms will be discussed in the context of how they should be modeled: decay, displacement/interference, the role of experience, processing speed, inhibition deficits, and a related note on the role of external memory. The ideas behind decay and displacement are old ideas and are not particularly contentious, which cannot be said for the more recent ideas concerning processing speed, inhibition deficits, and the role of external memory. Thus, the sections covering these topics are considerably longer to make clearer the case presented.

1. Decay

One of the features that seems to be present in the human WM system is that information that is not being actively maintained decays over time. Probably the best single experiment which illustrates this phenomenon is Reitman's (1974) classic paper using a variant of the Brown-Peterson task. Decay of information in WM is how information in WM is "lost" in several theories, including ACT* (Anderson, 1983), the MHP (Card, Moran, & Newell, 1983), and even as far back as the original buffer model of Atkinson and Shiffrin (1968). In accord with the empirical data on the subject (i.e. Peterson & Peterson, 1959; Melton, 1963), these theories use exponential decay functions. From a computational standpoint, this implies that, as time elapses, information in WM that has not been accessed should become increasingly difficult to access.

Naturally, one question that arises is "why?" That is, why do items decay over time? It is impossible to know precisely why items decay, but one reason that seems reasonable is derivable from Anderson's (1990) rational analysis approach. The probability that a memory item will be needed decreases as time passes, and, in accord with this, the human cognitive system decreases the centrality (via decreasing activation) of the item. Anderson (1990) offers a substantially more complete analysis of this idea. Based on this analysis, ACT-R (Anderson, 1993) uses a power function for decay rather than an exponential function. Qualitatively, power functions and exponential functions are quite similar—it may not be possible to clearly discriminate one from the other based on current experimental data.

2. Displacement/Interference

Decay is not necessarily the only means by which information is lost from working memory. One of the debates that went on for many years in the working memory literature is the "decay vs. displacement" debate—essentially, "how does information leave working memory?" While there appeared to be evidence for decay of information from WM, there also seems to be displacement—that is, new items force older items out of working memory. People's ability to process information also seems to suffer when there are more items in WM, so processing seems to be displaced by storage as well.

The idea of displacement[6] of older WM items by newer items goes back at least as far as the 1960's and probably further than that (see Melton, 1963; Keppel & Underwood, 1962). This idea was also incorporated into the influential Atkinson and Shiffrin (1968) memory model, in that items in the "rehearsed" part of the short-term store could be displaced from the rehearsal buffer by the entry of new items. (Displaced items then decayed, as mentioned above.) This is the central mechanism responsible for forgetting in at least one computational model, that of Kotovsky and Kushmerick (1991). A variant of

[6] Displacement is also referred to as "forgetting by interference" by some authors, implying a very general kind of interference: any new information interferes with all old information.

this can be found in ACT-R (Anderson, Reder, & Lebiére, 1994), in which the total amount of source activation (which is used purely for storage in the model) is held constant and evenly distributed among activation sources. Thus, adding a new activation source reduces the amount of activation available to all sources; essentially, new sources displace other sources.

Another form of displacement has been demonstrated by Baddeley and his colleagues as well as others (e.g. Baddeley, Eldridge, Lewis, & Thomspon, 1984; Carlson, Sullivan, & Schneider, 1989; Anderson, Reder, & Lebiére, in press): the "displacement" of processing. This was discussed earlier in the context of the development of the WM construct; as people store more information in working memory, they become slower in performing other computations. For example, people trying to remember an eight-digit number are slower to perform simple reasoning tasks than those trying to remember only a two-digit number (see Baddeley, 1986 for a review of simple tasks like this one). A more complex example of this phenomenon can be found in MacDonald, Just, and Carpenter (1993), where it was shown that increasing the storage requirements of syntactic bindings can significantly slow sentence processing. A sound computational model incorporating working memory constraints must address the displacement of both processing and storage; new items entering into working memory should reduce the retrievability of older items and slow down processing.

3. The Role of Experience

Though working memory is considered part of the cognitive architecture and thus separate from knowledge, the division is not a hard and fast one; the two interact in many ways. This is an important part of the "activated LTM" view; it is generally assumed that experience determines what is learned and thus, what is in long-term memory. As hinted at by Miller (1956) and clarified further by Chase and Simon's (1973) famous chess experiments , experience determines what a chunk is. That is, experience determines what the basic elements of LTM are, and thus what can be in working memory as activated

LTM.

The relationship between experience and working memory goes deeper than that, though. Expertise in a domain can functionally determine working memory capacity—in particular, having a great deal of experience in a particular domain can serve to hugely increase functional capacity in that domain. An excellent example of this is is a study by Chi (1978) which was a primarily a replication of the Chase and Simon (1973) study, but with a twist. The subjects in the Chi study consisted of two groups: normal non-chess playing adults and child chess experts. While the children scored lower on a digit span task, they outperformed the adults on chessboard reconstruction. This large functional increase in capacity associated with expertise is the subject of a excellent recent review by Ericsson and Kintsch (1995) who note that extensive experience in a domain can increase functional working memory capacity tenfold. As pointed out also by Anderson (1993), this is not simply a result of experts having larger chunks than novices. Even with a great deal of experience, chunk size seems to be bounded. Any computational system that is to be consistent with this evidence needs to be able to demonstrate the same kind of great increase in functional capacity.

4. Processing Speed

While there are changes associated with experience in domains in which a great deal of experience can come in to play, there are clearly general differences that arise in domains in which people have little experience. Explaining changes in cognitive performance as a function, to a large degree, of reduced working memory capacity is still somewhat unsatisfying. Even if a reduction in working memory capacity is responsible for reductions in cognitive functioning, that is only a structural account at a latent level that still begs the causal question: what causes the reduction of working memory capacity? Because working memory seems so critical in cognitive aging, there have been serious efforts to answer this question, particularly in the aging literature.

Researchers first observed differences in processing speed between younger and

older adults long ago (for reviews, see Salthouse, 1985, 1996, and Cerella, 1985). In general, older adults are slower than young adults on performing cognitive tasks. This basic phenomenon has been used to describe a wide variety of age effects and is often referred to as general slowing. While slower simple and choice reaction times are often associated with age, the answer to the question of how slowing can affect performance on tasks that do not immediately seem related to response time (e. g. text comprehension, integrative reasoning, matrix reasoning) has not always been clear. One way to link these phenomena is through working memory.

If one assumes the viewpoint described earlier in which WM is considered the active part of LTM, then, critical to the functioning of working memory would be the rate at which activation can accumulate and the rate at which activation dissipates or decays. To the extent that there are differences in rates of activation and decay, there should be differences in the functional capacity of working memory. It has been fairly well established that older adults are generally slower on even basic reaction time tasks and have lower working memory capacity, but this does not suffice as proof that slowing causes reduced working memory capacity. Several things would serve to strengthen this argument, in particular evidence that the same individuals that are slowed are also the ones that have reduced WM capacity. One way of approaching this problem is to use the hierarchical regression approach seen earlier, but this time with measures of working memory as the criterion and speed measures as the mediating variable. This is precisely what Salthouse and Babcock (1991) did in two experiments involving 460 subjects, and they did indeed find that processing speed greatly attenuated the relationship between working memory and age. This was replicated in several studies also conducted by Salthouse (1991b, 1992a). The relationship between simple cognitive speed and working memory has also been shown to hold with younger adults through the use of factor analysis techniques (Kyllonen & Christal, 1990).

Salthouse and Babcock further argued that the speed-WM relationship was due primarily to differences in rate of activation between individuals and not differences in rate

of decay. This is certainly a plausible argument, as several studies that investigated rates of decay of information from short-term memory using the Brown-Peterson task (and variants thereof) have found that rate of decay is quite stable across individuals, both within and across age groups (for a review, see Craik, 1977). Further evidence that working memory capacity is a function of rate of activation (and not decay) comes from follow-up work by Salthouse (1992a, 1994a). In the Salthouse (1992a) study, activation rate was assessed by rapidly presenting subjects first with a single digit and then with a series of operation-digit pairs (e. g. – 4, + 2). Subjects were to keep a running total and eventually report that total. What was varied was the presentation time for the stimuli. Presumably, if older subjects take longer to activate new working memory items, their threshold time for this task should be higher. As expected, it was, and this measure correlated negatively with a composite working memory measure and positively with a composite processing speed measure. Decay was also assessed with an arithmetic task in which subjects had to keep track of a set of running totals. The number of operations between the last update for a particular number and the time of recall was manipulated, and results from this measure failed to correlate with age or working memory measures. A similar procedure—varying presentation times and number of items between presentation and test—was employed in Salthouse (1994a) with qualitatively similar results: older adults had longer threshold times but the decay function was roughly the same for both young and old subjects. One promising fact is that, as far as I know, no evidence that strictly contradicts this position exists; that is, a positive correlation between working memory capacity and age or WM capacity and speed has not been found, and neither has a negative correlation between speed and working memory.

Thus, there is certainly empirical evidence that suggests that adult age differences in working memory are a result of differences in rate of activation. Certainly, though, one can argue that processing speed is no better a causal explanation of age-related changes in cognitive function than is working memory—after all, this is merely another structural reduction. Cognitive performance declines may be caused by working memory declines

which may in turn be caused by slowing, but what causes slowing? Fortunately, there may well be an answer to this question that is grounded in neurophysiology. Changes in processing speed may very well be a result of physical changes in the central nervous system, such as white matter dementia, frontal lobe damage, and the like, and reflected in ERP and P300 measures of the temporal nature of neural processing (Birren & Fisher, 1995, provides an excellent review). Thus, the processing speed/working memory account may provide the links between a behavioral and biological theory of aging.

The processing speed account of WM functioning has other advantages. It is parsimonious in that it stipulates a single and fairly easy to operationalize variable as the source of age-related changes in working memory. One of the practical advantages of this simple relationship is that it is also computationally specifiable. That is, it is possible to build computational systems that make use of this construct. Some early attempts at examining systems that realize slowing of activation have already been conducted by Salthouse (1988) and Cerella (1990). Both of these efforts, however, have been in context of "content-free" connectionist models, making them in many ways similar to the mathematical models that were popular in psychology in the 1960's. For a variety of reasons, such models are falling out of favor; the Salthouse and Cerella models have also failed to attract much attention since their publication.

5. Inhibition Deficits

It is important to note that other explanations have been offered as to the source of the working memory capacity decline associated with aging. One particular explanation has attracted a fair amount of attention of late, this explanation being based on "inhibition deficits." This position is generally associated with Hasher and Zacks (1988) and is commonly offered as an alternative to the processing speed position. According to the inhibition deficit account, the total capacity of working memory does not decline with age, but the functional capacity—the amount of working memory that is actually useful—does. This is because, as people age, their ability to inhibit task-irrelevant items declines.

Working memory thus tends to fill with irrelevant information which is confused with the task-relevant information at retrieval. This account was originally developed partly out of dissatisfaction with reduced resource accounts (Hasher & Zacks, 1988) and partly on the empirical results from Hamm's dissertation work (reported in Hamm & Hasher, 1992). In the Hamm and Hasher study, both old and young subjects read passages that initially supported more than one inference but ultimately resolved to support only one of the inferences (the "target inference"). Subjects were presented with terms and asked if the terms were consistent with their current interpretation of the passage. The major result was that older subjects responded "yes" more often to the incorrect interpretation, even after encountering the disambiguating information. That is, they were unable to successfully inhibit the inappropriate information. Hamm and Hasher (1992) maintained that this is inconsistent with general resource reduction because they found some evidence that older subjects actually maintained a larger (or at least an equal) number of alternate interpretations as compared to the younger subjects. They also maintained that older subjects' responses are not the result of a general tendency to say "yes" to all presented items because their "yes" responses were not more frequent than the younger adults on the target inferences.

Further evidence for the inhibition deficit account comes from the results of some negative priming studies, in particular Hasher, Stolzfus, Zacks, and Rypma (1991). In the negative priming paradigm, subjects search for a target item and are to ignore distractors on several trials. At some point, one of the distractor items is made a target and, if negative priming occurs, the response to the new target is slower. Hasher, et al. (1991) found negative priming for younger adults, but not for older adults and thus argued that older adults suffered from a deficit in their ability to inhibit information in working memory.

Other evidence for inhibition deficits has been offered by Hasher and Zacks (1988), though not directly supported by Hasher, Zacks, or their colleagues. These include increased Stroop interference (a result that has been found by several other researchers), an anecdotally higher incidence of task-unrelated thoughts, and an anecdotally high incidence

of personalistic intrusions in conversation. These are somewhat less than convincing arguments for several reasons. For a start, Giambra (1989) reported on the basis of retrospective reports and five laboratory studies that the frequency of task-unrelated thoughts not only did not increase with age, it reliably *decreased* with age. This is perhaps the most direct contradiction of the inhibition deficit account, though other researchers (Kliegl & Lindenberger, 1993) have found increasing intrusions in older subjects. Still, results here are at best mixed.

If Hasher and Zacks are correct and the Hamm and Hasher (1992) results generalize, older adults should have difficultly in "releasing" items they are no longer required to remember. Thus, they should be more prone to proactive interference in simple short-term memory tasks. However, there is evidence that older adults are not, in fact, more susceptible to proactive interference (Puckett & Stockburger, 1988). Casting further suspicion on the Hamm and Hasher results is a study of face recognition reported in Bartlett (1993) in which it was shown that older adults do, in fact, have a greater tendency to respond "yes" to recognition probes, whether those recognition probes had been seen or not. That is, Hamm and Hasher's findings may be a result of a general increased tendency to respond positively rather than an inhibition deficit.

The negative priming results also have problems. First, it has been demonstrated that distractor inhibition is neither necessary nor sufficient to produce negative priming (J. Park & Kanwisher, 1994) and further, that it is possible to get negative priming in older adults that is as large as the negative priming found for younger adults (Sullivan & Faust, 1993). Even if the one accepts the negative priming data presented by the Hasher and Zacks group, there are data which cast doubt about the direction of causation. It may be the case that reduced capacity causes negative priming rather than the reverse. Engle, Conway, Tuholski, and Shisler (1995) found in a sample of young adults that negative priming effects could be eliminated by increasing working memory load with a secondary task. They suggest that negative priming is a result of having additional capacity to allocate, which presumably the older subjects would lack.

Thus, the Hasher and Zacks account is at least questionable even on the central tasks that have been used to support it. One further problem with this account is that it is not at all clear what is supposed to cause the deficit in older subjects' ability to inhibit, thus leaving the ultimate cause of age-related declines just as much a mystery as before this account was proposed.

On the other hand, there is evidence that in a major class of inhibition phenomena—Stroop interference—that slowing mediates the age differences. Salthouse and Meinz (1995) found that a composite interference measure based on Stroop-type tasks did indeed mediate age differences in working memory, but less so than did speed measures. Beyond that, variance in speed measures were also found to mediate the age-related variance in inhibition. This is consistent with the Engle et al. (1995) results in that what Hasher and Zacks identify as a causal factor may well be a result of the same mechanisms that cause a reduction in working memory capacity.

Theoretically, inhibition deficits can easily be accounted for with speed measures. If one views inhibition as the propagation of negative activation and accepts the argument that the propagation of all activation is slowed with age, then one would naturally expect some form of inhibition deficit in the elderly. Rather than being an alternative to slowing, inhibition deficits may instead be a result of slowing. This is in sharp contrast with the way Hasher and Zacks typically frame the inhibition deficit hypothesis, with inhibition deficits and slowing being essentially mutually exclusive. Even if certain classes of inhibition deficits are legitimate, this argument is, at present, simply untenable.

6. Capacity vs. speed-decay

One of the interesting issues in modeling working memory is that systems such as CAPS (and also ACT-R with proposed changes, see Anderson, Reder, & Lebiére, 1994) restrict working memory by placing a ceiling on the total amount of activation available in the system. While this may yield a method for providing working memory constraints to modelers, it has the drawback of being quite arbitrary. No explanation has been offered as

to what this activation limit actually is or why it should be there, other than that people do not seem to be unlimited. This seems bizarre, since it has been demonstrated time and again that people with higher working memory capacities do better on a wide variety of cognitive tasks—so why would anyone be limited? Capacity-based systems (or at least CAPS and the simple system proposed by Kotovsky and Kushmerick, 1991) have another theoretical drawback, in that when the systems are operating below their capacity ceiling, unused working memory elements do not decay. This essentially means that a person who puts a two-digit number into working memory and then does nothing that creates much of a memory load—but still does not rehearse it or code it into LTM—should still have that two-digit number available in working memory several hours later. This is at best a questionable prediction.

A system based on processing speed and decay over time, on the other hand,would be making use of mechanisms that are more well-founded. It has been demonstrated by experimental psychologists repeatedly beginning over 30 years ago that items in temporary storage decay, yet this has by and large still not been incorporated into computational models. It is also clear that speed of processing is involved in the determination of individual differences found on working memory measures. Recent evidence (see Birren & Fisher, 1995 for a review) has also linked speed of processing to neurophysiological mechanisms. So, while the exact nature of the speed/working memory relationship may not be clear, a speed-decay mechanism certainly seems more plausible than the arbitrary activation ceiling found in many systems.

Recent empirical evidence bears on this issue as well. An elegant paper by Towse and Hitch (1995) set out to discriminate “between cognitive space and memory decay hypotheses” (p. 110) with an experimental technique by which they equated a pair of memory and counting tasks on difficulty but not execution time. A “cognitive space” (capacity) theory would predict no difference between these conditions, since tasks of equal difficulty should use the same amount of capacity. A decay-based theory (such as SPAN) predicts that if the tasks are equally difficult, then the version that simply takes

longer should yield poorer performance, which is precisely what was found by Towse and Hitch.

CHAPTER VI.

MODELING GOALS AND APPROACH

The theory of working memory to be outlined here is computational, and thus predictions using the theory will be based on models constructed with the theory. This is a conscious choice that has a tremendous impact on the form of the theory—computational theories look unlike any other kind of theory and involve their own sets of strengths and weaknesses. Thus, some justification for this decision is warranted. Why model at all? Why was a production system chosen as the framework? What is the scope of the system? What issues does the decision to model raise?

1. Why Model?

Computational models provide a number of potential advantages when combined with sound empirical methods. First, computational models provide a sufficiency account for a theoretical explanation. Often, when psychologists propose explanations for an observed phenomenon, it is not clear that the mechanisms stipulated in the explanation are actually capable of producing the observed behavior—the vagueness of verbal descriptions make them hard to carefully assess. However, if a computational model of the phenomenon can be constructed, then there is a guarantee that the proposed mechanisms can generate the behavior. If the mechanisms are not sufficient to generate the behavior, then they will not yield a working model. Some care must be taken, though, to not mistake sufficiency for necessity. A computational model provides a demonstration that the behavior *can* be produced by a particular mechanism, but not that it *must* be produced by the given mechanism—there may be many other mechanisms that could generate the same

behavior. Still, making claims about what a mechanism can do without a simulation model that demonstrates that the mechanism is indeed capable of doing so is a weaker claim than one that is supported by the existence of a working model. Without a model, sufficiency claims for a mechanism are simply that—claims. Interpretations of empirical data alone are good at suggesting that a particular kind of mechanism might produce a particular behavior (or class of behaviors), but can not demonstrate that the mechanism they suggest actually is capable of doing so.

Secondly, computational models allow certain kinds of questions about mechanisms to be addressed that would be much more difficult to address without such models. Probably most important among those questions is "what is mechanism X?" The strength of computational models in this regard is their specificity—it should be possible to point to some aspect of the model and decide if that is, in fact, X or something that can produce X. In a computational model, X has to be specified clearly or the model will not work. The specificity of description is a great advantage in computational modeling.

If it is not possible to find a particular component in a computational model, then at least one can say that the model does not offer an explanation of that mechanism. Without explicit process models, it is not always clear what researchers mean when they use terms like "central executive," "capacity," "processing resources" and whether or not any particular piece of data is relevant to those terms. A harsh evaluation of such explanations is found in Salthouse (1988, p.3): "[I]t is quite possible that interpretations relying on such nebulous constructs are only masquerading ignorance in what is essentially vacuous terminology." For example, without an explicit model, it is not clear at all what "capacity" is and what the ramifications of a change in capacity are. With a computational model, the ramifications of changes should be examinable by running multiple instantiations of the models with those changes made and examining the effects. If the change in the model's behavior is in accord with the changes observed in human behavior, this provides a form of construct validation as well as a sufficiency argument for the mechanisms present in the model.

Another problem with explanations based solely on verbal descriptions and observation is that it is very difficult to assess the internal coherence of such explanations, while such assessment is much more straightforward with computational models. Verbal theories can easily hide subtle (and not so subtle) inconsistencies that make them very poor scientific theories. Computational models, on the other hand, force explanations to have a reasonably high level of internal coherence. In addition, computational models should make clear the assumptions used in their construction, such as the form of representation used. For example, computational models are sometimes compared on the number of parameters that are pre-specified vs. the number that are fit to data. Less well-specified explanations make such comparisons virtually impossible. How is it possible to assess the coherence of a verbally-described system in which the components themselves are not specified? What would internal coherence even mean in such cases, and how could two such explanations be compared? The fact that the answers to such questions are far from clear is a compelling argument for constructing computational models.

Third, computational models allow for making specific predictions. Verbal models can often be interpreted in different ways to predict different outcomes for some task. Computational models, on the other hand, typically generate very unambiguous outcomes. Models that predict reaction times, for example, output reaction times as predictions. While that may seem tautological, verbal theories that supposedly make claims about particular phenomena often do not produce much in the way of specific predictions to which data can be compared. If one wants to figure out how a particular mechanism impacts performance on a certain task, building a computational model of the task will provide an answer. This is important, because for many of the mechanisms that have been discussed in cognitive psychology, it is not at all clear what the ramifications are for certain tasks. Falsifiability is often very difficult to assess in psychology, because many theories do not make predictions that are specific enough to falsify. Computational models do not guarantee falsifiability, but they do offer much greater hope of it. Models that make specific, quantitative predictions do more to advance a theory because their accuracy can be more

easily assessed than informal verbal explanations. Furthermore, models can be used to make predictions about tasks that have not yet been constructed. They can suggest tasks that might be interesting tests of a theory that would not otherwise be obvious. Models in psychology should suggest experiments, just as they do in other sciences.

2. Production Systems

One of the major decisions made after deciding to model is the choice of modeling framework. There are a variety of options, from the high-level AI systems based on case-based reasoning to the ostensibly neuron-like connectionist networks. The middle road in this range is occupied by production systems, collections of IF–THEN rules that act on the contents of a temporary declarative memory.[7] Why chose a production system? There are several reasons.

The first reason to choose production systems is historical. Production systems have a long history in cognitive psychology, dating back to the early production systems of Newell (1972). Few formalisms have been around this long in cognitive science; the longevity of production systems is a testament to their success. Production systems have, if anything, gained strength over the years, though some might claim they reached their heyday in the mid–1980's. Despite that claim, there are currently four major production systems in use by various researchers: Soar (Newell, 1990), ACT-R (Anderson, 1993), CAPS (Just & Carpenter, 1992), and most recently, EPIC (Kieras & Meyer, 1994). There are a number of other researchers who, while not committed to any of these four perspectives, treat production systems as one of their central formalisms (e.g. H. Simon). One of the more influential theoretical works in cognitive psychology in the past two decades is Anderson's (1983) ACT*, a precursor to ACT-R. One of the reasons for choosing production systems, then, is their track record.

[7] In production system terminology, the term "working memory" is typically used to describe this temporary declarative memory. This is confusing, given the focus of the current work on the psychological notion of working memory. Thus, the term "working memory" will *not* be used to denote the production system sense of the term.

Track record alone does not justify the choice of production systems. It is instructive to ask why production systems have been successful to get a better understanding of their strength as a theoretical framework in psychology. The present discussion will focus on the aspect of such systems that seem most centrally relevant to the examination of working memory: the level of representation and description. Production systems are attractive as psychological theories because of the "grain size" of description. A great deal of work in cognitive psychology has been concerned with memory and natural language, and the unit of analysis in much of this work has been the proposition (see, for example, Anderson & Bower, 1973). Productions seem a natural grain size to work with because the IF side of a production is typically a series of tests on a set of propositions. Thus, they can build on the body of work in language and memory. The grain size issue is particularly important when considering working memory, since propositions can fairly easily be mapped to the "chunks" construct developed in early research on working memory (see Anderson, 1976 and Anderson, 1983 for discussion of this mapping).

The production itself is also a useful unit of analysis. Productions can be cast at the level of the individual action: a keystroke, for example, could be triggered by a single production. This makes individual productions small enough units to be comprehensible without making them so small that the modeler is forced to micromanage everything in the model. Furthermore, productions are very modular representations. This is technically convenient for the modeler and has theoretical advantages as well; for example, productions used in one task can often be reused to do another. Work on the transfer of cognitive skill (e.g. Bovair, Kieras, & Polson, 1990; Singley & Anderson, 1989), has demonstrated that transfer can be modeled at roughly the production level—that is, it the basic unit of transfer of skill from one domain appears to be at least production-like. Of course, productions can be fairly large or fairly small, which makes them useful for working at different levels of analysis.

Finally, the chunk-production level of representation allows the modeler to express concepts at a high enough level to be semantically useful (e.g. "a red block") without

sacrificing connections to work at lower levels. While it is probably easier for modelers to work with larger representational units like scripts (Schank & Abelson, 1977), it is not clear how scripts relate to lower-level cognitive processes and models. This concern with lower-level processes has led to low-level modeling strategies like the original connectionist or PDP models (Rumelhardt & McClelland, 1986), which are at least on the surface more neurally plausible. Fortunately, careful analysis of the computational properties of connectionist networks and productions systems has shown that, in general, production systems can be formally expressed as connectionist networks (Tourtzky & Hinton, 1988; Dolan & Smolensky, 1989). In order to demonstrate this, most of ACT-R has been implemented as a connectionist system (Lebiere & Anderson, 1993).[8] Thus, any evidence laid down to support the "realism" of connectionist systems can be applied to production systems as well.

3. System Building

Since production systems have a stalwart history and strong presence in the field today, there seems little reason to reinvent the wheel. Thus, in constructing SPAN, a great deal has been openly borrowed—right down to large pieces of Common Lisp source code. This borrowing reflects the belief that what has gone before is valuable and important.

It will also become clear that SPAN is not nearly as complete as some of these other systems. This is also intentional. SPAN is a testbed for a theory of working memory, not a complete cognitive architecture. While SPAN is certainly intended to be a general production system, it is not intended to be a "unified theory of cognition" on the order of Soar or ACT. There are a wide variety of phenomena in cognitive psychology which SPAN does not and is not intended to address, such as learning and categorization. This is not to say that these phenomena are in some way peripheral or less important than what is addressed, but rather that such issues are currently beyond the scope of the present

[8] The connectionist version of ACT-R, called ACT-RN, did not implement every feature in ACT-R but showed how the realization of most of the features as connectionist networks is technically feasible.

research. Hopefully it will be possible in the future to extend SPAN to address a wider variety of phenomena, but for now, working memory is the central focus.

4. Caveats

Model-building is not without its pitfalls, and it is important to be up front about them. This section will discuss some of the potential difficulties faced by modelers and computational accounts in general.

4.1 Task analysis

One of the important issues in building models is task analysis, the process by which the modeler maps observable behaviors in a particular task situation to unobservable operations that are in the language of the modeling formalism. There is no magic formula for doing this mapping, yet the details of this mapping can indeed affect the behavior of models. Strangely enough, the term "task analysis" does not even appear in the index of most monographs about modeling and modeling systems, for example *Issues in Cognitive Modeling* (Aitkenhead & Slack, 1985), *Unified Theories of Cognition* (Newell, 1990), and *The Architecture of Cognition* (1983). Two pages are devoted to the subject in *Rules of the Mind* (Anderson, 1993), but little is said about how one can or should go about doing a task analysis.

This odd oversight probably stems from the fact that well-established guidelines for task analysis simply do not exist yet. The task analysis for a particular model is usually the result of one person's best guess at how the task is done, and this is by no means guaranteed to yield anything more than a candidate for how the task might be performed. This is further complicated by the fact that the subjects being modeled are not likely to approach the task with only one strategy and perform the task in a uniform way. Nothing in SPAN (or any other modeling formalism) provides a "magic bullet" that makes this indeterminacy go away. There is no automated or even generally-taught system for going from behavior to formalism.

So what can be done? While this indeterminacy is a problem, it is not an

insurmountable issue. No model of any unobservable processes (like cognition), no matter what the task analysis, could be claimed to be "the true way the task is done by people." It is a mistake to believe that task analysis must be perfect for modeling to be successful, because this assumes that claims regarding models are claims about necessity. They are not, and cannot be.[9] Given the perspective that what a model is about is sufficiency and predictiveness, and not necessity, the indeterminacy of task analysis is not all that damaging.

For example, consider two modelers who are using some production system to model behavior on a particular task. Since the task analysis process is not well-constrained, it is unlikely that they will generate identical sets of productions. There are now two alternatives for the outcomes of this process: either the two sets of productions, despite their differences, make the same predictions, or they do not. If they do, it is highly likely that, while there are differences in the appearances of the sets of productions, they are in fact computationally equivalent within the constraints of the formalism's level of predictive detail. In this case, there is no problem with indeterminacy. If, on the other hand, the two sets of productions differ in the predictions they generate, there is still hope; the predictions can be compared to empirical data and evaluated on the basis of this comparison. Task analysis is not, then, totally indeterminate—the *process* may be, but the *results* are not. An inaccurate task analysis will lead to inaccurate predictions. If the predictions are accurate, then at the bare minimum, a clear sufficiency argument has been made for the model based on that particular analysis of the task.

When a model does a poor job of predicting, though, the indeterminacy in task analysis creates uncertainty regarding blame assignment. Was the model inaccurate because the task analysis was flawed or because the theory upon which the model is based was flawed? Proponents of the theory on which the failed model is based are likely to argue the former, opponents the latter. There is no automatic way to resolve this disagreement; however, the proponents are free to propose other task analyses that do a better job. If one

[9] This, of course, has not prevented some people from making necessity claims based on models. Neither has the knowledge that affirming the consequent is a logical error stopped people from doing it.

cannot be found, the task in question represents a serious challenge to the theory, either in accuracy or scope. But challenge and falsification are key components in scientific progress.

The key to handling the indeterminacy of task analysis is empirical. For model-building to truly be a scientific endeavor, serious comparisons with laboratory results are required. Computational models are models of something, and this something is critically tied to human behavior. Modeling without serious comparisons to human data is as fruitless as mountains of such data without theory—building models is not a substitute for running empirical studies, it is a complement to running them.

4.2 Fascination and reverse anthropomorphism

The admonition in the preceding paragraph reflects the "dark side" of computational modeling. Care must be taken to ensure that concern with internal coherence does not override concerns about correspondence to empirical phenomena. While this course is common and sometimes justified in Artificial Intelligence, it is a dangerous trap for psychologists. One of the reasons this trap is so dangerous is that it is such a compelling trap—spending hours and hours perfecting any reasonably large piece of computer software carries with it an immense amount of emotional baggage. It is easy to become entranced by the perceived elegance of the finished system, *even if that system does not correspond with human cognition*. It is important not to confuse the means and the end, despite the fact that it is easy to do so. If this warning is kept in mind, however, the benefits of modeling likely outweigh the drawbacks.

Further, it is important to temper claims about human minds/brains based on computational modeling, or what might be called "reverse anthropomorphism." This is the all-too-common practice of attributing humans with the properties found in computational models. From a historical perspective, this computational metaphor may have been necessary to spark the "cognitive revolution" and generate interest in computer simulations,

but it hardly seems necessary or even desirable today.[10]

Despite the advocacy of production systems found here, there will be no claims that there are productions floating around in people's heads. SPAN is not intended to be a veridical picture of what human brains actually do. A SPAN model, like any computational simulation model, is still just a model. It makes no more sense to say that a cognitive simulation actually *is* cognition than it does to say that a computer simulation of a hurricane is actually a hurricane. Oddly, some people seem willing to make these kinds of claims for cognitive simulations when no one would make them for weather simulations. Human brains are no more digital computers than tornadoes, airplane wings, economies, or anything else simulated on digital computers. Computers can provide useful theoretical tools without necessarily providing metaphors for human thought. Critics of cognitive science (hard-line behaviorists and outraged humanists alike) are correct when they condemn these kinds of metaphors and attributions—what they often fail to recognize, though, is that the metaphors and attributions are by no means necessary for the computer to be a useful tool for investigating questions about human cognition. Simulation is not, nor is intended to be (at least in the case of SPAN and SPAN models), duplication.

[10] The computer metaphor has been called the "strong AI" stance by Searle (1992), who also provides a strong criticism of this position.

CHAPTER VII.

SPAN SPECIFICATION

SPAN is an attempt to computationally realize the mechanisms described in Chapter 5. That is, SPAN includes decay, displacement of both storage and processing, mechanisms for increasing functional capacity with experience, and an interface to an external world model. How each of these mechanisms is handled is described in the remainder of this chapter. The chapter will close with a discussion of some of the issues this specification raises.

1. General Concepts

1.1 Representation

SPAN uses two kinds of representations: declarative knowledge units (nodes, elements, or chunks) and procedural knowledge units (productions, links, or rules). The chunk-production style of representation in SPAN is nothing particularly new—it is, in fact, nearly identical to the one used in ACT-R and CAPS. It is very modular, which is important for giving the system certain kinds of "graceful degradation" properties. That is, the deletion of a single production or node will not cripple the entire system, which potentially has applications in modeling certain kinds of brain damage (though, oddly, this has not yet been much pursued). Productions are relatively small units, which seem particularly appropriate for modeling incremental skill acquisition and transfer (for a summary, see Anderson, 1993, chapters 1 and 2). The rationale behind the procedural/declarative distinction is well-explained by Anderson (1983, 1993) and is consistent with limited physiological evidence for the distinction in both normals and

amnesiacs (e.g. Squire, 1987; Cohen & Squire, 1980; Nissen, Knopman, & Schacter, 1986).

Declarative knowledge units in SPAN correspond in terms of structure to those described by Anderson (1983, 1993): propositions, images, or temporal strings. Propositions are the most common representation used and the only ones which are actually implemented in SPAN. Each element has associated with it a current activation value, which ranges continuously from -1 to +1 just as in typical PDP systems. Any element with an activation above the minimum value of -1 will experience decay over time—the case for decay has already been made. Each element also has associated with it a strength or, to use a PDP term, a bias. This is the amount by which the activation input to a node is multiplied. (More about both of these last features can be found in the "processes" section.)

Procedural knowledge units consist of pairings between input conditions and output activation, much like productions in CAPS and links in a connectionist network (the terms rule, production, and link will be used interchangeably). A link has the following general form:

$$\texttt{IF (node}_1 \texttt{ … node}_n\texttt{) THEN ACTIVATE ((target}_1\ x_1\texttt{) … (target}_m\ x_m\texttt{))}$$

If all *n* nodes in the condition (or left-hand side) have non-negative activation values, then the *m* target nodes will receive input according to the specified x value. Nodes 1 to *n* are said to propagate activation to target nodes 1 to *m*. For example, the rule:

```
IF (enemy(X) threatening(X)) THEN ACTIVATE ((goal(destroy,X) 5))
```

says that if X is an enemy and that enemy is threatening, send 5 units of activation to the goal of destroying X. The elements for "enemy(X)" and "threatening(X)" must both have activations of 0 or greater for activation to be propagated, or, in production-rule terminology, for this rule to match. Because of the serial nature of current computer hardware, and to some extent software, continuous flow of activation and pattern-matching is infeasible. Pattern matching and activation propagation is done in *cycles*, just as in any

other production system. A cycle consists of one pattern-matching phase in which all matched productions are identified and a propagation phase, in which all activation values are updated. Note that there is no "conflict resolution" in SPAN—all productions with their conditions matched "fire" on each cycle. It should also be noted that strengths may be negative, that is, inhibitory; no special status is granted to negative numbers.

1.2 Processes

This is the area in which SPAN is much more unique. In SPAN, there is a fundamental tension between decay and propagation of activation. As mentioned earlier, each node has associated with it some activation value. This activation naturally decays over time, according to an exponential decay function (consistent with various data, e.g. Melton, 1963) with a half-life of approximately 1000 ms (Baddeley, 1986).This half-life is based on Baddeley's summary of work on articulation rate; essentially, working memory items that cannot be rehearsed within two seconds are typically lost. This time course for working memory decay can be reasonably approximated with a 1000 ms half-life function.

The activation of a given node on a cycle is based on four things: its activation on the previous cycle, the amount it decays between cycles, the node's bias, and the input to the node. A node's bias is a function of frequency of use—more often-used nodes should have higher bias. As already noted, input comes from matched productions which also have associated with them strengths, in much the same spirit as the strengths found in ACT-R productions (Anderson, 1993). The basic input signal to a node is the simple sum of the activations propagated to that node by productions multiplied by a rate parameter.

The rate parameter represents the rate at which an individual can propagate activation—it co-determines (along with strengths of particular productions) the amount of activation that can be propagated per unit time. It is assumed that the rate parameter is global in that it applies to the activation propagated by all productions. This parameter is also assumed to be the main source of individual differences in cognitive architecture—that is, different people will have different rate parameters. It is further assumed that this parameter is the central factor in reductions of working memory capacity associated with

aging; that is, as people age, this parameter gets smaller. This rate parameter is perhaps the key difference between SPAN and other approaches to handling working memory—the limited resource in SPAN is time, not activation. While there are no limits to the total amount of activation in the system, memory elements that do not receive activation rapidly enough are lost because of decay. This makes time the critical commodity and makes basic activation rate one of the key determiners of system performance.

There is no true notion of "capacity" built into SPAN—capacity is functionally determined by the dynamic between decay and propagation of activation. If the decay function is uniform and a slower propagation rate applies, the system will functionally have a smaller capacity. As the global rate parameter gets smaller, the ability of productions to overcome decay is weakened, leading to longer latencies and potentially higher error frequencies as a result of the functional reduction in capacity.

Another important property of human working memory is displacement or interference, which occurs as a result of "noise" in SPAN. Noise is conceptualized at the workload or strain on the system. The effect of an input signal to a node is reduced as a function of the total activation noise in the system at any given time. Noise is what prevents SPAN from simply activating everything in declarative memory. As more and more elements get activated to around threshold levels, the slower SPAN can propagate activation because the signal:noise ratio will decrease. This means that productions have to fire more often to propagate the same amount of activation, but more time means that each element will decay more. Thus, noise results in both the slowing of processing and, as a result makes it more likely that older elements will be lost—what might be called interference or displacement. Again, there is some preliminary neurophysiological evidence based on P300 measures that increased workload is associated with slowing (Fowler, 1994).

The total noise is based on the sum of the noise generated by all of the individual nodes. The noise function at an individual node is driven by two factors: very small activations are associated with little noise, and very large activations are associated with

little noise. The first of these is relatively intuitive; for tasks like reading, a relatively large number of items have at least some activation, but this does not swamp the system. The second property is somewhat less obvious, and, besides decay, is one of the major differences between SPAN and CAPS. The idea is that very high node activation is produced by strong rules applied to high-bias elements. Processes that make use of such rules and nodes are going to be highly practiced, skilled processes, which tend to be relatively load-insensitive and also tend to interfere with other processing much less (see section 5.3). Thus, chunks that are close to threshold should generate the most noise, while those far above and far below should generate relatively little noise.

What are the functional implications of such a system? As already discussed there is no hard and fast limit on the total amount of activation that can be present in SPAN, but SPAN's behavior will closely resemble that of a capacity-constrained system—it will appear to have a limit. Decay and propagation are fundamentally in conflict, and anything that slows propagation makes it harder to overcome decay. Thus, when the system is loaded (which slows it), performance degrades—as if there were a limit. This limit will, in general, be a function of cognitive speed. However, expertise in a particular area can offset this—a slow individual may be able to outperform a fast individual if the slower person has stronger and/or better structured rules and chunks. (Again, see section 5.3.)

SPAN has no box which could be pointed to and labeled "working memory." Certainly, there are some important distinctions to be made among memory elements: those that are above threshold, those that are active but below threshold, and those that are inactive. The subset of elements above threshold determines which links will be active (which productions will fire), and are thus the most critical elements at any given moment. The active but sub-threshold elements are important in that they contribute to noise and are the source of priming. Since SPAN has no special "working memory" system, the term working memory will be applied to phenomena of temporary processing and storage. (This is not unlike Atkinson & Shiffrin's relegation of the term "short-term memory" to a description of phenomena and not cognitive structure.)

1.3 External memory

One of the features that distinguishes SPAN from other production systems such as Soar is the relatively rapid decay of memory elements. However, it seems obvious that not all of the information immediately available to the human cognitive system (i.e. not everything that productions can match) consists of temporarily-active internal representations. In particular, the external world can be used to "store" information, and the behavior of the cognitive system is clearly affected by this capability. As a brief example, consider the task of multiplying two ten-digit numbers without the use of any form of external memory aid. This task is next to impossible for most people without some form of external memory aid (e.g. paper, a chalkboard), while with such a memory aid the task is merely tedious. However, much less is known about how external memory interacts with the cognitive system than is known about mechanisms of working memory; thus, this is much more tentative than the previous sections.

In order to reflect this, SPAN has a separate memory "area" which will be referred to as "external memory." External memory consists of declarative memory elements, but these elements have no activation value associated with them. Logically, then, they do not decay or contribute noise. These elements are all available to be matched by productions, but cannot be acted upon directly by productions. The role of perception in SPAN is, in essence, to "translate" continuous sensory data into discrete symbols which can be recognized by the production pattern-matcher.[11]

There are two things which determine the input to the perceptual processor: the actual state of the perceived world and attention. Obviously, it is impossible to know what is perceptible without knowing what is present in the environment. Further, it is assumed that it is impossible for the perceptual processor to extract every possible piece of information from the environment—it is difficult to see both the general layout of the forest and the patterns on the bark of an individual tree at the same time. Attention in SPAN is defined as the portion of the environment which is being translated by the perceptual

[11] How this is actually accomplished is far beyond the scope of SPAN; as with most cognitive models, this is just assumed to happen.

processor. This is a “window”-style conceptualization of attention. While productions cannot directly change external memory, they can send commands to the perceptual processor which alter the location and resolution of the window. Exact details about the nature of the window have yet to be worked out, but similar and reasonably effective visual attention mechanisms for production systems have been proposed (e.g. Weismeyer, 1992; Anderson, Matessa, & Douglass 1995; Kieras & Meyer, 1994).

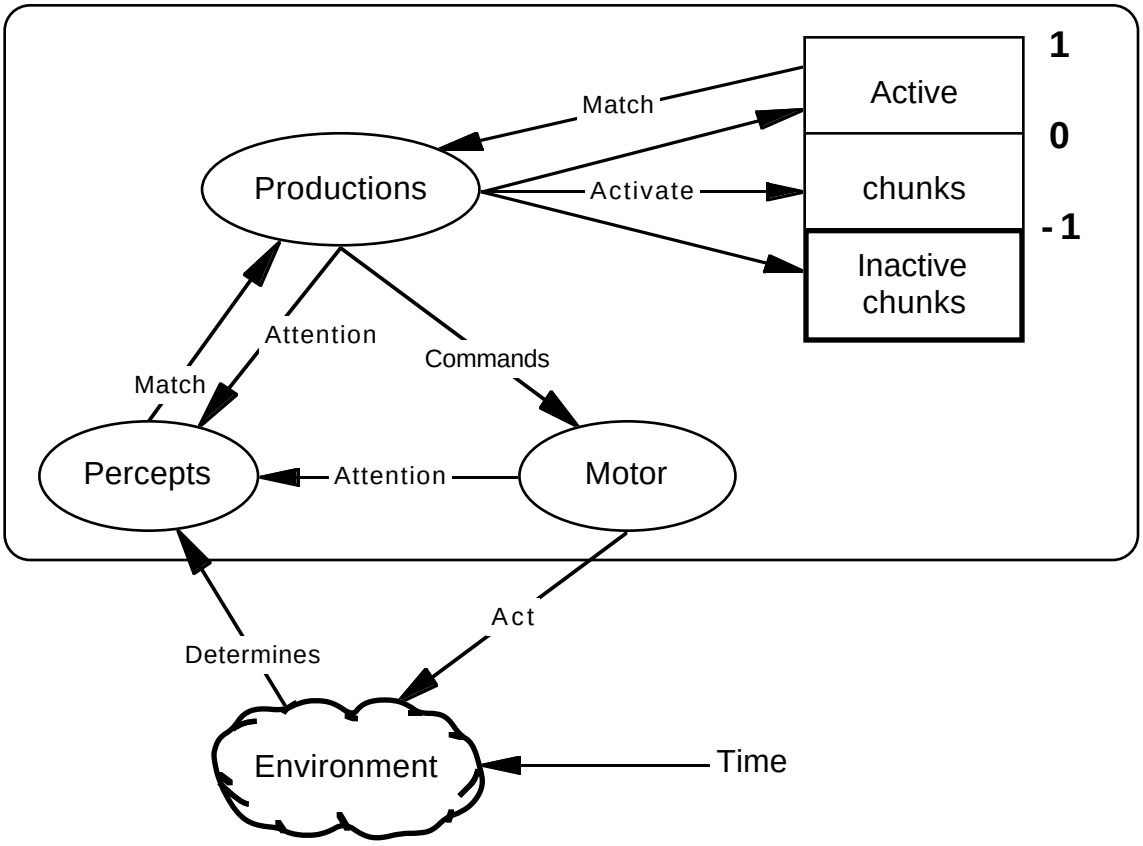

Figure 7-1 Information Flow in SPAN

How this plays out in terms of information flow is depicted in Figure 7-1. Productions can match both internal declarative elements with activations over 0 and elements that are currently available in external memory. Firing of productions can

propagate activation among declarative elements and send perceptual/motor commands. The state of the external world model is determined by motor commands and time; this world model determines what is available for perception to provide as external memory. It should be noted that many of these ideas are not new in that I have grafted a similar system on to CAPS (Byrne, 1994). However, for a number of technical reasons, CAPS is not particularly suited for this endeavor, whereas SPAN has been designed to support it. Of course, perception and action as outlined here are somewhat crude and do not do justice to the current state-of-the-art, represented by EPIC (Kieras & Meyer, 1994). In building models, model-specific approximations of the EPIC parameters will be required to get accurate timing estimates. This is an unfortunate situation, but it may be remedied with later integration work (see Chapter 12 for more information).

2. Implementation

This subsection describes the actual equations which drive the system. While the basic concepts underlying SPAN's memory activation management functions are conceptually simple, they must be completely specified in order for SPAN to be implemented. Since SPAN has been implemented, these specifications can be provided. The following notational conventions will be employed:

$a_{i,t}$	the activation of declarative element *i* at cycle *t*
Ω_t	total noise at cycle *t*
$\delta()$	decay function applied to an element
β_i	bias of declarative element *i*
γ	global rate parameter
N_e	total number of declarative elements (nodes) in the system
$p_{n,i}$	input of production n to declarative element *i*
$N_{p,i}$	number of productions with inputs to declarative element *i*

First, there is the decay function. Each declarative memory element in SPAN has associated with it a real activation value ranging from -1 to 1. In order for a production to match an element, the activation of that element must be non-negative. If an element's activation is above -1, it will decay according to the following function, which yields a half-life of approximately 1000 ms (Baddeley, 1986):

$$f(t) = 2.3e^{-.0007t} - 1.3 \qquad [7.1]$$

Note that this function would imply decay past -1, so values less that -1 are set to -1 and the element does not decay further.The function used for decay is assumed to be fixed across individuals. Though it is assumed that time is continuous, SPAN still functions in discrete cycles so the decay function is applied once per cycle to all elements. Cycle time is by default set to 70 ms in accord with the cycle time of the MHP (Card, Moran, & Newell, 1983), but the cycle time can be set by the programmer to be of any duration without throwing off the decay function.

The second necessary piece is the noise function. The underlying idea, as described previously, is that highly active elements (those with activation close to 1) should produce little noise, since such elements are likely to arise only as a result of strong productions targeting high-bias elements—"automatized" knowledge. Conversely, elements with very low activation values (close to -1) should also generate very small amounts of noise, since simple spreading activation productions should constantly be creating many such elements (this is the source of priming). One way to do this is by squaring the activation of each element and taking the difference from 1. This way, an element at either 1 or -1 activation contributes zero noise and an element at 0 activation (which is just at threshold) contributes 1 unit of noise. Total noise is simply the sum of the noise associated with each individual element, except that this leaves noise in activation squared units. Thus, the square root of this total is taken, yielding Equation 2:

$$\Omega_t = \sqrt{\sum_{i=1}^{N_e} (1 - |a_{i,t}|^2)} \qquad [7.2]$$

The final SPAN equation determines the net activation of an element at a particular time, taking into account decay, element bias, global propagation rate, production input, and noise. The amount of activation left over after decay during one cycle is a simple application of Equation 1. Added to the remaining activation is the signal (input from productions times the propagation rate) to noise ratio multiplied by the node's bias. This yields the following equation:

$$a_{i,t+1} = \delta(a_{i,t}) + \beta_i \frac{\gamma \sum_{n=1}^{N_{p,i}} p_{n,i}}{\Omega_t} \qquad [7.3]^{12}$$

One other implementation point concerns goals. In other systems (e.g. Soar and ACT-R), goals are a privileged class of working memory elements and are handled differently than other elements. In SPAN, this is not the case—goals are elements which decay and can be lost just like any other element. Relative to other systems, goals are special in SPAN exactly because they are *not* treated as different by SPAN. This should make it possible to model errors and increased task difficulty associated with goal loss (for examples, see Baars, 1992).

Currently, the programmer interface for specifying interaction with an external world is somewhat crude, but is more serviceable than the interface grafted on to CAPS (Byrne, 1994). One likely possibility for future implementations is to include the perceptual-motor system which is part of the human performance model EPIC (Kieras, Wood, & Meyer, 1995; Kieras & Meyer, 1994).

[12] All of these equations have been put together with a production rule compiler and implemented in Common Lisp. SPAN should run in any Steele (1990) compliant Common Lisp environment, though it has only been tested under LispWorks and MCL implementations of Common Lisp.

3. Other Issues

3.1 Goal management

Some words regarding goal management are appropriate here. As previously noted, goals are declarative memory elements that decay and generate noise, just like all others. This means that, over the course of executing a task, goals will tend to disappear, even if the task context does not change. Further, there is indeed evidence that supports the idea of hierarchical goal management that is at least stack-like (e.g. Newell & Simon, 1972; Anderson, 1993). These two views of goals seem somewhat at odds with one another, so finding some middle ground is probably the most appropriate initial approach.

SPAN takes a two-pronged approach to solving this dilemma. First, all goals keep track of their "parent" goal, so that a hierarchy is maintained. Second, goals depend on their parents for activation. That is, as long as a goal is active and above threshold, a production will fire giving its children activation. This prevents inordinate goal loss while at the same time preserving hierarchy and dependence on activation. This is identical to the goal management scheme presented for the CAPS model in Byrne and Bovair (in press).

3.2 Learning

While learning is not the primary focus of SPAN,[13] it is still important to consider the effects of learning on SPAN's behavior. In general, learning in SPAN ought to produce two kinds of effects: new structure and strengthening of old structures. New structures would be new declarative representation (new nodes) and new productions. The creation of new nodes corresponds to the learning of new declarative facts. Building new productions is essentially a kind of speedup learning, akin to knowledge compilation in ACT and chunking in Soar. The effects of new productions should be similar to the effects seen in these systems, with some additional benefit. To the extent that proceduralization reduces the number of declarative nodes that must be active, new productions will make the system more efficient because it will be less noisy. The reliance on fewer active nodes

[13] Nor is learning the primary focus of CAPS, just as working memory is not a primary focus of Soar and problem-solving is generally not the focus of PDP models.

also makes a compiled procedure less susceptible to decay of those nodes, so a compiled procedure should be more robust to working memory load and interruption.

Strengthening has been discussed in the context of bias and noise. Essentially, strengthening is an efficiency improvement. Stronger nodes reach threshold faster, which should yield content effects within classes of stimuli. This is consistent with phenomena such as the faster naming and lexical decision associated with high-frequency words as opposed to low-frequency words. Stronger rules produce more rapid and efficient processes, which is similar to but not as dramatic as proceduralization in its effects. ("Automatic" processes are conceptualized as very strong well-compiled procedures.)

Again, a note about external representation is relevant here. Because of the performance penalty associated with a great deal of declarative representation, SPAN also predicts that novices should be particularly susceptible to working memory loads, and that one of the things people might try to learn is new strategies that externalize representation, effectively reducing the working memory load. This is precisely what Beach (1978) found in his study of novice and expert bartenders: novices attempted to hold drink orders in working memory and were highly susceptible to memory load, while experts constructed external cues so they could reduce the demand placed on their working memories. Learning how to externalize representations would perhaps be a more well-studied phenomenon except that such strategies are not well-supported by a laboratory environment.

3.3 Inhibition

Unlike in CAPS, in the current implementation of SPAN there is no distinction between positive and negative activation—productions can propagate negative activation just as well as positive activation. No special status is given to productions with negative strength, simply due to Occam's razor—adding extra mechanisms did not seem justified.

If negative activation is thought of as inhibition, then two qualitative predictions about inhibition can be derived from SPAN: slower individuals should also be less effective inhibitors and that working memory load should also reduce inhibition, much the

same as memory load reduces positive propagation of activation. Despite the fact that I had no intention of worrying about inhibition when I designed SPAN, there is preliminary evidence which bears out both of these predictions. Salthouse and Meinz (1995) have shown a correlation between processing speed and Stroop interference, supporting the first prediction. Engle, et al. (1995) have shown that negative priming can be essentially eliminated with the imposition of a memory load. While these two studies do not prove that SPAN's treatment of inhibition is correct, they do indicate that at least SPAN is not headed in a direction that is clearly wrong.

3.4 Alternate views

There are other views of working memory which are not entirely in accord with the perspective presented here. Most notably, the present view treats WM as unitary—each person has a single working memory. This is in contrast to the view of Baddeley (1986), whose WM construct is somewhat different. For Baddeley, working memory is the "simultaneous processing and storage of information" and is distributed across three systems: the central executive and its two slaves, the articulatory loop and the visuospatial sketchpad. While the WM construct advocated here is in agreement with Baddeley that both storage and processing play a role, the three-part division of working memory is another story. While Baddeley goes into considerable detail about the articulatory loop and its properties and somewhat less into the visuospatial sketchpad, the central executive is left as essentially a black box that may or may not be connected in any way to long-term memory.

While Baddeley's framework has proven very productive in terms of generating interesting experiments, it is not well-suited to computational modeling. This is due primarily to a lack of specificity. It is difficult to even begin constructing a computational model of working memory based simply on "simultaneous processing and storage of information" and controlled by a "central executive" without guidance on the properties of these constructs. Due to the lack of specificity, Baddeley's framework is often agnostic about empirical results—any number of results may or may not be consistent with the

"central executive." This is not to say that Baddeley's work is without merit; quite the contrary, Baddeley's empirical contributions have been tremendous. His framework simply lacks specificity in areas important for making certain kinds of empirical claims and for constructing computational models.

At the other extreme in specificity is Schneider's (Schneider & Detwiler, 1987; Schneider & Oliver, 1991; Schneider, 1993) connectionist/control architecture for working memory. WM in Schneider's system is a distributed collection of very small, very specific temporary buffers and an elaborate message-passing controller. This system is explicitly non-unitary and quite complex.

While it is probably possible to derive specific predictions from Schneider's theory, it is probably easier in most cases to run an empirical investigation than it is to generate such a prediction. This is not necessarily because there is anything fundamentally wrong with Schneider's theory; it is a question of level of analysis. Schneider's theory is very low-level and makes constructing models of high-level activity a very arduous task. One of the salient properties of a model is that it should abstract some of the details, and Schneider's system in many cases is more complicated than the data to which it might be applied.

3.5 Task analysis

As discussed in Chapter 6, task analysis is not determinate, and SPAN represents no exception to this rule. However, all of the models so far constructed with SPAN do share common themes in the the task analysis used to construct them. The task analysis employed in generating these models is intentionally minimalist. That is, the models are designed to do the bare minimum in order to complete the task. A quick trip to the Appendices will demonstrate that these models are, on the whole, fairly small and simple. This was done to minimize the number of potentially performance-affecting judgment calls made by the modeler and let the system architecture determine, as much as possible, each model's performance.

This does not mean that all judgments calls can be eliminated. In particular, the

strategy for goal management impacts model performance because that can determine how many goals must be active. The general strategy adopted here is also intentionally simple-minded. First, tasks should in general be broken down into subgoals whenever there is more than one state change necessary for the goal to be satisfied. (A "state change," for instance, might be the setting of a particular control or the retrieval of a particular memory item.) The main reason for this is that without subgoaling, the IF sides of productions can get quite cumbersome as they have to test for the state of a large number of items. Subgoaling dramatically reduces the number of tests. Second, subgoals should be performed in sequence rather than in parallel. This tends to reduce overall memory load and allows dependencies to be satisfied before moving onward.

This is not a particularly comprehensive set of guidelines, but they do serve to give all of the models more or less the same "feel" at the production level. This kind of minimalist task analysis is probably not adequate for large, complex problem-solving tasks like the now-famous cryptarithmetic problems of Newell and Simon (1972). Such tasks take on the order of one to ten minutes to execute, whereas the tasks that have been modeled with SPAN take on the order of a one to ten seconds to complete. Simple task analysis seems appropriate for simple and rapid tasks. The relatively simple and rapid tasks to which SPAN has been applied are described in much greater detail in the following three chapters.

CHAPTER VIII.

A SPAN MODEL OF FAN EFFECT

In this chapter, the first real model of human data constructed with SPAN is presented. Fan effect was chosen as the first benchmark task for SPAN for a number of reasons. First, a great deal is known about the empirical phenomenon of fan effect, so there is a solid database and well-established methodology for investigating it. Second, the only other running computational model of fan effect that has been prominently presented is an ACT-R model (Anderson, 1993), so despite the task's solid empirical record, it is not one that has been approached by many computational systems. This is also a task for which it has been shown there are individual differences relevant to SPAN, so it makes an ideal first task for SPAN. If it is possible to successfully model fan effect with SPAN, that is a good indication that the basic equations underlying SPAN are on the right track.

1. The Task

One of the primary "spreading activation" measures for the last twenty years (Anderson, 1974) has been fan effect, or the amount that recognition responses are slowed by the interconnectedness of the terms involved. Subjects in this paradigm study a number of simple sentences such as "the doctor is in the park" and "the hippie is in the theater." Subjects are then shown sentences involving the same actors and locations that appeared in the study set and asked whether they had studied the sentences or not. Response time is consistently (e.g. Anderson 1974, 1983; Anderson & Reder, 1980) a function of the number of sentences in which the actor and location appear—as the total number of appearances goes up, so does the response time.

Fan stimuli are typically represented by digit pairs where each digit represents the number of studied sentences in which the actor and location in the target sentence appeared. For example, the code 2-3 means that subjects saw the target actor in two of the studied sentences and the target location in three of the studied sentences. Through a composite of a number of fan experiments, Anderson (1974) estimated the average difference between the 1-1 condition and the 3-3 condition to be approximately 250 ms. The standard story explaining this effect (Anderson, 1983) is that because activation is split between all the connected propositions, the response is slower.

An account based on slowed propagation would also predict a reduced rate of spreading activation. However, some researchers (such as Duchek & Balota, 1993) have argued that older adults do not have slower spreading activation because there is no difference between young and old in measures of semantic priming, an alternate way of attempting to assess spreading activation.[14] What is curious about this argument is that it relies on a "no difference" result but reports neither power or reliability, the latter of which is a notorious problem for difference scores. Yet if one examines the actual data they presented, there is a slightly greater reduction in RT for older subjects (which would be consistent with a proportional slowdown since older RT's are larger), the difference simply is not statistically significant. Duchek and Balota's claim of "no difference" when there is an actual difference in means without reporting power or reliability is hardly convincing. This is not a terribly damaging case, particularly in light of evidence for spreading activation differences in other tasks.

On the other hand, if it is the case that older subjects have particularly slowed memory activation, they should be especially prone to the fan effect. This result is exactly what was found by Gerard, et al. (1991). The question remains, though, as to whether this susceptibility to fan is related to working memory capacity. If it is, this makes the case stronger for rate of activation being responsible for differences in working memory. In a recent experiment conducted by Cantor and Engle (1993) with young adults, a link

[14] Even the measure is somewhat contentious, though—there is not clear agreement on what it is that semantic priming actually measures.

between working memory capacity and fan effect was found: subjects with lower WM spans were indeed more prone to fan effects.

2. The SPAN Model

At a qualitative level, an increase in fan associated with reduced WM capacity (i.e. slowing) is what any speed-decay-noise theory would predict. However, SPAN makes it possible to make these qualitative predictions more quantitative. A model of the fan effect task was constructed using the ACT-R model of fan (Anderson, 1993) as a guide (source for the model appears in Appendix A). The SPAN model begins with an external representation of the sentence and two "garbage" elements representing leftovers from previous trials. Also, since the exact goal activation time course during the inter-trial interval is not known, the four goal elements necessary for the task were also set to random values at the beginning of each run. Because these random factors produce some variation in the response time of the model, the model was run 20 times in each condition and the mean taken.

In creating a response, the model first performs a very minimal parse of the sentence. The model then performs lexical access on the subject and location by spreading activation (by productions) from the representation of the parse to initially inactive semantic representations of the words. When the semantic representations reach threshold, they cause productions to fire that propagate activation to declarative memory elements representing the studied facts, which are initially inactive. The productions which spread activation from the representation of the sentence to the representation of the studied facts are the key productions in this model, and have the following general form:

IF the goal is to recognize the sentence, and
the representation of the sentence contains a subject X, and
X has a semantic entry, and
no studied facts are above threshold
THEN transmit activation to facts with subject X

There is a corresponding production which fires for the location as well. Because there is no restriction on parallelism in SPAN, this pair of productions can fire many times in parallel (three times each per cycle for the subject-retrieving production and the location-retrieving production in the 3-3 condition).

When a fact reaches the activation threshold, it is compared to the representation of the target sentence. If the fact matches, the model generates a positive response; otherwise, it generates a negative response. Because SPAN allows strengths to be set on individual productions and declarative units, a large number of degrees of freedom is potentially available. In order to counter this, uniform weights (1.0) were used on all productions and declarative units, with two exceptions: the productions that maintain subgoals and the productions which perform lexical access. It is assumed that these processes are highly practiced so the strengths of these productions were set 50% higher than the others.

The model gets slower when higher levels of fan because several facts gradually become active, which increases the total noise in the system. The increase in noise causes a slowdown, not only of fact retrieval, but slowdown as goals are lost and need to be refreshed. A more sophisticated model might include the potential for errors as a result of goal loss; however, subjects typically make few errors in these experiments so it is not clear that such an approach would be realistic.

The criterion data set that was modeled is the data collected by Gerard, et al. (1991) because their procedure generated means that show both the desired fan effect and also included young vs. old groups, showing a slowing effect with age. In addition, these data were used to argue for the inhibition deficit hypothesis, and a SPAN model of these data would make it clear that an inhibition deficit is not be necessary to explain the data.

The first goal was to model the young subjects; to do so, the gamma (propagation rate) parameter was set to 1.0. The initial model was constructed and used uniform strengths for all the facts and fact-retrieval productions. Latencies were derived from the model by adding one “perceptual” cycle time (taken from Card, Moran, and Newell’s MHP) and one “motor” cycle time (also taken from the MHP) to the total latency of the

model run, which is simply the number of cycles multiplied by the default cycle time. The fit to the data was quite reasonable, though the model was slightly faster than the subjects, as seen in Figure 8-1. It was hoped that it would be possible to model the difference between the young and old groups with a single change to the model, a change in the gamma parameter. Again, this produced results close to the observed means (see Figure 8-1).

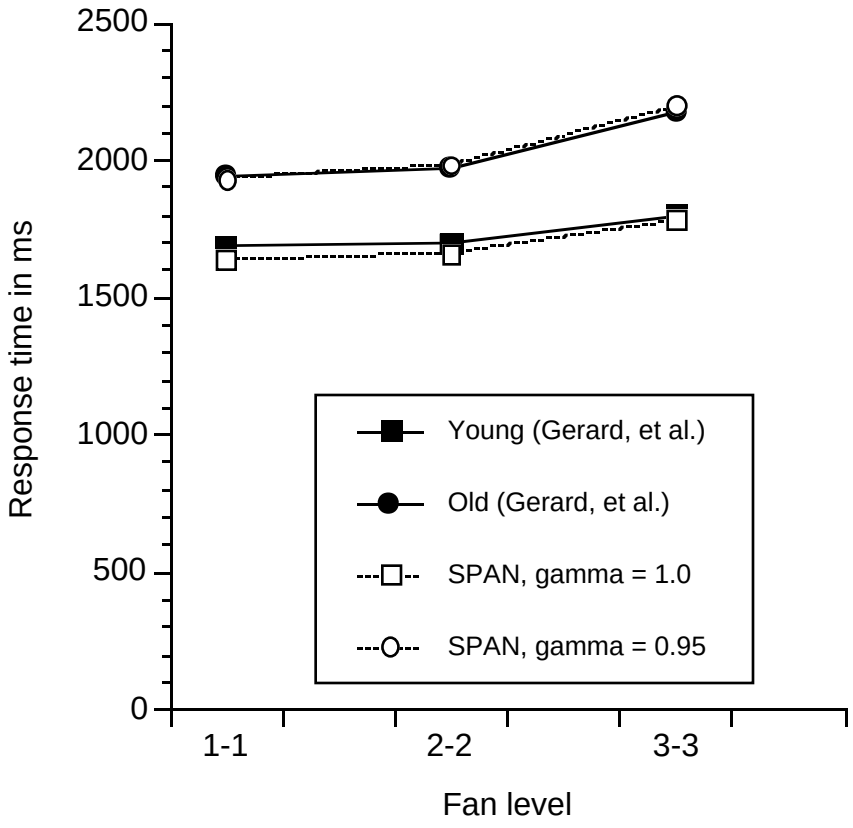

Figure 8-1. Fan effect means

The SPAN model reproduced both of the desired effects: an increased response time for higher fan levels and a slower response for older subjects. Note that the responses of the older subjects were more than 5% slower than that of the young subjects. For a fan

level of 1-1, they were 15% slower, while for a fan level of 3-3, they were 21% slower—and the SPAN model produced similar results. This illustrates an important point about SPAN and simplistic interpretations of general slowing: a reduction of global propagation rate does not necessarily imply a reduction in response time of the same magnitude—task demands play an important role in the determination of how a rate reduction affects response times.

CHAPTER IX.

A SPAN MODEL OF THE DIGIT SYMBOL TASK

This chapter describes a second application of SPAN, this time to a task that has played a key role in our understanding of the relationship between working memory and speed: the Digit Symbol task, as administered by Salthouse and colleagues (e.g. Salthouse & Coon, 1994; Salthouse & Babcock, 1991; Salthouse, 1992a). In order to understand the model, it is necessary to understand the task faced by the subjects, so this will be described first. This will be followed by some commentary on the importance of the task. Finally, the model of the task will be described and the results discussed.

1. The Digit Symbol Task

The Digit Symbol task used by Salthouse is a choice reaction time task based on the paper-and-pencil Digit Symbol Substitution task in the WAIS-R (Wecshler, 1981). In Salthouse's computer-administered Digit Symbol task, subjects are presented with a digit-symbol pair, called the target, and a code table (see Figure 9-1). If the target symbol matches the symbol associated with the same digit in the code table (the corresponding entry in the code table will be called the "code"), subjects are to respond positively by pressing the slash ("/") key on a computer keyboard. If the two symbols do not match, subjects are to respond negatively by pressing the "z" key. The measure of interest is response time, as all subjects are encouraged to—and typically do—maintain high (90%+) accuracy. In the Salthouse studies mentioned above, subjects typically perform 18 practice trials and then 90 test trials and the median time is used as the score for each subject. For half of the trials the target and code match, in half they do not. Each digit is used as the

target digit 10 times, and the digit-symbol pairings in the code table do not change over the course of the experiment.

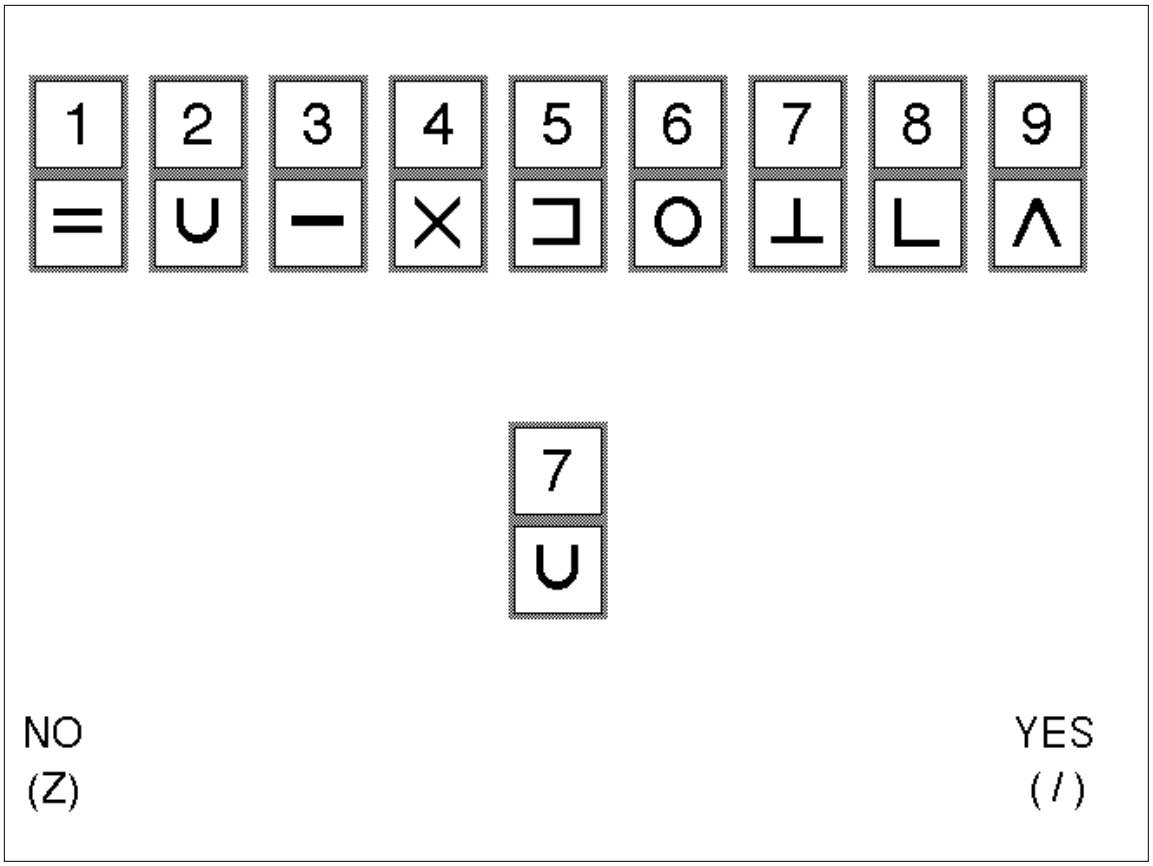

Figure 9-1 Screen shot of standard Digit Symbol task

The reference data for the model comes from Salthouse and Coon (1994) Study 2, which consisted of 40 young and 40 older adults. The mean score for the young group was 1039 ms and for the older group, 1754 ms. This is a considerable difference; the older subjects were nearly 70% slower than their younger counterparts—the young and old distributions did not even overlap. These differences are important because differences in scores on the Digit Symbol task have proven to be an excellent predictor of individual

differences in a wide range of cognitive tasks, particularly working memory measures (e. g. Salthouse, 1991b). The Digit Symbol task is one of the primary tasks upon which Salthouse's results on speed, working memory, and aging are based; it is one of the standard "speed" measures.

It is clear that older adults are generally slower to perform the Digit Symbol task than younger adults and that this slowing is predictive of decrements in other areas. However, a clear analysis of exactly what it is that older adults do more slowly has been missing. The goals of this modeling effort are: (1) To demonstrate that slowed memory activation can account for the observed differences, which would provide a computational sufficiency account of the phenomenon—this has been missing up to now. (2) To point out what it is that the older subjects do more slowly. It is hoped that the model would indicate where we should start looking in trying to answer the question of why this task is a good predictor of aging and memory effects.

2. The Model

The task itself is obviously a simple one, and thus the models of the task are also quite simple. The basic operations the task requires is the comparison of two symbols, one associated with the target and one associated with the corresponding digit in the code table. The model, then, has as its basic requirements making representations of the two symbols simultaneously available for comparison. The simplest way to accomplish this is to have one symbol in memory and one visually available, which is what the model attempts to do.

2.1 Assumptions

Time is the key to understanding the Digit Symbol task, since the dependent measure is not a measure of performance quality such as number correct, but of performance latency. A long latency in this task is two seconds, so the model is necessarily a low-level model. Thus, a great deal of explicitness regarding the time constants used in the model is called for. The time (in milliseconds) for a variety of operations in the model

is presented in Table 9-1. These time constants are borrowed almost entirely from the current state-of-the-art in modeling low-level timing constraints, EPIC (e.g. Kieras & Meyer, 1994).[15] While they may not be exact, they represent our current best approximation.

Table 9-1 Operator times used in the Digit Symbol Model

Operation	**Time**
Production cycle	50
Eye movement (compute location and move)	200
Extract symbol from visual buffer	250
Issue motor command	70
Keystroke	100

There are several other assumptions made in constructing this model. First, some assumptions are made about the state of working memory at the beginning of each trial: it is assumed that the system begins a trial with the main goal already active at some random above-threshold value in working memory and that there is some "garbage" element in working memory that is left over from a previous trial, also with a randomly-determined activation value.

Representations in the model are very atomic—it is assumed that each of the symbols in the table can be represented with a single declarative memory element. Further, it is assumed that each symbol has some kind of verbal code that can be constructed so that the symbols can be rehearsed. (Rehearsal will be discussed in greater detail in Chapter 10). Activation parameters also affect the system's ability to construct representations, so those were generally fixed. All declarative elements in the model had the default bias of 1.0 and all the productions had a strength of 1.0—with one exception. This exception is the

[15] The eye-movement time is a cruder estimate than is typically used in EPIC, where degrees of visual angle covered by an eye movement are used to help determine movement time. The 200 ms approximation appears to work for general eye movements around a computer screen and has been used by Anderson and his colleagues working with ACT-R (e.g. Anderson, Matessa, & Douglass, 1995).

production which creates an internal declarative element that is a copy of something in the visual buffer. It is assumed that this production is run fairly often so its strength was set slightly above the default of 1.0 to 1.3. This is a reasonable assumption, since the model can rarely make the baseline times shown for Model I if this strength is set lower. Young subjects were modeled with a global propagation rate (gamma) of 1.0 and it was hoped that older subjects could be modeled by adjusting this downward.

The model also assumes that there is a production that maps a positive or negative comparison directly to a motor action. That is, when the model matches the target symbol with the code symbol, a production immediately takes care of generating the appropriate motor action—the subject does not either retain the mapping of "match" to the "/" key in working memory or retrieve it from long-term memory. It may be reasonable to assume that such a production can be built in the 18 practice trials. This assumption is somewhat trickier than the rest and will be discussed further in the discussion section at the end of the chapter.

2.2 Model I: The single-look model

The initial attempt to model this task was primarily an attempt to construct a model that met two criteria: (1) the model was as simple as possible, and (2) the model performed the task as quickly as possible. In terms of simplicity, the goal was to construct a model that would perform the absolute minimum number of operations possible to perform the task. This model (productions can be found in Appendix B) performed the task as follows:

- It begins the trial looking at the target.
- A memory representation of the target symbol is then constructed.
- The eyes are then moved to the code based on the code number.
- The code symbol (in the visual buffer) is compared to the memory representation of the target symbol.

This model should parallel the run-time for a model constructed in CPM-GOMS (Gray, John, & Atwood, 1993) or EPIC, since it uses the same kinds of time constraints. This run-time based on this analysis should produce a response in 1020 ms, as follows:

- 250 ms to extract the target symbol
- 50 ms (one cognition cycle) to generate an internal symbol for the target
- 50 ms (one cognition cycle) to initiate the eye movement
- 200 ms to make the eye movement
- 250 ms to extract the code symbol
- 50 ms (one cognition cycle) to compare the target and code symbols
- 70 ms (one motor processor cycle) to run the keypress
- 100 ms to execute the keypress

The time of 1020 ms is within 19 ms of the average time for the younger subjects, so this very simple analysis of the minimum operations necessary to perform the task immediately puts us in the right ballpark. On the other hand, this analysis suggests that simple reductions in the efficiency of the cognitive operations might not be enough to produce a large decrement in performance; there are only three cognitive operations that figure into the total time. If each of these took even twice as long, this would only add 150 ms to the latency, which hardly approaches the 715 ms difference between young and old subjects found in the Salthouse and Coon study. This might seem to suggest that something more complex than simple slowing is going on.

2.3 Results of Model I

With a propagation-rate parameter (gamma) of 1.0, the model sometimes does correctly perform the task in the expected 1020 ms. However, even with this high propagation rate, the model cannot always successfully perform the task. Why? This happens when the model cannot execute the "compare the target and code symbols" step because the representation of the target has decayed to below threshold in the time that the model spent waiting for the eye movement and symbol extraction to complete.[16] With the high propagation rate this is not a common occurrence, but as the rate parameter is

[16] A word should be said about the probabilistic behavior of the model. The model's mechanisms are all entirely deterministic—there is no randomness in the model itself. As previously mentioned, though, there is some randomness in the initial conditions. The model is sensitive to differences in initial conditions, and hence each run of the model does not yield the same latency.

decreased, this becomes a more common occurrence.

Clearly, however, this is not what human subjects do—they do, in fact, successfully complete the task. Model I simply assumed that the initial representation of the target would be adequate, which was ineffective in all but the best of circumstances. Model II attempts to rectify this situation, again attempting to do so with a minimum of additional complexity

2.4 Model II: The contingent two-look model

In Model I, the model fails because it "loses" the representation of the target while moving to the code. Simply put, the target symbol is not active enough to remain above threshold in the 450 ms between the beginning of the eye movement and the availability of the code. One apparent solution to this problem would be to simply wait for the internal representation of the target to be "active enough" before initiating the eye movement to the code. While this seems like a simple approach to the problem, it is in fact impossible to implement in the current instantiation of SPAN. Why? Because, from the perspective of the pattern-matcher, elements are either above threshold or they are not—SPAN does not differentiate between activations of 0.1 and 0.9. The ability to perform this kind of "active enough" test would change the fundamental nature of SPAN—what would thresholds matter in that case? In Model I, the eye movement to the code is initiated as soon as there is a representation of the target symbol that reaches threshold. This is all the productions can test for, the presence or absence of an element. The simple solution of waiting for the initial representation to build up until it is "active enough" goes against how SPAN works.

The model can, however, detect that it has lost the target once it reaches the code. This is precisely what Model II does. Model II was constructed with two aims in mind: (1) it still performs the task as quickly as possible, and (2) if the fastest possible execution yields a failure, do the minimum required to compensate for any loss. Thus, Model II begins a trial much like Model I does:

- It begins the trial looking at the target.
- A memory representation of the target symbol is then constructed.

- The eyes are then moved to the code symbol, based on the target number.
- If the representation of the target symbol is present, compare target symbol and code symbol.

However, Model II's behavior is more sophisticated at this point. If no representation of the target is present when the code is recognizable, then the model takes the following steps to correct for the failure:

- Build an internal representation of the code symbol.
- Begin rehearsing the code symbol so it will not be lost.
- Move the eyes back to the target symbol.
- Compare target symbol (visual buffer) with code symbol, which is being rehearsed.

Model II (productions can be found in Appendix B), which is really an extension of Model I, has the important property that it only does additional processing (and takes further action) when it must. This is important if one assumes that the subjects are trying to perform as quickly as possible—when memory conditions are "good," they pay no time penalty. This strategy allows them to respond as quickly as possible on at least some of the trials.

2.5 Results of Model II

Model II is much more human-like in its performance. With a high global propagation rate (gamma = 1.0), the model is rarely required to make the second eye movement back to the target. As gamma is decreased, though, several things happen:

(1) The model "loses" the target more often and is forced to start rehearsal and make the second eye movement.

(2) More "garbage" is left in working memory at the end of the trial (the now-irrelevant rehearsal subgoal is still active).

(3) The speed at which all cognitive operations can be performed decreases, more often requiring two or three cycles instead of one.

All of these things combine to slow total performance. As gamma decreases,

response time increases. Figure 9-2 shows reaction time as a function of gamma for Model II. Since the behavior of the model is somewhat probabilistic, the graph points represent the mean of 140 runs of the model. Note that the mean for a gamma of 1.0 is 1055 ms, which is fairly close to the observed mean for the young adults of 1039, and the mean for a gamma of 0.8 is 1748, which is quite close to the observed mean for the older adults (1754).

It is important to realize that the performance differences are due primarily (but not exclusively) to an increase in the number of eye movements. The second eye movement (back from the code to the target) costs at least an additional 450 ms. The slope of the line represents a combination of the increased probability that the second eye movement will be necessary and the slower execution of basic cognitive operations. Note that the increase in time is more than simply a 1:1 increase in time with decreased propagation rate: a 20% change from a gamma of 1.0 to a gamma of 0.8 slows the model by around 70%—approximately the same as the difference between the young and old groups in the Salthouse and Coon data set.

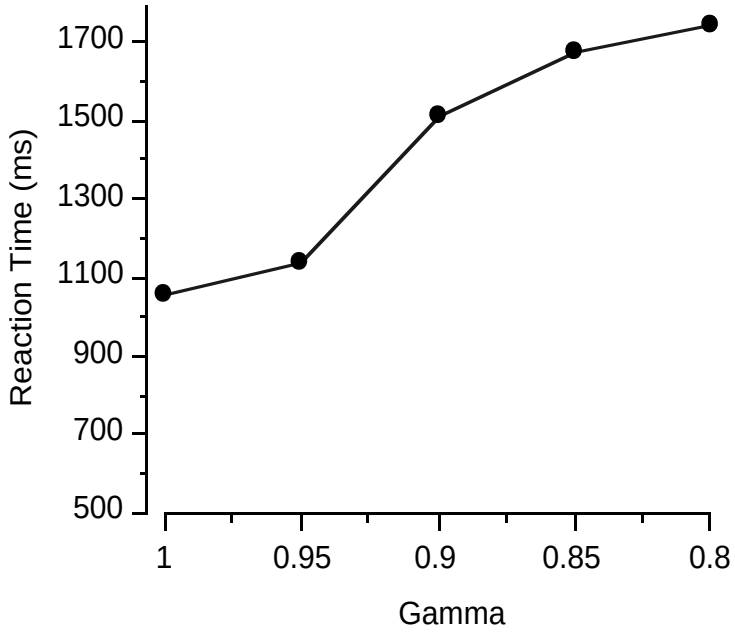

Figure 9-2 Digit Symbol mean RT as a function of Gamma parameter

3. Discussion

First and foremost, this model makes a clear prediction of what one ought to find out in the laboratory: slower performance on this task is the result of both longer looking times at both the target and code as well as more eye movements. This prediction is empirically testable, but does require sophisticated eye-tracking equipment. In this case, SPAN does fulfill one of the desiderata of constructing a model: generativity. The SPAN model generates clear predictions about what ought to be going on in this task—it predicts not only that there should be individual differences in task performance (that much we already knew), but what it is that is responsible for those differences: increased looking times as well as more eye movements. According to the model, simply slowing down the basic cognitive operations (i.e. longer looking times alone) is simply not enough to account for the large differences in performance.

This is a first step toward answering the question of *why* the Digit Symbol task is a good predictor of performance on a wide range of other cognitive tasks. If the Digit Symbol task provides a good measure of memory activation rate and if the gamma parameter in SPAN can create performance differences in a wide variety of tasks, then a score on the Digit Symbol task should predict those differences. Those are two reasonable "ifs" to wonder about, of course. The latter claim, that the gamma parameter in SPAN can account for a wide range of performance differences, can only be supported by building models of many tasks. This dissertation takes a step in that direction with models of three different tasks. The model of the Digit Symbol task presented here certainly makes the case that time on the Digit Symbol task is closely related to the value of the gamma parameter, but it is not undeniable proof—few things in modeling are. However, the model does demonstrate that slowing of activation rate is sufficient to account for the observed differences in Digit Symbol performance and, as yet, no other account has been developed that can make this claim.

In fact, the Digit Symbol task is the task that most clearly differentiates SPAN from other systems such as CAPS. It seems unlikely that the observed difference between

young and old could be modeled in CAPS, since working memory elements in CAPS are only lost as a result of displacement, not decay. Since there is nothing else that needs to be in memory except for the target (besides the task goal), there should be no displacement with CAPS. Thus, the target element should never get lost regardless of the capacity parameter (unless it is pathologically low) and everyone should be able to do the task in 1020 ms. It is also unclear how such an individual difference could even be considered in systems like EPIC and Soar, which do not lose memory elements without having them explicitly removed.

This is not to say that the SPAN model of the task is by any means infallible. First and foremost, there are other strategies that subjects can adopt that may make responses slower in some cases and faster in others. This seems particularly likely for the slower subjects, since they are unlikely to be able to execute the "optimal" strategy very often. It seems reasonable to assume that they might, in fact, switch to an alternate strategy in which they immediately assume they will forget the target on the way to the code and thus begin rehearsal of the target before moving to the code in the first place.[17] Under these conditions, the SPAN model predicts that these subjects should have particularly long initial looking times at the target and then make only one eye movement to the code. Subjects adopting this more conservative strategy will never have trials that go as quickly as those for subjects adopting the strategy used by Model II, but they will also not have trials that take as long. This is clearly an empirical question that can only be resolved with detailed eye-tracking studies.

Second, there are two other problem areas for the SPAN model: particularly fast responses and particularly slow responses. Over half of the young subjects had a median time that was less than 1020 ms, and the fastest median time was a speedy 726 ms, almost

[17] A SPAN model that employs this strategy has also been constructed. Though large number of runs were not done with this model, the basic differences between this and Model II is that the RT as a function of gamma (viz. Figure 9-2) is shallower than that for Model II, but the intercept is higher. This might be why some young subjects are especially slow and some older subjects particularly fast, but there is no way to assess strategy use from the reaction time data alone.

30% faster than the minimum 1020 that the model stipulates. According to the timing parameters, this fast a response should simply not be possible. Of course, the timing parameters could be too conservative, but it seems unlikely that they could be off by quite that much. Actually, what is probably going on in the very fast responses is that subjects are learning the digit-symbol pairings in the code table. If this happens, then an interesting result comes out. The time it takes to move the eye and extract a symbol from the visual buffer in the SPAN model is 450 ms. If one assumes that these operations are not necessary, that makes the response considerably faster. How much faster? Well, if one assumes that the 450 ms operation is supplanted by a retrieval of an element in declarative memory, and this element takes two production cycles to recover (100 ms), then the minimum response time of the model is cut down by 300 ms to 720 ms—right in line with the fastest subjects. The idea that the learning would tend to happen more within the young group is a tenable supposition, as several studies have shown that it is more difficult to learn when less memory capacity is available, in tasks ranging from simple paired associates (Baddeley, et al., 1984) to more complex problem-solving (Reber and Kotovsky, 1991). It has also been shown that training on the digit-symbol pairings does indeed reduce reaction times (Salthouse & Kersten, 1993), though it is not clear that the same learning goes on during simple execution of the Digit Symbol task itself. Whether or not the learning and commensurate reduction in eye movements is what is happening in the Digit Symbol task is clearly an empirical question, though the prediction made by this account is an intriguing one—the very fastest subjects can do the task without making an eye movement at all. This is an interesting (and very testable) prediction, to say the least. Unfortunately, to the best of my knowledge, no eye-tracking data are currently available for the Digit Symbol task.

This is not to say that this phenomenon is currently model-able with SPAN. At present, SPAN includes no learning mechanism, so it would be impossible to model this process with SPAN. However, the SPAN model does allow the estimation of the maximum benefit of the learning to be approximately 300 ms (350 ms if they learn the table

very well and can retrieve entries in one cycle), and this is indeed consistent with the times for the fastest subjects. So the SPAN model is at least consistent with the fast responses even though it cannot model the process of acquiring them.

On the other end of the performance spectrum, SPAN has trouble with the very slow responses as well. Even with the gamma rate set at 0.8 (at which point the model averages a response in 1754 ms), the model only rarely produces a single response slower than about 2100 ms. However, several subjects in the Salthouse and Coon data set have slower median times than this, with a few scores in the 2500 ms range. While these might simply be very slow subjects who should be modeled with even lower gamma rates, that seems unlikely since some of these subjects with very slow Digit Symbol scores had reasonable Computation Spans (see Chapter 10 for more information on the Computation Span task). This could well be the result of a violation of one of the assumptions made in building the model, namely that the mapping between the match condition of the target and code is handled by a single production. That is, there is a production that compares the representations of the two symbols and if they match generates the appropriate motor command (i.e. "press the slash key").[18] While that assumption certainly appears to be reasonable for the younger subjects with their very rapid response times, it may well not be the case for many of the older subjects. For them, the mapping between "match" and "press slash" might be stored as a declarative memory element that must be retrieved. This would add additional time according to how long the retrieval would take. On the other hand, the mapping might not be retrievable at all, in which case the display would have to be searched to find it—a time-consuming process. This last source of extra latency would also be assessable via an eye-tracking study, while it is not entirely clear how easy it would be to assess whether or not subjects are retrieving a declarative representation of the match-response mapping.

One other problem with the SPAN model is the simplistic treatment of the eye movement component of the model. It is highly probable that subjects do not, particularly

[18] There is, of course, a corresponding production that checks to see if there is a mismatch and then generates the "press the z key" motor action.

at the beginning of the experiment, immediately move their eyes directly to the correct key symbol on the code table. That is, subjects probably engage in some kind of limited visual search of the table, as has been suggested by Salthouse (e.g. 1994b). Modeling this process would require a more sophisticated model of visual perception and attention than is currently part of SPAN; hopefully, this can be addressed with future work.

Finally, the SPAN model is somewhat inconsistent with the human data in that the SPAN model does not make errors on this task. In this particular task, this is not a huge shortcoming since the subjects make very few errors themselves (the total error rate in the Coon and Salthouse study was under 4%). Still, there are subjects who make more errors than others and there may be some speed-accuracy tradeoff effects that it might be possible to model if we had a clearer understanding of what it is that causes the errors in the first place. While no model of these effects is in place, SPAN holds some promise in regards to error modeling as activation-based mechanisms generally seem better able to cope with errors than strictly symbolic systems (see Anderson, Reder, & Lebiere, in press; Byrne & Bovair, in press for some examples of activation-related errors).

In summary, it seems that the SPAN model is generally consistent with the Salthouse and Coon results. Most importantly, the model of the Digit Symbol task demonstrates that an account based on general slowing of memory activation rate can indeed produce the kinds of differences observed in human subjects even if it cannot model every last detail of every difference. The model also provides information about what to look for in a more low-level empirical investigation into the Digit Symbol task. In addition, the SPAN model presented here is as yet the only computational account of the Digit Symbol task and the age-related differences found on this task.

CHAPTER X.

A SPAN MODEL OF COMPUTATION SPAN

This chapter describes the third and final application of SPAN, this time to the "Computation Span" task used by Salthouse and colleagues (e.g. Salthouse & Coon, 1994; Salthouse & Babcock, 1991; Salthouse 1992a). Again, the task itself will first be described, then the model of the task presented, and finally results of the model presented and discussed.

1. The Computation Span Task

The Computation Span task is one of a family of tasks designed to assess working memory capacity. These tasks have two central components: processing and storage. Subjects are to perform some cognitive task requiring non-trivial processing while retaining some items for later recall.

In the Computation Span task, subjects perform simple arithmetic calculations while retaining a number from each of those calculations. Subjects are asked to solve problems like "8 + 3 = ?" and then retain the second operand (in this case, 3) for later recall. After the presentation of each problem (see Figure 10-1 for an example), the subject selects the correct answer from a set of three alternatives. An alternative is selected by using the arrow keys on the keyboard to move an arrow on the screen to point at the appropriate alternative. The number of arithmetic problems presented on each trial is gradually increased from one to nine, with three trials at each series length. Once a subject has solved a number of problems equal to the series length, a screen with the word RECALL appears and subjects type in the digits they were to remember in the order they

were presented (see Figure 10-2). The experiment program continues increasing the series length as long as subjects are correct on the solution to the arithmetic problem and the recalls for two of the three trials. A subject's span is the longest series length that is successfully completed.

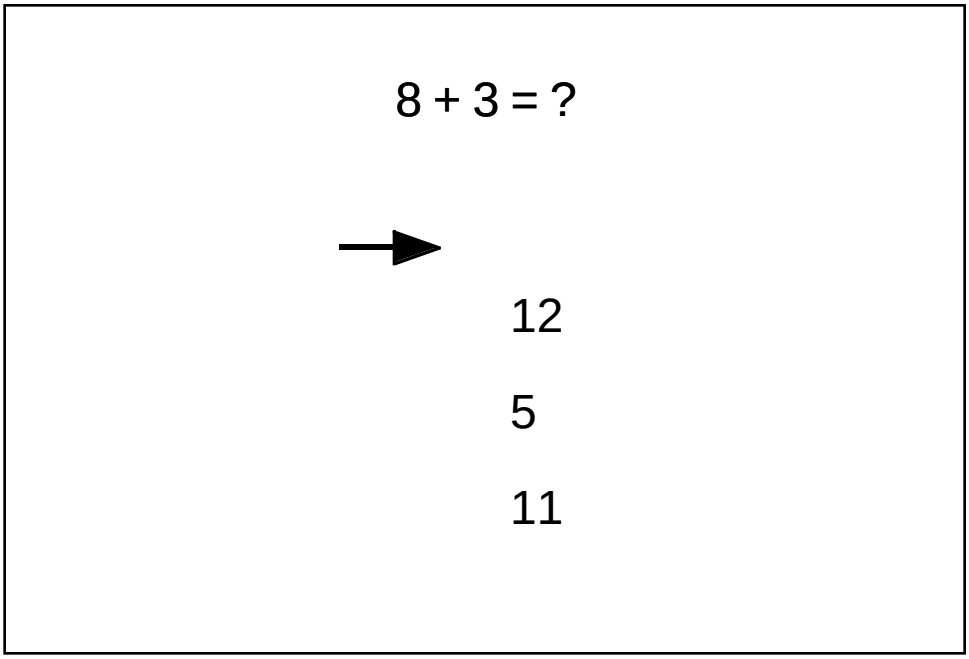

Figure 10-1 Problem presentation for Computation Span

The reference data for the model again come from the Salthouse and Coon (1994) Study 2. In this study, the mean span for the young group was 5.55 and for the older group was 3.94. It is also important to note that performance on the Digit Symbol measure and the Computation Span measure were correlated, $r(79) = -0.47$, $p < 0.001$.That is, subjects with larger spans also had faster Digit Symbol times (this result has been replicated numerous times). Data like these are one of the key sources of evidence for the working memory/speed relationship.

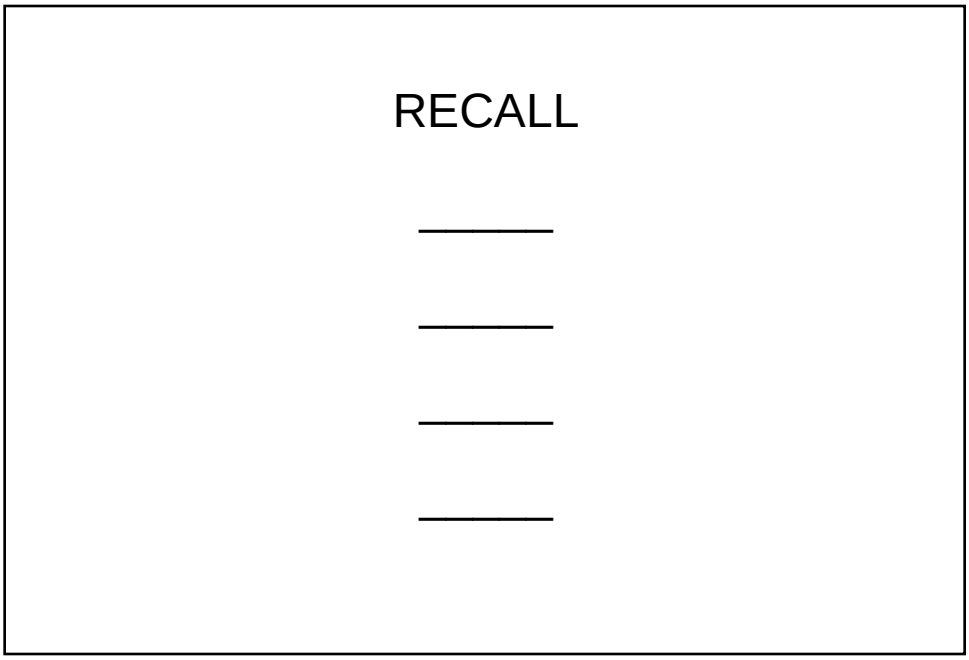

Figure 10-2 Recall screen for span of 4

This family of working memory assessment tasks that includes Computation Span takes Baddeley's definition of "simultaneous processing and storage of information" as the basis for what the task ought to include. Thus, all these tasks include both a storage and processing component and gradually increase difficulty by increasing the storage demand. Other examples of tasks from this family include Daneman and Carpenter's (1980) Reading Span task and Engle's Operation Word task (e.g.Turner & Engle, 1989; Cantor & Engle, 1993). Not only are these tasks designed to match up with Baddeley's theoretical definition of working memory, but, as mentioned earlier, working memory measures like these are good predictors of performance on a wide variety of cognitive tasks. Oddly enough, a detailed model of performance on any of these tasks has never been presented.[19]

[19] For example, in the Just and Carpenter (1992) CAPS paper, subjects were categorized as either high- or low-span on the basis of their performance on the Reading Span task and models of high- and low-span subjects were built. However, parameters for these models were not set on the basis of a model of the Reading Span task. It is not entirely clear exactly how Just and Carpenter determined the parameters that differentiated high- and low-span subjects for their CAPS-based CCREADER model.

No one has yet demonstrated a set of mechanisms that actually explains how or why people who have longer spans are different from those with shorter spans—it has simply been assumed that differences are the result of greater or lower working memory capacity. This explanation begs the question—what is it that causes the reduced capacity and why does that manifest itself on these kinds of tasks? Thus, the SPAN model presented here is the first of its kind and will hopefully provide a more satisfying answer to some of the questions regarding tasks of this kind.

2. The Model

There are two basic requirements in this task for the model to perform: answering the arithmetic problem and remembering the digits. Again, the goal of the model construction was to generate a model that performs the task using as few operations as possible. Thus, the arithmetic problem is solved by simple retrieval of a declarative arithmetic fact and comparison of that result to the answers on the screen, one option at a time. Retaining the digits is accomplished by a rehearsal process that will be described below.

In addition to performing the basic task, it was hoped that the model would demonstrate a performance decrement as a function of reduced speed (propagation rate). Once again, a gamma value of 1.0 was used to simulate the young subjects. Furthermore, it was hoped that results from the Digit Symbol model could be used to predict performance on the Computation Span task. That is, based on the results obtained from that model, a gamma value of 0.8 ought to predict the performance of the older subjects in the Salthouse and Coon study. Complete productions for this model appear in Appendix C.

2.1 Assumptions

This model is somewhat higher-level than the Digit Symbol model presented in Chapter 9. Because the key measure of interest is not latency but rather accuracy, less

attention has been paid to timing issues. The model of the Computation Span task does not take times for eye movements and keystrokes into account in order to make it possible to focus on the more central issues of task performance and rehearsal. The model still uses a cycle time of 50 ms and again uses single-element representations for the key items (in this case, number) in the task. Default rule strengths of 1.0 are used for all rules and element biases, with the exception of the representations of numbers and math facts. Since these are likely to be fairly well-used elements, they use the same "stronger" value of 1.3 that is used in the other models. Some assumptions are also made about strategy in this task—in particular, it is assumed that all subjects make use of explicit rehearsal to maintain the to-be-remembered numbers. The rehearsal process plays an important role in this task and so will be described in some detail.

2.2 Rehearsal

Regardless of memory load, memory elements in SPAN are lost over time because of decay. In order for a memory element to be active later on, some process must be working to supply activation to that element. One of the ways that people maintain a set of items over brief periods of time is through rehearsal. Strangely enough, there are no prominent computational models of rehearsal in the cognition literature, so again this SPAN model is breaking new ground.

One of the basic facts about rehearsal we know is that it is dependent on some kind of verbalization—articulatory suppression typically destroys the ability to rehearse (Baddeley, 1986). However, Baddeley's conception of verbal rehearsal is quite primitive—rehearsal is a thing, the "articulatory loop"—one of Baddeley's three "boxes" that make up working memory. However, Baddeley's contribution to our understanding of rehearsal from an empirical standpoint is unmatched. It is Baddeley's (1986) summary of results on rehearsal that was used to determine the decay equation used in SPAN, which sets the time course requirements for the rehearsal model.

In SPAN rehearsal is not a thing but a *process.* First, the rehearsal process requires that a "rehearse" goal be active. Rehearsal as a process works like this: A command is sent

off to the motor processor to vocalize the item to be rehearsed. The motor processor creates this vocalization (latency: 70 ms, the standard motor processor cycle time), and then the perceptual processor "hears" the vocalization by creating an element in external memory for that item (latency: 150 ms, the perceptual processor cycle time to recognize a familiar sound). A production then fires which transmits activation to the internal representation of the item. Because SPAN can do things in parallel, it does not wait for the rehearsal of one item to complete before initiating the rehearsal of the next item. This allows rehearsal of more items in sequence. Without this interleaving, the SPAN model can rehearse only a few items, but with it, the model can rehearse about eight items. If the model is not doing anything else, it can maintain this rehearsal indefinitely by simply looping over the same set of items again and again. A special dummy "start" item (think of this as an intentional pause) must be included in the rehearsal for the SPAN model to recall the items in order; otherwise the model has no way to tell which item is the first or last item on the list. One other major simplification has been made, and that is the assumption that all items take equal time to rehearse. This is technically false (we know it takes longer to say "seven" than it takes to say "one") but since only one of the digits takes more than one syllable, this is a reasonable approximation.[20]

This model of rehearsal is fairly simple but interacts with other processes in subtle ways. This will become clearer as the full model of Computation Span is described.

2.3 Computation Span

To perform the Computation Span task, the model goes through several steps:

(1) Add the appropriate operand to rehearsal loop

(2) Use operands and operator to retrieve value of arithmetic computation

(3) Move the arrow on the screen to the target item and hit "return"

Then, of course, there is

[20] This approximation, while tenable for arbitrary English codes for digits, would not work well for arbitrary words. However, what is clear is that SPAN could rehearse fewer items if the items took longer to verbalize, which is consistent with empirical findings.

(4) Recall

Each of these steps is considered a subgoal in the task, and each bears further explanation.

Step (1), adding an item to the rehearsal buffer, does not add an item to any kind of rehearsal buffer, because there is no such buffer—rehearsal is a process, not a buffer. What the model actually does here is wait for the "start" item to come around in the rehearsal loop, and instead of issuing the motor command to vocalize the "start" item, it issues a motor command for the vocalization of the new item. Thus, before the item is considered added to the loop, a representation of the verbal code for the item is constructed and then sent off to the motor processor.

Step (2), retrieval of the solution, is done in much the same way as the instructional examples of arithmetic provided with ACT-R (Anderson, 1993). That is, the model has a store of declarative arithmetic facts of the form (operand1 operator operand2 result). The operands and operator are used to retrieve the appropriate arithmetic fact, and an element representing the result is constructed based on the retrieved fact. The model proceeds to step (3) when this result becomes available.

Step (3) simply matches the obtained result with the appropriate answer on the screen. The arrow on the screen starts above the first item, so the first thing done in step (3) is issue a motor command to hit the arrow key to the first answer. If the first answer does not match the computed result, another motor command is issued to move down to the second answer and this process is then repeated. When the pointed-to answer matches the computed answer, a motor command to press the "return" key is issued, which signifies the end of the question.

When the word RECALL appears, Step (4) starts. To recall the items, the model waits until the "start" number comes around in the rehearsal loop, then presses the key on the keyboard which corresponds to each number as it comes through the rehearsal loop. Each number keystroke is followed immediately by a press of the "return" key, which advances the cursor position to the next open slot for recall. If the model has successfully rehearsed all the items up to this point, they will all be successfully recalled.

3. Results

The key results are summarized in Figure 10-3, which shows the probability with which the model completes a given span length for gamma values of 1.0, 0.9, and 0.8. Each point represents the results of 50 runs of the model. Again, the behavior of the model is probabilistic partly because of randomness in the initial conditions. In this model, unlike the Digit Symbol model, the problems themselves also affect the outcome—some problems take slightly longer because the answer is further down in the list of alternatives, and this can sometimes affect the outcome.

The first point to take away from the graph is that, in general, the higher the gamma, the more often the model was able to complete a given span. This does not apply to very short spans, which the model could always complete, and very long spans, which the model could never complete. The relationship between gamma and span is what was hoped for—the faster the propagation rate, the longer the span. Notice in particular the span that the model would predict for gammas of 0.8 and 1.0. For a gamma of 0.8, the model nearly always completes a span of 4 but almost never completes a span of 5. Some simple arithmetic on the completion probabilities indicates that the expected span value for a gamma of 0.8 is 3.95—this is very close to the observed mean span of 3.94 for the old group in the Salthouse and Coon (1994) data set.

Now, looking at the curve for gamma of 1.0, the model nearly always completes a span of 5, and sometimes can complete a span of 6, though it never completes a span of seven. The expected value of span for gamma of 1.0 is 5.27, which is slightly less than the observed mean of 5.55 for the young group in the Salthouse and Coon (1994) data. What is significant about these values of gamma is that the values of 0.8 and 1.0 used here also did a good job of modeling the young and old groups on the Digit Symbol task (though again, the young subjects did slightly better than the model predicted). That is, a reduction in the gamma parameter of 0.2 predicts appropriately-sized performance decrements in both the Computation Span and Digit Symbol tasks.

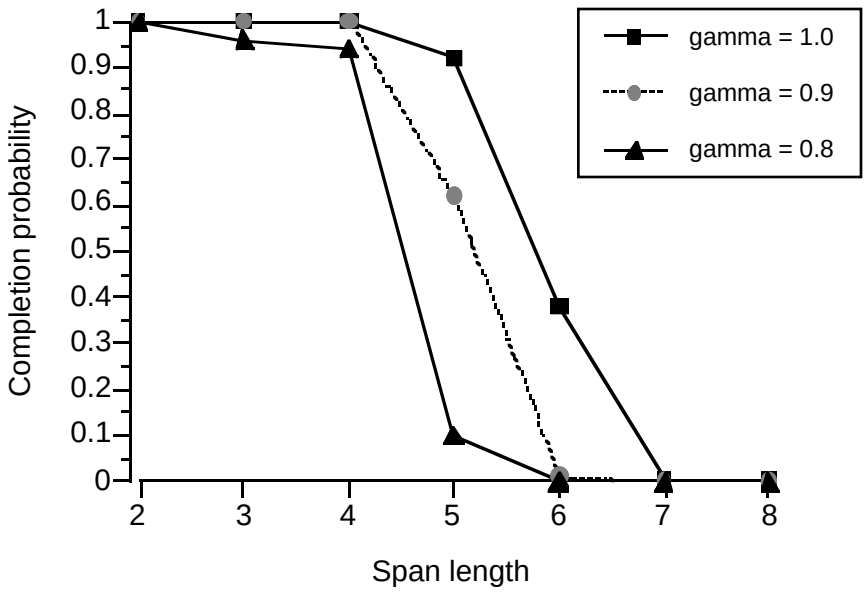

Figure 10-3 Probability of completing span lengths

Also like the Digit Symbol model, the Computation Span model shows graded performance—intermediate values of the gamma parameter produce intermediate performance (see the curve for gamma = 0.9 in Figure 10-3 as an example). Thus, this change in a single parameter value may well be suitable for modeling not only extreme-groups designs, but more subtle differences between individuals.

One natural question to ask about the Computation Span model is Why does performance degrade when the propagation rate is reduced? The answer lies in the noise function in SPAN. As more items become active and noise increases, the amount of activation propagated by productions is reduced. For individuals already operating at a lower propagation rate, negative consequences are more likely. In the Computation Span model, rehearsal is affected by the amount of noise in the system. Each number being rehearsed has a limited amount of time to be activated on each pass through the rehearsal loop. If an element does not receive enough activation during a particular pass, then it will fall below threshold and be unavailable on the next pass through the loop—the item will be

forgotten. The increased noise created by the arithmetic problems can be enough to render rehearsal unable to maintain a large set of items—and slower individuals are especially susceptible. The slower the initial propagation rate, the more damage noise does to the process, effectively "swamping" the system and making it unable to propagate the necessary activation. This is in accord with what aging researchers (e.g. Salthouse, 1992b) sometimes call the "complexity effect": the more complex the task, the larger the differences between young and older adults. As this SPAN model demonstrates, this phenomenon could simply be the natural result of a reduction in the rate of activation propagation.

However, like the Digit Symbol model, the Computation Span model does have its weaknesses. First of all, it does not make arithmetic errors; that is, the model always retrieves the correct arithmetic fact for a given problem and locates the correct response. While this is almost certainly not true of the human subjects, it is not totally unreasonable—subjects are instructed to concentrate primarily on the arithmetic task when the combination of the two tasks becomes difficult. The model also does not do recall in a particularly graded way when it fails to get recall completely correct—it either recalls all the numbers or it recalls none of them. This is because if it fails to recall one item in the list, the model loses its place and fails on all the other items. While clearly unrealistic, this is not intrinsic to how the model works—there simply are no productions present to handle failure cases, though there could be. The model was not intended to mimic failure behavior, only predict failure likelihood. Third, the model performs the task somewhat faster than human subjects. The latency data gathered from a sample of 12 young and 9 older subjects (not the same subjects as the Salthouse and Coon subjects, since latency data were not collected in the Salthouse and Coon study) appears in Tables 10-1 and 10-2.

Table 10-1 Completion times for arithmetic problems in Computation Span

Data from	Completion time
Mean latency, young subjects	3659 ms
Mean latency, model with γ = 1.0 (length 4)	1628 ms
Mean latency, old subjects	6960 ms
Mean latency, model with γ = 0.8 (length 3)	1963 ms

Table 10-2 Mean completion time for recall in Computation Span

Span length	Young Subjects	Model, γ = 1.0	Old Subjects	Model, γ = 0.8
3	3462 ms	-	6686 ms	2275 ms
4	4263 ms	3470 ms	7997 ms	3685 ms
5	5064 ms	4576 ms	9308 ms	4356 ms
6	5865 ms	5694 ms	-	-

While only a limited amount of data is available, the model is universally faster than the human subjects. This is hardly surprising, since the model does not include some of the lower-level operations like eye movements, nor is time for keystrokes factored in. Since eye movements and keystrokes can indeed take a long time to execute, it stands to reason that the latencies generated by a model that does not include them are not particularly close to the observed values. Predicting response time was not the goal of the model of the Computation Span task, though, so these mismatches are not particularly consequential.

On the other hand, the model is consistent with some of the patterns of latencies in the human subjects. The older subjects took longer to answer the arithmetic problems and to do the recalls, and the gamma = 0.8 model takes longer to do both of these than the gamma = 1.0 model does. Furthermore, human subjects (both young and old) take longer to do the recalls when there are more items to recall, as does the model. The actual effects on the time estimates of the model were it to include the lower-level operators would surely be higher and thus more in line with the human data, but because cognitive, motor, and

perceptual operations can potentially occur in parallel, the magnitude of such additions is impossible to estimate.

Of course, this model represents only one of several possible strategies. In this model, rehearsal of the second operand is initiated before solving the arithmetic problem. This is not necessary—nor do subjects necessarily rehearse. While it is not at all clear when subjects who rehearse add the new item to the rehearsal loop, it is highly unlikely that many of the subjects do not rehearse—at least those with spans above one or two. Articulatory suppression has repeatedly been shown to reduce performance on working memory measures (see Baddeley, 1986 for several examples), suggesting that most people do spontaneously rehearse in working memory-type experiments if they are allowed to do so. As for the effects of when rehearsal of a new item begins, the implications of this are not as clear but there is nothing in the model that suggests this should have much of an effect in either direction. Certainly, a failure to rehearse would affect performance, but as long as the items are rehearsed, the performance should be more or less unaffected.

Finally, like the Digit Symbol model, this model has trouble producing particularly low and particularly high spans. The source of very low spans may indeed be subjects who attempt the task without rehearsal, but as yet there is no empirical evidence to support this claim. Perhaps running an experiment in which articulatory suppression is used would generate such low spans, but this would not prove that lack of rehearsal is how the Salthouse and Coon subjects did so. On the other end of the performance spectrum, the SPAN model does not correctly complete a span of 7, even with a gamma of 1.0. Some of the higher spans produced by the human participants might be a result of individuals with particularly high propagation rates, but there are alternatives to this interpretation. In particular, learning provides one potential explanation. Some subjects might be able to chunk the digits or encode them in long-term declarative memory, increasing their functional capacity. SPAN is currently incapable of modeling either of these processes. However, the model is not intended to be a model of idiosyncratic subjects—it is intended to capture the overall pattern of the data, which it does fairly well. So, while there are

inaccuracies in the model, none of them are particularly limiting inaccuracies.

4. Discussion

The model presented here represents several important steps in understanding working memory. First, it provides an explanation for *why* working memory tests like the Computation Span task are useful measures when it comes to predicting performance on other "fluid" cognitive tasks. Essentially, the information the Computation Span task provides is a measure of how well an individual can handle noise generated by processing demands—and this, in turn, is determined by how fast an individual can propagate activation. Slower individuals should get "swamped" earlier and thus receive lower span scores. Thus, the Computation Span task, and those like it, provide a good measure of propagation rate—just as the Digit Symbol task does. They simply provide two different ways to measure it.

Why, then, are the various working memory measures and the Digit Symbol measure imperfectly correlated? There are a variety of possible explanations. First, none of them are perfect measures of propagation rate—all the tasks in the laboratory can do is attempt to measure the effects of different propagation rate. The imperfect correlations may arise because the effects of propagation rate are sensitive to factors other than just the task. Performance on the Digit Symbol task is clearly sensitive to strategy, and performance on the Computation Span task is actually quite sensitive to the strengths of a few key productions and the biases of the declarative elements being rehearsed. Factors like fatigue, boredom, and motivation are all potential candidates and SPAN, like virtually all other computational models, has nothing to say about the effects of these factors—yet surely they do influence performance. Learning, too, probably affects performance on these tasks and SPAN does not handle that, either. Finally, it is not clear that scores on all such tasks are necessarily linearly related to propagation rate, an assumption correlational measures make. A more clear understanding of the relationship between activation rate and task performance, the beginnings of which have been demonstrated here, is the key to

untangling many of these relationships.

Despite its shortcomings, the SPAN model of Computation Span does break some important new ground. It is the first computational model of a task from the working memory assessment family, and does provide insight into why individuals differ on this task beyond the vacuous “some individuals have more capacity” explanation. Equally importantly, it demonstrates a clear quantitative correspondence between scores on the Digit Symbol task and the Computation Span task. More will be said about this aspect of the model in the following chapter.

CHAPTER XI.

GENERAL DISCUSSION BASED ON THE MODELS

1. Sufficiency of Slowing

The models demonstrate one of the central contributions of SPAN: a strong demonstration that slowing is sufficient to produce differences in cognitive performance that mirror those found in the cognitive aging literature. One of the questions that has been raised about "general slowing" accounts of cognitive aging (e.g. Hertzog, personal communication; Engle, personal communication), is "How can slowing be responsible for such widespread effects?" Cognitive aging researchers have found age-related decrements on a wide variety of tasks—how could a single, simple mechanism possibly be responsible for such widespread effects? The models presented here are a beginning of an answer to this question. They show that slowing alone is sufficient to explain individual differences in three different cognitive tasks: sentence verification (fan effect), the Digit Symbol task, and Computation Span. Looking carefully at how SPAN works and how the effects of slowing have played out in these tasks, it is clear that slowing can affect performance on quite a wide variety of tasks. This is because slowing affects the overall efficiency and robustness of the cognitive system. Any task that is critically dependent on speeded operations or simultaneous maintenance of multiple pieces of information can potentially be affected by slowing, and this is essentially what has been found in the aging literature—a global performance decrement on "fluid" ability measures.

Other accounts of aging and working memory do not have this character. The "inhibition deficit" account, for example, has not been demonstrated to be sufficient to

explain any differences on any task.[21] Nor do task-specific deficits (e.g. Rogers, Fisk, & Hertzog, 1994) explain the generality of the results seen in cognitive aging (for an excellent recent review, see Salthouse, 1996). SPAN provides the sufficiency argument for slowing that these other explanations lack.

Not only is the slowing account the only one that has demonstrated sufficiency for some aging and memory phenomena, it is the most parsimonious explanation for such effects. Task-specific deficits require that a new explanation be constructed for each task encountered. At what point does it stop? Is there a visual search deficit in older adults, or are there thirty-six of them, one for each digit and each letter of the alphabet? How does each one work? Why are there so many of them? Inhibition deficits fare little better on such questions. How does an inhibition deficit explain longer reaction times on simple tasks like the Digit Symbol task—what irrelevant material is there to inhibit in this simple task? How does the inhibition deficit actually work? These accounts generate as many questions as they answer.

This is not to say that slowing is the only possible explanation or even the best possible one, nor does it mean that there is not an inhibition deficit or task-specific decrements. Like other computational models, SPAN does not provide a necessity argument, only a sufficiency argument. SPAN has demonstrated that slowing is sufficient to produce performance differences on the three aforementioned memory and speed tasks—but it does not rule out other explanations, nor is it a guarantee that slowing can account for all age-related changes in cognition. SPAN simply provides a piece of converging evidence that speed is a key component in understanding aging and cognition. It is, however, an important piece since cognitive aging has generally lacked formal accounts.

[21] Sufficiency has been *claimed* for the inhibition deficit hypothesis, but it has never been demonstrated—there is an important difference. It is not at all clear that the inhibition deficit account is specified well enough to base a running simulation model on it. Without better specification, claims about sufficiency are just that: claims.

2. Uniqueness

One legitimate concern regarding SPAN is its uniqueness with respect to other production systems—what differentiates SPAN from other systems? Could another system be made to behave more like SPAN with minor changes? The three SPAN models presented here provide clues to the answers to these questions. There are currently four major production system theories being promoted by various researchers: CAPS, Soar, EPIC, and ACT-R. The relationship between each one of these systems and SPAN will be examined in the remainder of this section.

2.1 SPAN and CAPS

The current version of CAPS, which is best described in Just and Carpenter (1992), is a production system that was designed to model individual differences in working memory and reading. SPAN and CAPS have a great deal in common, right down to common source code. In many ways, CAPS was the inspiration for SPAN and SPAN borrows heavily from CAPS. Some of the central features borrowed from CAPS include activations for all declarative memory elements, threshold-based pattern matching (i.e. an element has to be above threshold to be matched), parallel rule firing, and activation propagation as a production action. I am deeply indebted to Just and Carpenter (1992) for many such ideas. However, CAPS and SPAN diverge in several important respects.

First and foremost, of course, is the role of speed. In SPAN, speed determines effective working memory capacity. In CAPS, it works the other way around: speed is a function of memory capacity—but only in certain circumstances. CAPS is only slowed when the system is trying to use more activation than it is allowed. This is one of the key differences and is brought out nicely by the Digit Symbol task. In the Digit Symbol task, the subjects only have to remember one item, the target symbol. Thus, for there to be any slowing in CAPS in the Digit Symbol task, the system's capacity would have to be lowered such that less than two items (one of the task goal) could be held above threshold. However, such a low capacity seems unlikely, since the CAPS models presented in Just and Carpenter (1992) use capacities on the order of 100 items and even the smaller models

in Byrne and Bovair (in press) use capacities on the order of 8 items. It is hard to imagine how CAPS could do much of anything with a capacity of only two items—the Computation Span task, for example, certainly requires a capacity higher than that.

A second key difference between SPAN and CAPS also involves time. When CAPS models are slowed by reducing the capacity, the slowing is reflected in an increase in the number of production cycles taken to finish the task. However, the relationship between number of cycles taken and real time is unclear. In the Just and Carpenter (1992) article, various graphs disagree to a great degree on what a cycle means in real time. For example, in their Figure 8, nine production cycles are mapped to a real difference of 150 ms, or about 17 ms/production cycle. In the very next figure, 6.5 cycles are mapped to a difference of about 300 ms, or about 46 ms/production cycle—this is a factor of three difference. This is possible in CAPS because there is no "real time" clock anywhere in the system, all mechanisms are based on cycle-to-cycle operation. In SPAN, this is simply not possible since the decay function is defined on a real-time basis, not a cycle-time basis. This forces the modeler to choose a real-time value for the duration of a production cycle. This makes reaction time predictions in SPAN truly quantitative, whereas in CAPS only qualitative patterns can be consistently reproduced.

The third major difference between SPAN and CAPS is decay. SPAN has it, CAPS does not. Declarative memory elements in CAPS only lose activation when the system is trying to use more activation than it is allowed, just like slowing in CAPS. Again, CAPS should not only fail to show slowing in the Digit Symbol task, but also it should fail to show forgetting as well. Further, there is empirical evidence that unrehearsed items in memory do indeed decay, as discussed in Chapter 5. This gives SPAN an edge in explaining a number of apparently decay-related phenomena, such as why rehearsal is necessary to maintain only a few items, even in the absence of other simultaneous memory load.

There are other differences that may or not be major, since these mechanisms have not been carefully explored in either system. The first among these is the difference

between SPAN's noise function and the global capacity constraint in CAPS. SPAN's noise function is designed to create a system in which very strong productions targeting frequently-used elements "costs" the system as little as possible, creating a functional increase in working memory capacity for activities that make use of such rules and chunks. How this would be handled in CAPS is unclear, since productions that request more activation can actually penalize other productions and elements. However, there may be other ways to achieve similar effects by adjusting the thresholds on a production-by-production basis, which is possible in CAPS (and used extensively in the CCREADER model). Since neither SPAN's noise mechanism or the CAPS threshold-lowering approach has been extensively tested, it is not clear whether either account is the right one.

Another difference between the two systems that seems on the surface to be a large difference is the unitary vs. fragmented view of working memory. CAPS explicitly embraces the idea of separate "resource pools" in working memory, e.g. verbal and spatial. SPAN, on the other hand, is a unitary system. Since empirical evidence for the existence of separate resource pools is mixed, it is not clear that either of these views is superior to the other. In fact, it is not entirely clear that this difference in carving up the memory even necessitates any different predictions. It may be possible to model the differences on, for instance, verbal and spatial memory tasks with differences in productions rather than differences in the underlying architecture. Sometimes such discriminations are difficult to make, and SPAN was designed to incorporate as few additional mechanisms as possible.

The last difference between SPAN and CAPS that merits mention is the handling of inhibition, or negative activation. In SPAN, productions which propagate negative activation are treated no differently than any other productions. Thus, inhibitory processes in SPAN would be affected by age and memory load. As mentioned in Chapter 5, there is at least preliminary evidence that this is the case. Interestingly, productions in CAPS which request negative activation are treated differently than other productions. Negative activations are *not* scaled back in CAPS when scaling back takes place—they are

completely unaffected. In a sense, productions which propagate negative activation are privileged productions not held to the same constraints as other productions. Based on this, it is reasonable to expect that CAPS models would not predict reduced inhibitory abilities in older subjects. However, since the current models constructed with SPAN and CAPS make little use of inhibition, it is important to temper this comparison. It is not yet clear whether this difference is an important one or not.

2.2 SPAN and Soar

Soar is designed to be a "cognitive architecture" in a very complete sense. Soar has been applied to a wide variety of tasks; there are Soar models that can read, play video games, and, most importantly, learn. Learning is the cornerstone of Soar, as there are essentially no numerical parameters anywhere in Soar. Because Soar is essentially parameter-less, the only way individual differences can be modeled in Soar is via differences in knowledge.[22] This is a strong theoretical stance which is hard to accept at face value—if there is any literature in psychology that makes the case for knowledge being intact while still having performance on a wide variety of tasks suffer, it is the cognitive aging literature. The parameter-less nature of Soar is also manifested in the fact that there is no structural limitation on Soar's working memory capacity. If there were such a limit, it could presumably be affected by the value of some parameter, and since there are no global parameters in Soar, that would be inconsistent. There really is not any clear or obvious way to think of working memory limitations in Soar, and so there is no corresponding way to think about individual differences in working memory. This one of the two central difference between Soar and SPAN. With respect to the central aim of SPAN as a testbed for a theory of working memory, SPAN currently has a clear advantage over Soar.

Though it is less central to SPAN's aims, one of the weaknesses of SPAN that arises even in the simple Digit Symbol model is the lack of learning mechanisms of any

[22] If there is another way to model performance differences in Soar, it is an esoteric method that is not obvious to Soar non-experts and has not been mentioned in any psychological work done with Soar.

kind. It is likely that at least some learning of the code table comes into play in the Digit Symbol task and SPAN is simply unable to model that aspect of task performance. Soar, on the other hand, was specifically constructed as a learning theory. This is quite sensible given the strong emphasis Soar places on knowledge. If knowledge is all-important, as it is in Soar, having an account for how the knowledge is acquired makes Soar a much stronger theory. In fact, learning places a strong constraint on how models are constructed in Soar, since ill-structured models will learn very odd things that are clearly nonsensical. Learning is the second major difference between SPAN and Soar, and an area in which Soar is clearly superior to SPAN.

There are a myriad of other less central differences between Soar and SPAN. In Soar, all knowledge is encoded in productions—there are no permanent declarative memory elements, unlike in SPAN. While productions in Soar fire in parallel like productions in SPAN, the "action" side of a Soar production cannot cause actions in quite the same way that SPAN productions can cause actions. Soar productions, for example, cannot directly initiate motor movements. They can, however, suggest that the system make a motor movement by manipulating "preferences" for what operator should be applied next. There are no activations in Soar, so there is no notion of a threshold for anything, and so on. The list of small but significant technical differences between SPAN and Soar is a long one—of the "big four" production systems, Soar is quite clearly the least similar to SPAN.

2.3 SPAN and EPIC

EPIC is a production system that is the brainchild of Kieras and Meyer (e.g. 1994) and is aimed at modeling human performance at an exceptionally detailed level. The major focus of EPIC is not, in fact, cognition but rather on the dynamics of perception, action, and cognition in rapid, dual-task situations. This is reflected in the amount of attention paid to the perceptual and motor aspects of human performance—Figure 11-1 gives some idea of the level of attention paid to these non-cognitive factors. Indeed, EPIC is a more fleshed-out version the Model Human Processor (MHP) of Card, Moran, and Newell

(1983). EPIC represents the state-of-the-art in integrating what has been learned about the performance characteristics of the human perception/action system into a production system framework.

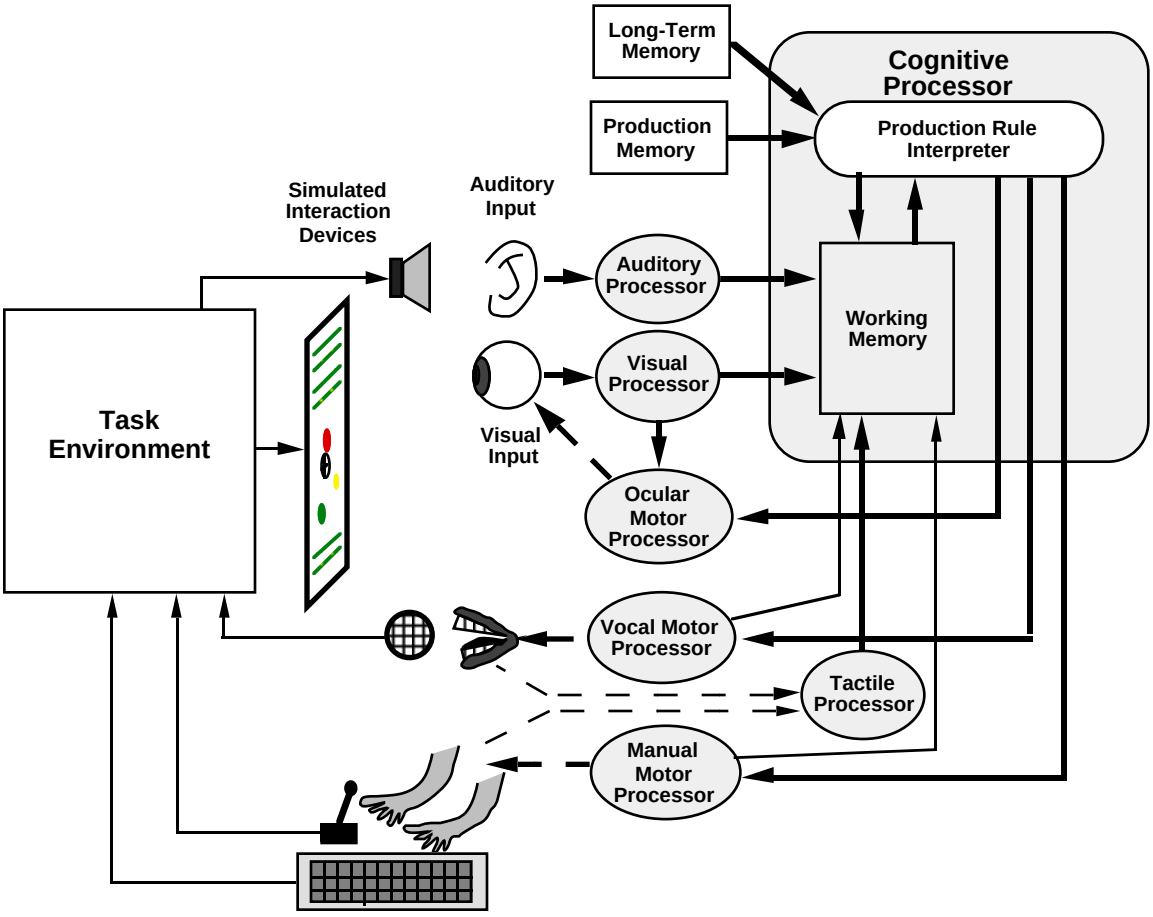

Figure 11-1 EPIC architecture

In terms of modeling working memory constraints, EPIC was never designed to model working memory-demanding tasks and thus includes no notion of limitations of working memory. In fact, EPIC was designed to show that to model certain classes of high-performance tasks, limitations on the cognitive processor are not necessary. The cognitive processor embedded in EPIC is the Kieras and Polson (1985) “parsimonious

production system," or PPS. PPS is just what one might expect, a very simple production interpreter. While, like SPAN, it has declarative and procedural long-term memory, parallel production firing, and no learning, that is where the similarity ends. PPS has no activations, no decay or displacement—nothing of the kind. There is nothing in the system that would prevent an EPIC model from constructing representation of a 1,000-word text that would be non-decaying, complete, and perfectly correct (Kieras, personal communication). Such a task would be next to impossible for SPAN and probably most human readers as well. This is not to say that EPIC is incapable of modeling any kind of individual differences. Each processor in EPIC has an average cycle time and performance is affected by those times. How exactly that relates to aging and memory phenomena has yet to be carefully examined.

In many ways, EPIC and SPAN are not at odds with one another. In fact, certain operations in the SPAN models presented here, particularly the Digit Symbol model, were specifically constructed to be approximations of how those operations would be handled by EPIC. The EPIC perception/action system is actually a desideratum for SPAN—were it technically feasible, the EPIC motor/perception machinery would have used it in lieu of the approximations. As it turns out, in the long run a fusion of EPIC and SPAN may indeed be technically feasible—SPAN would be inserted into the EPIC system in place of the current "cognitive processor."[23]

2.4 SPAN and ACT-R

ACT-R is the most recent instantiation of the ACT theory, whose origins go back well over twenty years (Anderson & Bower, 1973). ACT-R is probably the most complex of all the production systems, with activations, decay, rule strengths, learning, "rational analysis," and a host of other features and parameters.[24] There are, however, some special

[23] Preliminary work has been done with Soar in place of the EPIC cognitive processor with some success (Kieras, personal communication).

[24] No other production system I have yet encountered has as many parameters as ACT-R. For instance, in ACT-R 2.0, there are *sixteen* different numerical parameters that can be set for *each production.*

features of ACT-R that set it clearly apart from SPAN (and other systems as well).

First, ACT-R is unique among all the production systems discussed up to now in that productions are *not* allowed to fire in parallel. If more than one production matches on a given cycle, a "rational analysis conflict resolution" procedure is used to choose a single production. This conflict resolution procedure is fairly complex and takes into account match time as well as expected cost and probability of success for each production. However, since ACT-R propagates activation automatically and in parallel (without using productions to do so), the exact impact of this seriality in relation to other systems is unclear. Because activation is handled by what is essentially a connectionist network, productions in ACT-R are at a somewhat higher level than productions in SPAN or CAPS, focusing more on the action level. Still, it is not clear how ACT-R can tightly interleave dual tasks, which is possible in all the other systems described.

The second major difference between SPAN and ACT-R is is the handling of goals. In ACT-R, goals have a highly-privileged status among declarative memory elements. In ACT-R all goals are managed by a special stack. This means that one and only one goal is active at all times, and that goals cannot be forgotten. This restriction is further enforced by the fact that all productions are required to specify the goal to which they apply. This, in combination with serial production firing, very clearly differentiates ACT-R from SPAN. It is not at all clear how ACT-R would handle even simple dual tasks such as the Computation Span (in which the subjects have two goals, solving the arithmetic problems and remembering digits). Further, this makes certain kinds of errors, those involving goal forgetting, impossible in ACT-R. In contrast, SPAN treats goals like any other declarative memory element: goals decay and can be lost, and as many goals can be active at a time as the system can handle. Handling dual tasks is not a special problem for SPAN, except that multiple tasks tend to create additional noise—any problems that arise from dual tasks are functional, not structural.

On the other hand, ACT-R includes a number of features missing from SPAN, notably several different learning mechanisms. ACT-R has been applied to a much wider

variety of different tasks and so rests on a much stronger base of empirical evidence. The ACT family of theories has had a great deal of impact on cognitive psychology and its overall contributions would be hard to estimate—there is much to be said for ACT-R that certainly cannot be said for SPAN. However, there are areas in which SPAN and ACT-R diverge, and these may well turn out to be important areas.

One very interesting point of comparison between SPAN and ACT-R is the central issue for SPAN, the relationship between speed and working memory. ACT-R has only recently been modified to attempt to accommodate individual differences. This has been done by limiting the amount of "source activation" in ACT-R, which has two interesting effects: it reduces the functional working memory capacity of the system and slows down production matching (thereby slowing the whole system). This line of work in ACT-R is just getting started and it is not yet clear how this relates to SPAN's speed and activation functions, but there is some potential similarity. So, while SPAN and ACT-R have their differences, they may share some aspects. Hopefully, further work on individual differences by the ACT-R group will make the relationship between the two systems easier to evaluate.

3. Errors

Errors by subjects are found in virtually every set of data collected by experimental psychologists, and the tasks modeled with SPAN are no exception. While SPAN can certainly fail to remember digits in the Computation Span task, the SPAN models do not make errors in the same way human subjects do. It does not incorrectly reject sentences in the fan effect task, it does not incorrectly answer in the Digit Symbol task, and it always solves the arithmetic problems correctly when performing the Computation Span task. In some sense, the SPAN models are too good at the tasks. This problem is not limited to SPAN; in general, computational models capture little of the errorful character of human performance. Certain kinds of errors can be modeled by introducing systematic errors in the knowledge for a particular task (e.g. VanLehn, 1990), but this approach does not seem

like a good way to model errors in simple tasks like the Digit Symbol task or for adults doing simple arithmetic. This is a difficult issue in computational modeling. On the one hand, it is very difficult to build a model of a task that does not include correct or very nearly correct knowledge. (In this context, "correct" knowledge is correct in the sense that it enables the system to perform the task correctly.) On the other hand, when computational systems have the correct knowledge, they typically execute the task correctly every time. Errors are usually all-or-nothing—either the model always makes a particular error, or it never makes it. In general, SPAN has the same problem.

Systems like SPAN and CAPS may have some small advantage when it comes to certain kinds of errors, those based on working memory failures. Declarative elements that are necessary for correct task performance can get "lost" in both of these systems, so they may be better suited for modeling certain errors. Even this is hardly cut-and-dried, though, since it is usually not clear what the model ought to do when it loses a critical piece of information. It is simple to have the model simply fail in these cases, but that does not seem to match what we typically see subjects do in the laboratory—they typically respond with *something*, even if it is an incorrect something.

There are several approaches that might provide some relief for this problem but no panacea is in sight. For example, in cases where a critical piece of information is lost by its falling below threshold, the production pattern-matcher might simply take the most active element of that type as the match. This would likely lead to "interference" errors where the wrong piece of information is used by the system. On the other hand, this could lead to models that are radically difficult to manage because unwanted productions are constantly firing—inappropriate productions ought to match fairly often under this scheme. Another solution, which has been experimented with in ACT-R, is to use a "partial matching" algorithm in the production pattern-matcher, which causes the matcher to confuse "similar" declarative elements. This is a cumbersome solution in that it requires the modeler to provide similarity metrics between every single declarative element in the knowledge base. Without clear empirical evidence for what elements are "similar" to what others, this

requires a large amount of *ad hoc* estimation by the modeler.

Another approach taken by ACT-R (and probably other systems) is the use of purely stochastic mechanisms. That is, a random number generator is employed somewhere in the architecture, and slightly random behavior results, possibly leading to errors. For example, in ACT-R one can turn on a switch that will cause the activation values of individual elements to be "noisy" (this is noise in a different sense than in SPAN): when the activation of a particular element is computed, a random amount of activation is added or subtracted from the computed value. This can cause the "wrong" productions to win in conflict resolution and therefore generate errors. It would be fairly easy to add such a mechanism to SPAN, which would likely cause errors since elements that should be below or above threshold could end up on the wrong side of the cutoff.

The problem with this kind of mechanism is that it is a circular explanation for errors: Where do errors come from? The errors come from randomness in the system. And why is there randomness in the system? So that it will make errors. This explanation is hardly convincing. Even if it was not circular, it leaves a number of open questions. In the model, suppose the generator comes up with a value of -0.08. What does this mean? Do we have random number generators in our heads? Presumably not, but what cognitive process in humans does the random number generator in the system model? From what kind of distribution should the random number be drawn and why? Why not just find out what proportion of the data have errors and simply generate a binary value at the start of the trial—if it comes up 0, then the model will make an error.

Despite their prevalence, errors are probably one of the least well-understood aspects of human behavior. This is mirrored in most psychological theories and certainly reflected in current computational theories. SPAN is just as susceptible to criticism on this count as any other computational system, and it important to acknowledge this shortcoming both in SPAN and in the field in general. Thus, while SPAN has broken important new ground in modeling memory and aging, and is currently unique among production systems, it also has its weaknesses.

CHAPTER XII.

FUTURE DIRECTIONS

While the rest of this dissertation has described work that has been done on and with SPAN, this chapter is concerned with answering the "what's next?" question for SPAN. There are a myriad of directions in which SPAN could be taken, and this chapter outlines a potential research program based on SPAN. Most of the issues/directions covered here are concerned with the development of the theory, however, there are a number of technical issues that should probably be dealt with as well. Those will not be considered here because they are not relevant to SPAN as a psychological theory, nor would discussions of code and documentation issues make particularly interesting reading.

1. Further Modeling

The most obvious first step in testing SPAN is to attempt to build models of more tasks. SPAN has shown promise on the three tasks to which it has already been applied and makes a solid argument for the sufficiency of slowing. However, this case could be made considerably stronger by modeling more of the tasks researched in the areas of working memory, speed, and aging.

The most sound way to test SPAN's ability to model performance differences on cognitive tasks is to make sure that there is some way to get estimates of the gamma parameter to use in modeling the subjects performing the tasks. As of right now, the only speed-assessment task that has been modeled by SPAN is the computer-administered Digit Symbol task. Thus, the first step in extending the number of data sets to which SPAN can be applied is to construct models of some of the tasks used to assess speed, such as the

paper-and-pencil Digit Symbol substitution task, the perceptual speed tests found in the Educational Testing Service Reference Kit (Ekstrom, French, Harman, & Derman, 1976), and probably others. Once several speed tests have been modeled, it should be possible to estimate the average gamma parameter for each age group (e.g. young adults and older adults) in quite a number of studies. The next step would be to construct a model of the criterion task (or tasks), and then generate performance estimates for the different gamma values. Finally, the performance of the model at the various gamma values could be compared to the observed performance on the tasks for the different groups. To the extent that the predicted and observed values match, the SPAN model could be considered successful.

Even if perfect agreement was observed, there are always alternatives to any interpretation a SPAN model could provide for any particular data set. Even if this program was carried out successfully for thousands of tasks, it would not prove that slowing is the sole cause of age-related differences nor would it prove that SPAN is in some sense correct. This is neither expected or desired. In all likelihood, there will be tasks for which the SPAN model will fail to mimic the data, no matter how much effort is put into the model. This kind of result is not to be avoided, but sought out. This will provide information about other weaknesses in SPAN and might also provide challenges for slowing theory (though a failure for SPAN would not necessarily be a failure for other instantiations of slowing theory). Such information could help determine what changes need to be made to SPAN to make it a more realistic theory, and may help guide construction of subsequent models.

A key factor in determining how valuable future SPAN models will be lies in the selection of tasks. A wide variety of tasks should be available, but some stand out as particularly attractive candidates. One of them is matrix reasoning, such as the Raven Progressive Matrices (RPM). This class of tasks is attractive because matrix reasoning tests like the RPM are highly *g*-loaded test (Jensen, 1982) and the RPM and tests like it have been administered to young and old subjects who have also been measured on speed

(e. g. Babcock, 1994; Salthouse, 1993, 1994a). Furthermore, there is already a CAPS[25] model of the RPM (Just, Carpenter, & Shell, 1990), which provides evidence that building a production system model of the task is technically feasible and may provide the initial basis for a SPAN model. This would be an excellent test of SPAN's ability to predict age-related decrements.

Another set of tasks that has immediate appeal are the Stroop-like interference tasks used by Salthouse and Meinz (1995). The reason these tasks have a strong appeal is that these are the kind of tasks that the inhibition deficit hypothesis predicts should account for differences in working memory measures. This is what was found in the Salthouse and Meinz study, but they further found that speed measures and inhibition measures were related, and that speed measures accounted for more of the observed differences in working memory than did inhibition measures. If SPAN can provide a computational account of these results, this lends additional credence to the argument that it is speed, and not inhibition deficits, that are the root cause of age-related differences in working memory. This also provides an excellent opportunity to test the assumption made by SPAN that inhibition (negative activation) should be handled no differently than positive activation, which is one of the places where SPAN and CAPS disagree. This would not be a definitive test, but it would be the first test of its kind for inhibitory mechanisms.

2. Perception, Action, and Attention

SPAN as is stands now is primarily a theory of cognitive performance, with motor and perceptual/attention performance largely handled by abstractions and approximations. However, to construct models that are truly accurate approximations of human performance, issues in perception, action, and attention do need to be addressed. This is especially true in SPAN for perception/attention issues, since perception/attention determine what is available to SPAN in the form of external memory. To make SPAN truly

[25] This model was constructed with a older version of CAPS that was not capacity-constrained, so the Just, Carpenter, and Shell (1990) model does not provide a clear model of the effects of working memory capacity on matrix reasoning performance.

complete and able to handle interaction with the world, a model of what and when information from the world is available is critical. It would be an even larger step toward making SPAN a true “cognitive architecture” if this were accompanied by a theory of how and why as well as what and when. Obviously, generating a complete theory of attention, perception, and action and how that might work in the context of a production system is a tremendous task—it is probably better to measure the expected effort not in person-years, but in person-decades.

Fortunately, some of those person-decades have already been spent. In the last several years, considerable effort has been directed toward issues in perception, action, and attention in the context of production systems. Work has already been done with Soar (Weismeyer, 1992), and ACT-R (Anderson, Matessa, & Douglass, 1995), and these issues are the major focus of the EPIC system (e.g. Kieras & Meyer, 1994). For the purpose of extending SPAN, there seems to be little reason to reinvent the wheel. Since many of the ideas already incorporated in SPAN were borrowed, it is not at all strange to propose that more borrowing should take place.

In particular, SPAN should be able to borrow a great deal from EPIC, as briefly discussed in Chapter 11. The issues in integrating the perceptual/motor machinery found in EPIC with the cognitive machinery in SPAN are primarily technical, not theoretical. It would not be a trivial technical endeavor but should be a feasible one—this is based in no small part on the minimal assumptions made in the current cognitive processor at the heart of EPIC, PPS. Since the rest of the system makes few assumptions about what PPS can or cannot do, EPIC should be integrable with almost any other production system. This is actually a great opportunity, since Soar was very recently Soar was successfully integrated with EPIC. This work is so new that no interesting results have yet been generated, but it offers an interesting opportunity to compare Soar and SPAN if SPAN could be integrated with EPIC in this way.

Thus, the research program for SPAN in the areas of perception, attention, and action is primarily one of integration and exploration, not discovery. This is not to say that

there is no potential for discovery with SPAN. Because SPAN has a unique set of mechanisms for management of working memory, it may be the case that the speed and decay mechanisms of SPAN will provide unique insight into some of the issues in perception and action. It is far too early to tell if this will be the case, though, but there may well be some interesting opportunities in this area.

3. Learning

Some discussion of learning has already taken place at various points throughout this document. It is clear that learning affects human performance in a variety of ways and this includes even simple tasks like the Digit Symbol task. One of the previously-noted weaknesses of SPAN is that there is no learning of any kind. This is not unique to SPAN—there is no learning in CAPS or EPIC, either—but the lack of learning makes SPAN seem incomplete as a general cognitive architecture. Thus, learning seems a good candidate for inclusion in the research program.

The first question to answer about learning in SPAN is What can be learned? There are four things that SPAN could potentially learn: new declarative memory elements (DMEs), new productions, biases for declarative memory elements, and strengths for productions.[26] There are different possibilities for each, so they will be considered separately. Again in the spirit of not trying to reinvent the wheel, there are places in SPAN where learning mechanisms can probably be borrowed rather than invented.

3.1 Learning new declarative memory elements

This is probably the most wide-open area for learning. The two major production systems that include learning mechanisms, Soar and ACT-R, have different approaches to the problem. The Soar approach, called “data chunking,” is better developed (see Rosenbloom, Newell, & Laird, 1991). The process is complex and for reasons of brevity will not be described here, but it might be possible to adapt the process for use in SPAN.

[26] One might also think about learning modifications to existing declarative memory elements and productions, but it is not clear that would be any different than learning a new production or a new DME.

Some modifications would certainly be necessary because of the different styles of knowledge representation—there are no independent DMEs in Soar—and Soar's reliance on problem spaces, but these issues are probably more technical than conceptual. In ACT-R, new DMEs can be created in the system as the result of the action of a production or by "encoding from the environment," but neither of these are specified in much more detail than that.[27] Some kind of learning mechanism for new declarative information is probably desirable, but the exact form such a mechanism would take is still an open research question.

3.2 Learning new productions

Unlike learning declarative memory elements, the question of how new productions are learned has been researched extensively by both the Soar and ACT research efforts. These learning mechanisms are concerned with learning a certain kind of production that will be called a *control production*. Control productions are productions that are concerned with what the system should do next, such as what the appropriate step to do in an addition problem at a particular point. This kind of learning is most closely associated with skill acquisition, though it is not restricted to the development of skill. This kind of knowledge is captured in both Soar productions and ACT-R productions, as it is in SPAN productions. The Soar chunking mechanism is probably not appropriate for SPAN because it is closely tied to Soar's universal subgoaling mechanism. The ACT theories have a long history of dealing with this kind of learning and there should be no particular difficulty in adopting the composition and compilation learning mechanisms described for ACT (e.g. Anderson, 1983).

However, there is a kind of production learning, probably not covered by the ACT mechanisms, that would likely be important to have in SPAN (it would be appropriate for CAPS as well). This concerns *associative productions*, or productions that express the relatedness between DMEs. This kind of production is not necessary in any of the flavors

[27] Ongoing work with the "visual interface" in ACT-R is attempting to address this question, but clear results have not yet been reported.

of ACT, since DMEs in ACT are part of an automatic spreading activation network. Since no such network exists in SPAN, associations between DMEs would have to be manifested as productions. For example, for "doctor" to prime "nurse," there would have to be a production that explicitly propagates some activation from "doctor" to "nurse." There are a number of ways this could be handled in SPAN, such as including an automatic activation network (a la ACT) or by some kind of associative production learning mechanism.

There are a number of technical considerations such solutions would raise. As an example, consider the situation if a weak default associative production was created the first time two DMEs co-occurred above threshold. The strengths of these productions would themselves decay over time but could be incremented according to some Bayesian or correlational formula each subsequent time the pair co-occurred. This approach has the advantage that it keeps track of associations as they occur in the system's environment, in a manner similar to the rational analysis underlying ACT-R. However, unlike in ACT-R, the activation of associates would be affected by the same factors that affect all other activations, e.g. noise and speed. Whether or not this is the right approach is not entirely clear, since results of priming studies with older adults are mixed (see Light, 1991 for a review). This method has the further pragmatic disadvantage that it will mean SPAN models will have to contain a rather large number of productions. However, the best current pattern-matching algorithms (e.g. Rete/UL, Doorenbos, 1995) have been shown to run rapidly even with millions of productions, so this is not necessarily a serious disadvantage.[28]

3.3 Learning declarative memory element bias

The bias of a DME determines how fast that element gains activation per unit of input activation. The bias parameter for a particular element is supposed to represent frequency of use; frequently-used elements should have a higher bias than those that are

[28] There is another solution, which is to have associations run automatically as in ACT-R, but simply make the activation rate sensitive to the gamma parameter and noise.

used less often. This suggests a straightforward learning algorithm: increment the bias of an element every time it is used. But what constitutes a “use” of a declarative memory element? Fortunately, this is relatively easy to define for a production system: a DME can be considered “used” every time it is matched by the left-hand side of a production. Of course, this does not resolve every issue with respect to learning element biases, there are several research questions which remain open. By how much should an element’s bias be incremented when it is used? Should that amount be constant or should it be a function of the element’s present bias. Should there be forgetting of bias over time (that is, should bias decay)? Should there be a maximum bias? How about a minimum bias? All of these questions are questions that a research program directed at learning issues needs to address. A good starting point might be the learning algorithms in ACT-R that govern base-level activations, but again, this is an open question.

3.4 Learning production strengths

Production strengths in SPAN control how much activation a production propagates when it fires. The idea is that more “trusted” productions should have higher strengths. What does a production have to do earn trust? It must show that it is useful by repeatedly producing successful results for the system. Thus, trust is a function of not simply frequency of use, but frequency of successful use. This is the classic “credit assignment” problem in AI. Several approaches have been found to work well in this problem, and one approach has been implemented in the context of production systems. This is Holland’s (e.g. Holland, Holyoak, Nisbett, & Thagard, 1986) “bucket brigade” algorithm. The technical details of the algorithm are not of interest here, but the general concept is simple enough. Whenever a goal is satisfied, the production that fired to satisfy it is given some credit. That production then passes some of the credit to the productions that created (or activated) the elements that triggered it. Those productions also pass some of the credit backwards, and then the next set passes backwards, and so on. Strengths are then incremented according to the amount of credit a production has received. This kind of learning has parallels in other systems, as it is not unlike backpropagation in PDP systems

(Rumelhart & McClelland, 1986) and the Bayesian updating done in ACT-R.

The same kinds of questions that were asked for bias learning would need to be answered for production strength learning: how much should strength be incremented, does it decay over time, is there a limit, etc. However the numbers work out, though, this kind of learning mechanism in combination with bias learning should make oft-used productions and elements stronger, and therefore more robust and less noise-generating, as discussed in Chapter 7.

3.5 Other areas of exploration

Once learning mechanisms have been incorporated into SPAN, there are several areas of exploration that might be pursued. Besides investigating how learning affects simple tasks like Digit Symbol, SPAN offers a unique perspective on novice/expert differences in memory. As mentioned in Chapter 5, experts seem to have higher functional memory capacity in their domain of expertise. The noise mechanism in SPAN was designed with this in mind and, with the inclusion of learning mechanisms, might be a fruitful avenue for exploration.

Another phenomenon that has received little attention and has certainly gone unmodeled is the relationship between working memory load and learning. There is some evidence that increased memory load hurts learning performance for both procedural and declarative information (e.g. Reber & Kotovsky, 1991; see Baddeley, 1986, chapter 3, for a review), but this phenomenon is not well-understood. Tying the effectiveness of the learning mechanisms to what amounts to an index of memory load, SPAN's noise value, may provide insight into how memory load affects learning performance. There is room for a great deal of theoretical and empirical work in this area, so any contribution SPAN could make would be a useful one.

There are a myriad of other issues, both in learning and in other areas, in which SPAN might be a productive research vehicle. This chapter has only outlined the one vision for SPAN. This vision is a research program of some scale that will surely take years to execute. Certainly, in those years there will be some roadblocks and failures, but

hopefully some important learning and progress as well.

4. Working Memory and the External World

Though not discussed in great detail up to this point, "individual differences in capacity" is not the only way in which working memory has been used to explain performance differences; working memory demand has also been used to explain the differences between tasks themselves. For example, the demands placed on working memory by the task instances themselves can differ and yield performance differences on isomorphs of the same general task. Probably the best illustration of this is in the Tower of Hanoi (TOH) puzzle. In this domain, the abstract description of the puzzle as a problem space is small and well-specified. What is interesting about the TOH here is that, though holding the problem space constant, puzzle difficulty can be manipulated by providing isomorphs which contain different amounts of information in their external manifestation (Kotovsky, Hayes, & Simon, 1985; Zhang & Norman, 1994). Those isomorphs that explicitly represent the puzzle state and the legality of moves are much easier for people to solve, presumably because people solving the easier isomorphs have less information to manage in working memory.

This line of research has not received much attention, though, probably because most laboratory tasks are designed to minimize working memory load by providing good external representation (Newell & Simon, 1972). However, it has recently become popular to criticize computational models of cognition for being poor at modeling dynamic interaction with the external world (e.g. Suchman, 1987; Greeno & Moore, 1993). It may be that a clearer understanding of working memory constraints can help provide a bridge between traditional cognitive models and the external world. How might this be possible? If working memory load helps determine performance, and the environment helps determine working memory load, then taking the two together should improve our ability to predict performance on a wide variety of cognitive tasks. In fact, there are at least two possible ways in which the environment affects working memory: through external

representation and through proceduralization.

A little has already been said about external representation, but more detail can be provided. Given a working memory system that has some kind of limitations, such SPAN, if a working memory element could be made available yet not decay or be displaced, tasks requiring that element should be easier. This is precisely what external representation can provide, by "storing" representations and by reducing processing demand—exactly the function of the pegs and disks in the standard version of the TOH. And, as one might expect, the more an isomorph represents externally (as opposed to isomorphs which force people to store puzzle state and information about move legality in working memory), the easier that isomorph is.

Take as another example of the power of external representations the task of multiplying large numbers. Even when using the same algorithm, external representation can make a huge difference—consider multiplying 45,692 by 89,763 without the use of an external representation, such as writing on paper. For most people, this is an incredibly difficult (if not impossible) task because it creates a tremendous working memory demand. Using paper as an external representation helps make the task tractable, because it affords the "free" storage of partial products.

It is probably possible to "offload" processing as well as representation to the external world, such as using a calculator to perform the previously mentioned multiplication.[29] This can reduce the load on working memory in several ways: first, many computations generate partial products, which are a WM burden; second, to the extent that the external process is faster than the internal one, there is less decay for elements in WM that are waiting for the results of the computation. Finally, the computations themselves take up "space" in working memory, which increases the load on working memory, as mentioned before.

The second way in which the environment and working memory may interact is

[29] It should be noted that the "external world" in this context includes other people; see Hutchins (1990) for an excellent example of multiple people "sharing" a large process which, in total, no one person could do but a group of people can divide amongst themselves and their tools to make tractable.

through proceduralization. Proceduralization is a kind of speedup learning in which declarative knowledge is re-represented as procedural knowledge yielding more rapid, fluent execution of a process. The claim is that when knowledge is proceduralized, the burden on working memory is reduced during procedure execution. The critical link in understanding the relationship between the environment and working memory load is that the environment of the learner has a large impact on exactly what it is that gets proceduralized. Thus, understanding someone's environmental history is critical to understanding what will and will not burden their working memory.

The first piece of this argument concerns the reduction of working memory load by proceduralization. How does this occur? Proceduralization produces efficient compiled procedural knowledge, eliminating the need to interpret more static declarative knowledge and allowing the procedure execute in fewer steps (Anderson, 1983). Thus, less declarative knowledge needs to be present in working memory for a proceduralized process to execute. This is often the case with experts in a domain, and may be why it is difficult for them to verbalize what it is they are doing (e.g. Ericsson & Simon, 1984). The elimination of declarative knowledge in working memory reduces the burden on working memory, since fewer active elements are necessary. In addition, since compiled procedures are faster, any declarative knowledge that is present has less time to decay and so is more likely to be available when needed. It has been empirically demonstrated that people's performance on highly proceduralized tasks is robust to additional working memory loads (Strayer & Kramer, 1990; Baddeley, 1986).

So if proceduralization can reduce working memory load, how important a kind of learning is proceduralization? One of the most central, according to the literature. Anderson (1983, 1993) calls proceduralization "knowledge compilation" and provides both empirical and theoretical reasons for considering it crucial for the human cognitive system. Card, Moran, and Newell (1983) present a framework for understanding human-machine interfaces and maintain that the most useful characterization of an interface is the procedural knowledge required to operate it. Continuing on this theme, the Soar system (Newell,

1990) provides a single learning mechanism, chunking, which is a very general form of proceduralization. The centrality of proceduralization to learning is firmly entrenched in both the data and theories of psychologists.

The key remaining question in understanding the proceduralization link between working memory and the environment is "what gets proceduralized?" Quite often, what is proceduralized is a function of the learner's environment. People proceduralize the constraints and affordances provided by the world around them. Arguments to this effect have been made repeatedly by a wide variety of researchers in a wide variety of domains, of which the following is but a sample. Vicente and Rasmussen (1990) provide evidence that people learn procedures more effectively when environmental constraints are made obvious. This is closely related to the conclusions of Kitajima and Polson (1992) in their Construction-Integration (Kintsch, 1988; Mannes & Kintsch, 1990) model of novice use of computer software. This suggests that when environmental constraints are difficult to find, proceduralization is poor. On the other hand, when proceduralization is good, the resulting behavior is fluid and characteristic of experts. Accordingly, Kirlik, Miller, and Jagacinski (1993) found that experts' behavior in their domain could be better predicted by a model of the environmental constraints, rather than a model of the experts (numerous models of the experts alone were tried without much success). This is in agreement with Casner's (1994) study, which found that the procedures used by airline pilots could also be well-predicted by a model of the plane's situation. Anderson (1990) maintains that not only proceduralization, but almost every aspect of the human cognitive system, is constructed to optimally match environmental demands. Learning environmental constraints is the cornerstone of both Skinner's (1966) and Suchman's (1987) view, further underscoring the importance of understanding environmental constraints.

Further support for the importance of proceduralized environmental constraints can be found in the human error literature. Reason (1990), Baars (1992), and Norman (1981) describe a kind of error they refer to as a capture error, which is said to occur when a person executes a procedure that is not the correct procedure to satisfy their goal, but is

consistent with the environmental cues present. Baars goes a step further, and claims that most errors are "liberated automatisms," or automatic procedures cued by a person's environment and detached from any goal structure. These error data suggest the importance of understanding proceduralization in reference to environmental features, which ultimately is related to working memory load.[30] To the extent that the environment determines what gets proceduralized and proceduralization determines working memory efficiency, the environment plays a role in determining working memory load.

A research program investigating the role of external memory need not be specific to SPAN; however, because SPAN is designed to address working memory constraints, SPAN may be a useful testbed for ideas about how external memory and working memory interact. Details of this interaction are not well-understood, thus any insight that could be provided by SPAN is potentially useful to the larger effort in this area.

[30] Error data (e.g. Just, Carpenter, & Hemphill, 1996; Byrne & Bovair, in press; Reason, 1990; Baars, 1992) also suggest that goal loss is related to memory load. This implies that goal management is susceptible to memory load and is not a privileged process as some systems (e.g. ACT-R) claim.

CHAPTER XIII.

SUMMATION

1. Review

The construct of working memory, in one form or another, has a long history in cognitive psychology. As it is seen today, working memory may be thought of as activated long-term memory, goals, and partial products, as well as active processing. It also has some kind of boundary on total activity that varies between individuals. Both cognitive science and cognitive aging have been concerned with working memory, though from different perspectives. In the computational modeling community, concern with working memory has been primarily theoretical, with working memory playing a kind of workspace role in the underlying cognitive architecture. In cognitive aging, working memory has largely been seen as the source of age-related decline in "fluid" performance. Differences in working memory capacity have been used to explain differences in performance on a wide array of different tasks, both within and between age groups.

Despite its prominence, the concept of working memory has not been particularly well-developed—there have been few theories proposed of how working memory actually functions, and even fewer that have been specified in enough detail to be implemented as computational models. Yet we know some of the important features of how working memory functions: items decay and are displaced by both storage and processing, speed of processing appears closely related to working memory capacity, and increased knowledge can lead to domain-specific functional increases in capacity.

SPAN is a computational theory of working memory designed to incorporate all these principles into a coherent system. SPAN is a production system in which all

declarative elements have an activation value that decays over time. Elements have to be above a threshold to be matched by productions, and it is productions which propagate activation and take action in the system. The amount of activation propagated by a production is a function of two things: (1) total system noise, which is a function of the activity levels of the declarative memory elements, and (2) a global propagation rate parameter, gamma. Decay and propagation of activation are fundamentally at odds and a decrease in propagation rate due to either slowing of the individual or task demands (i.e. noise) will make it easier for declarative memory elements to be lost. Thus both individual differences and task demands affect functional memory capacity.

In order to demonstrate that SPAN is a viable theory of memory and aging, models of three tasks were constructed. The first of these, the classic "fan effect" task, modeled data from an experiment contrasting young and old (Gerard, et al., 1991) on a well-established memory phenomenon. Since SPAN is a model of memory, this provided a good starting point for the model. The model successfully demonstrated not only the basic fan effect, that is, increased reaction time associated with higher connectivity between propositions, but also an increase in fan effect for the older subjects. The model demonstrated not just simple a qualitative difference, but simulated the magnitude of the observed differences as well.

The second and third models were based on the Salthouse and Coon (1994) data set and consisted of a model of a standard speed measure and a standard working memory measure. Since the slowing theory of working memory predicts that decreased speed should produce decreased working memory performance, these tasks provide an excellent test for the theory. The speed measure modeled was Salthouse's computer version of the Digit Symbol task. The Digit Symbol model did indeed show that propagation rate could affect the reaction time on this task. By constructing a less stable (i.e. less active) initial representation of the target symbol, slower versions of the model were more likely to lose their representation of the target and thus take extra time in performing additional eye movements and rehearsal. In order to model the magnitude of the young-old difference

found in the Salthouse and Coon data set, a 20% difference in propagation rate was required.

SPAN was also used to model a fairly standard working memory measure, Computation Span. This is a more complex task than either of the first two, requiring subjects to both solve arithmetic problems and remember digits for later recall. The SPAN model handles the arithmetic problem by retrieving arithmetic facts and matching them to the answers presented on the screen and handles the recall task by rehearsing the digits. As the task is made more difficult by increasing the span length, the model's performance degrades because the system noise generated by the task demands overwhelms the system's ability to propagate adequate amounts of activation. Slower versions of the model are overwhelmed earlier and thus score lower on the memory measure. Results of the simulations showed a clear quantitative link to the Digit Symbol task, as the young-old difference in the data set were again matched with a 20% change in the propagation rate parameter.

These models raise a number of interesting issues. First and foremost, they demonstrate the computational sufficiency of the slowing mechanism.That is, they prove that slowing is capable of producing not only the kind but the degree of performance decrement associated with normal aging in three different tasks. Second, these tasks demonstrate SPAN's uniqueness among production systems, as it is not at all clear that the other four prominent systems (CAPS, Soar, ACT-R, and EPIC) could generate the observed differences. The SPAN models also raise some interesting issues with respect to error data, since the models do not make errors in any of these tasks. Errors arising from the architecture (vs. arising from knowledge deficits or random number generators) are difficult to find in computational models, and SPAN is no exception.

Finally, some discussion of the future of SPAN was presented. There are three major areas for future development of SPAN. Most centrally, SPAN should to be applied to more tasks for which speed estimates are available for the subjects. Pushing SPAN into other task areas will provide more opportunities to test the limits of the theory and could

potentially provide additional insight into the relationship between slowing, cognitive performance, working memory, and age. SPAN could also use some work in the perception/action area, potentially borrowing from EPIC to provide a richer set of constraints. Finally, the relationship between learning and working memory could be further explored by examining the addition of one or more learning mechanisms to SPAN.

2. Discussion

This work has raised a number of issues in cognitive aging, research in working memory, and cognitive science (particularly computational modeling). All of these merit further discussion in light of the results of the research, as does SPAN itself.

2.1 Cognitive Aging

For cognitive aging, SPAN has demonstrated that the time for formalization of theories is now. The call made by Salthouse (1988) for formalization is certainly heedable, and not just at the abstract level provided by some earlier simulations. Specific theories of cognitive aging can be formally constructed and compared to results of particular experiments. The aging literature is rich with empirical data and fairly lean in terms of the development of theory—many terms are still used in a vague way, many constructs poorly specified. While it is true that the bulk of the research in cognitive aging has gone on in the past decade or so, the field is not still so exploratory that a poor level of specification is still necessary. SPAN stands as an example of the specificity that is possible.

Of course, SPAN is more than just a specification, it is a specification of a particular theory of working memory. That theory is an instantiation of the general slowing theory most closely associated with Salthouse (particularly Salthouse, 1996). This theory states that the key factor contributing to a wide variety of performance differences is speed, operationalized in SPAN as a global propagation rate parameter. SPAN represents an advance for the slowing theory, in that slowing is now the only theory in cognitive aging for which there is a fully-specified computational model capable of modeling the quantitative young-old performance differences found in the laboratory. Slowing has been

shown (Salthouse, 1996, provides a good review) to be sufficient to explain a variety of aging data from the statistical standpoint (i.e. controlling for speed attenuates age differences), and now slowing has been shown to be computationally sufficient as well. This is a powerful combination that puts slowing in a unique position. Any theory that would claim to explain age-related changes in cognition (e.g. inhibition deficits) should now have to prove both statistical and computational sufficiency, which is a substantial undertaking.

Of course, slowing in SPAN is of a very particular kind. First, the rate parameter in SPAN applies to all productions at all times. This does not imply that performance on every task should show the same degree of slowing; as the models demonstrate, different task demands produce different signal-to-noise dynamics and therefore different levels of slowing. The fact that the gamma paramter is uniform within an individual does not imply that there are only general decrements in performance. As has been discussed, strategies and knowledge can be employed to affect performance, even in the presence of a lower propagation rate. Exactly what task-specific factors exist beyond general slowing is still an open empirical question in a number of domains—it may be possible to use SPAN as a tool to help uncover where general vs. specific deficits come into play.

Second, the rate parameter in SPAN applies only to cognitive operations—it has no direct effect on the time course of perceptual or motor operators. This is not to say that is has no effect on the time course of these operations; in general, perceptual/motor operators such as those employed in EPIC or CPM-GOMS models are preceded by one or more preparatory cognitive operations and those preparatory operations will be slowed along with the more central cognitive operations. Of course, this perspective may turn out to be implausible, but there is no fundamental commitment in SPAN to postulating only one kind of slowing. The perceptual and motor processes could be slowed as well, which would contribute to an overall degradation of system performance. While SPAN presents a fairly unitary picture of slowing, it is not limited to this perspective.

Most importantly, SPAN has demonstrated, for cognitive aging in general and

slowing theory in particular, that computational modeling provides a useful way of formalizing and testing theory. Even if SPAN turns out to be flawed in many of its particulars, this contribution should stand out as the central contribution of SPAN.

2.2 Working Memory

SPAN also has a particular perspective to offer on some of the issues in working memory. One of the more interesting features in this regard is that there is no real notion of "capacity" in SPAN. Simply put, such a notion is not necessary. In SPAN, capacity is functional, not stuctural, and this functional capacity emerges based on the demands of the task, the global propagation rate, the strategy employed, and the knowledge brought to bear. All of these factors contribute to SPAN's performance on any specific task. This perspective on working memory as functional rather than structural has several interesting implications.

First, this suggests that the best general working memory measures should be tasks that are knowledge-lean, not amenable to rehearsal, and have a minimum of different ways to complete the task (to minimize strategy differences). Because particularly strong productions in SPAN yield a certain level of imperviousness to working memory demands, this suggests that knowledge-intensive tasks used to assess working memory capacity are tapping something else along with the general source of differences. A good example is Daneman and Carpenter's (1980) Reading Span test. Because there are wide individual differences in the amount of practice with reading and related knowledge (e.g. word knowledge), the Reading Span test measure probably confounds the general speed factor and reading knowledge. Of course, this probably makes Reading Span a better predictor of performance than a more general working memory test for memory-intensive reading tasks such as those described in Just and Carpenter (1992). If one is concerned with performance in a particular domain, this kind of confounded measure is probably advantageous.

On the other hand, the best *general* working memory assessment technique is almost certainly not a single measure, but rather several that employ different kinds of

materials, preferably making use of multiple modalities. The variance shared by a group of these measures is probably the best way of assessing general working memory functioning. This is hardly a revolutionary idea, but it is consistent with the way SPAN is formulated—one general speed parameter and then numerous small and specific parameters all contribute to overall performance. If the influence of the small and specific factors can be statistically controlled, what remains should be a reflection of the influence of the speed parameter.

Interestingly, the criteria SPAN suggests for a test to be the best general measure of working memory functioning—knowledge-lean, hard to rehearse, minimal strategy differences—are probably best met within a single tests by matrix reasoning tests like the Raven Progressive Matrices. The visual patterns are indeed difficult to rehearse, there is little in the way of other domain knowledge that is likely to help much, and the test is in principle simple enough that there should be minimal strategy differences. What is striking about this expectation is that matrix reasoning tests are indeed some of the most highly *g*-loaded tests we know of (Jensen, 1982). Given the general influence slowing has on SPAN's performance, the widespread effects of age on cognitive functioning, and the performance associations between working memory and other cognitive measures, it is not much of a stretch to think of working memory and therefore speed as our best understanding of general individual differences in cognitive performance, and therefore something like *g*. This is, of course, somewhat speculative, but it is an intriguing prospect.

Another interesting issue for SPAN is raised by the SPAN approach to rehearsal. Rehearsal in SPAN is conceptualized as a process, primarily perceptual/motor, rather than a separate structure as in Baddeley's conception of working memory. In general, SPAN represents a very different way of viewing working memory than many other conceptions. In SPAN, working memory "capacity" is the result of the interaction of a unitary/general speed factor and potentially very numerous task and knowledge factors. There is no obvious need to postulate the existence of separate memory systems or resource pools, though there may be particular processes (such as rehearsal) which can affect performance.

Rather than an “articulatory loop” structure, there is a process by which memory elements are “refreshed” by what amounts to self-directed speech. This process creates noise, just as any other cognitive process would, and can thus seem to be resource-demanding. At the same time, the process of rehearsal has limitations because of the temporal constraints of the system (i.e. decay). This approach to working memory—and memory in general—of looking for an explanation first in terms of the processes involved seems more parsimonious than what might be called “Tulvingism,” or the tendency to postulate a new memory system when a memory performance difference is uncovered.

2.3 Cognitive Science

Within the computational modeling community, modeling of individual differences has a long history. However, the source of those differences has almost universally been attributed to differences in knowledge. What SPAN and the models presented here demonstrate is that it is possible to locate significant individual differences in the architecture. Work in cognitive aging suggests not only that it is possible to do so, but that it is important to do so. There is a great deal of detailed quantitative data regarding individual differences in the empirical literature that has been long ignored by computational modelers. Perhaps these data have been considered unapproachable with computational techniques, but SPAN demonstrates that attempting to deal with such data is a tractable research problem. There is no longer any compelling reason that the scope of computational models could not reach further into the individual differences literature.

Working memory is a natural place for this effort to start. Cognitive science has always been deeply concerned with memory, even if working memory in particular has not always been the focus of computational modeling efforts. Taking working memory constraints more seriously could help bring the computational modeling and cognitive aging communities together, which could be a potentially fruitful union for both. Computational modeling efforts need to broaden their appeal and take on more and more general kinds of problems, and the extensive data gathered by researchers in cognitive aging seems a prime target. The aging literature has a number of striking regularities, such

as the general decline in fluid abilities, while at the same time hosting a number of controversies about underlying mechanisms that computational models are an excellent position to address.

2.4 SPAN

SPAN as a theory has both strengths and weaknesses. On the down side, SPAN is still fairly primitive in its handling of perception and action, which makes detailed latency modeling of higher-level tasks such as the Computation Span difficult. Learning is still something of a weak point as well, given that there is some learning going on even in simple tasks like Digit Symbol. SPAN is also fairly cumbersome to work with from a technical standpoint—it is hardly polished software. And, like all other computational systems, task analysis and model-building are somewhat underdetermined.

On the other hand, SPAN has demonstrated a great deal of promise. SPAN provides the first formalization of slowing as mechanism for working memory in a production system context, and has shown that it can be used to construct reasonably accurate quantitative models of some of the old vs. young differences found in the cognitive aging literature. This represents at least a moderate success in bridging the gap between cognitive science and cognitive aging, with many potential gains down the road. In fact, potential may be the greatest strength of SPAN. The work laid out in this thesis represents not the completion of a research program, but the beginning of one. There are a number of research paths that can be followed from this starting point, and it is hoped that at least some of them can be fruitfully explored over the next several years.

APPENDIX A.

FAN EFFECT MODEL SOURCE CODE

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : (c)1995 Michael D. Byrne, All Rights Reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : fan-effect.span
;;; Version     : 1.0
;;;
;;; Description : First attempt to model the fan effect in SPAN.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package :span)

(load "SPAN:default-DMtypes.span")

(defDMtype comprehension subject location)
(defDMtype fact subject location)
(defDMtype recognition value)
(defDMtype visible-word is index)
(defDMtype lex-entry ^word)
(defDMtype garbage tag)

(load "SPAN:default-productions.span")


(p finished!
   (goal ^is respond ^status satisfied)
-->
   (halt))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Parsing
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(p parse-complete
   {(goal ^is parse ^status active) <g>}
   (comprehension ^subject <s> ^location <l>)
   (lex-entry ^word <s>)
   (lex-entry ^word <l>)
-->
   (modify <g> ^status satisfied))



(p lexical-access-subj
   (comprehension ^subject <s>)
   {(lex-entry ^word <s>) <e>} < 0.0
   - (fact)
-->
   (transmit 1.5 <e>))


(p lexical-access-loc
   (comprehension ^location <l>)
   {(lex-entry ^word <l>) <e>} < 0.0
   - (fact)
-->
   (transmit 1.5 <e>))


(p make-comprehension
   (goal ^is parse ^status active)
   (visible-word ^is <a> ^index 2)
   (visible-word ^is <l> ^index 6)
   - (comprehension) >= -1.0
   -->
   (make comprehension ^span-pool internal ^span-act -1.0
                       ^subject <a> ^location <l>))


(p activate-comprehension
   (goal ^is parse ^status active)
   {(comprehension) <c>} < 0.0
   -->
   (transmit 0.8 <c>))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Recognition handling
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(p recognize-complete
   {(goal ^is recognize ^status active) <g>}
   (recognition)
-->
   (modify <g> ^status satisfied))


(p refresh-comprehension
   (goal ^is recognize ^status active)
   - (goal ^is parse ^status active)
   {(comprehension) <c>} < 0.0
-->
   (transmit 1.0 <c>))


;;; Productions that transmit activation from the comprehension to the
;;; relevant facts.

(p find-subject
   (goal ^is recognize ^status active)
   (comprehension ^subject <a>)
   (lex-entry ^word <a>)
   - (fact)
   {(fact ^subject <a>) <t>} < 0.0
-->
   (transmit 1.0 <t>))


(p find-location
   (goal ^is recognize ^status active)
   (comprehension ^location <l>)
   (lex-entry ^word <l>)
   - (fact)
   {(fact ^location <l>) <t>} < 0.0
-->
   (transmit 1.0 <t>))


;;; If there's a fact that matches the comprehension, then it's recognized.

(p positive-recognition
   (goal ^is recognize ^status active)
   (comprehension ^subject <a> ^location <l>)
   (fact ^subject <a> ^location <l>)
   {(recognition ^value positive) <t>} < 0.0
-->
   (transmit 1.0 <t>))
```

```
;;; On the other side, if we have a fact and that fact doesn't match, it's
;;; a "no"

(p negative-recognition
   (goal ^is recognize ^status active)
   (comprehension ^subject <a> ^location <l>)
   (fact ^subject <a> ^location <> <l>)
   {(recognition ^value negative) <t>} < 0.0
-->
   (transmit 1.0 <t>))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Finally, the action productions
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(p respond-yes
   (goal ^is press ^status active)
   (recognition ^value positive)
   {(command ^do press-yes) <t>} < 0.0
-->
   (transmit 1.0 <t>))


(p respond-no
   (goal ^is press ^status active)
   (recognition ^value negative)
   {(command ^do press-no) <t>} < 0.0
-->
   (transmit 1.0 <t>))


(p refresh-recognize
   (goal ^is press ^status active)
   {(recognition ^span-act <a>) <t>} < 0.0
   - (recognition ^span-act {>= <a>}) < 0.0
-->
   (transmit 1.0 <t>))


(p press-complete
   {(goal ^is press ^status active) <g>}
   {(goal ^is respond ^status active) <tg>}
   (command)
-->
   (modify <g> ^status satisfied)
   (modify <tg> ^status satisfied))
```

```
(p refresh-recognition
   (goal ^is press ^status active)
   - (recognition)
   - (comprehension)
   - (fact)
   {(recognition) <t>} > -1.0
-->
   (transmit 1.0 <t>))


(p trap-end
   (command)
-->
   (halt))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Set up declarative memory
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(eval `(make goal ^span-act ,(random 1.0) ^is respond ^status active))
(eval `(make goal ^span-act ,(random 1.0) ^is press ^parent respond ^status active))
(eval `(make goal ^span-act ,(random 1.0) ^is recognize ^parent press ^status active))
(eval `(make goal ^span-act ,(random 1.0) ^is parse ^parent recognize ^status active))

(eval `(make garbage ^span-act ,(random 1.0) ^tag 1))
(eval `(make garbage ^span-act ,(random 1.0) ^tag 2))
;(eval `(make garbage ^span-act ,(random 1.0) ^tag 3))

(make command ^span-pool motor ^span-act -1.0 ^do press-yes ^tag press)
(make command ^span-pool motor ^span-act -1.0 ^do press-no ^tag press)

(make recognition ^span-act -1.0 ^value positive)
(make recognition ^span-act -1.0 ^value negative)

;; doctor in zoo is target 1-1
;; fireman in train is target 2-2
;; artist in house is target 3-3
;;
;; doctor in hood is foil 1-1
;; fireman in store is foil 2-2
;; hippie in house is foil 3-3
;;

(make fact ^span-act -1.0 ^subject doctor ^location zoo)
(make fact ^span-act -1.0 ^subject fireman ^location train)
(make fact ^span-act -1.0 ^subject fireman ^location hood)
(make fact ^span-act -1.0 ^subject artist ^location house)
(make fact ^span-act -1.0 ^subject artist ^location train)
(make fact ^span-act -1.0 ^subject artist ^location store)
(make fact ^span-act -1.0 ^subject writer ^location house)
```

```
(make fact ^span-act -1.0 ^subject dude   ^location house)
(make fact ^span-act -1.0 ^subject dude   ^location store)
(make fact ^span-act -1.0 ^subject hippie ^location blah)
(make fact ^span-act -1.0 ^subject hippie ^location blah2)
(make fact ^span-act -1.0 ^subject hippie ^location blah3)


;;; Lexicon entries

(make lex-entry ^span-act -1.0 ^word lawyer)
(make lex-entry ^span-act -1.0 ^word plumber)
(make lex-entry ^span-act -1.0 ^word teacher)
(make lex-entry ^span-act -1.0 ^word boat)
(make lex-entry ^span-act -1.0 ^word park)
(make lex-entry ^span-act -1.0 ^word church)
(make lex-entry ^span-act -1.0 ^word bank)
(make lex-entry ^span-act -1.0 ^word doctor)
(make lex-entry ^span-act -1.0 ^word zoo)
(make lex-entry ^span-act -1.0 ^word fireman)
(make lex-entry ^span-act -1.0 ^word artist)
(make lex-entry ^span-act -1.0 ^word writer)
(make lex-entry ^span-act -1.0 ^word dude)
(make lex-entry ^span-act -1.0 ^word hood)
(make lex-entry ^span-act -1.0 ^word house)
(make lex-entry ^span-act -1.0 ^word train)
(make lex-entry ^span-act -1.0 ^word store)
(make lex-entry ^span-act -1.0 ^word hippie)


(make visible-word ^span-pool external ^is the ^index 1)
(make visible-word ^span-pool external ^is doctor ^index 2)
(make visible-word ^span-pool external ^is is ^index 3)
(make visible-word ^span-pool external ^is in ^index 4)
(make visible-word ^span-pool external ^is the ^index 5)
(make visible-word ^span-pool external ^is hood ^index 6)
```

APPENDIX B.

DIGIT SYMBOL MODEL SOURCE CODE

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : (c)1996 Georgia Institute of Technology, All Rights Reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : ds3.span
;;; Version     : 1.0
;;;
;;; Description : Productions to model the Digit Symbol task
;;;
;;;             : Pseudo-conservative version:
;;;             : * Looks at target
;;;             : * Initiates eye movement
;;;             : * Looks at key
;;;             : * If target is present, compares
;;;             : * If not, sets up rehearsal of key
;;;             : * Moves back to target
;;;             : * Then compares
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :span)
(clean-up)


;;; set up parameters
;;;
(setf *action-queue* nil)
(set-switch :cycle-time 50)
(set-switch :gamma-rate 1.0)

;;
;; DM types and default stuff
;;
(load "SPAN:default-DMtypes.span")
(defDMtype symbol is tag rehearse)
```

```
(defDMtype garbage tag)
(defDMtype rehearse item)

(p done-yes
   (command ^do press-yes)
-->
   (halt))

(p done-no
   (command ^do press-no)
-->
   (halt))

(load "SPAN:default-productions.span")


;;;
;;; in the first step the target is out in the world
;;;

(p create-symbol-from-buffer
   (goal ^is do-ds ^status active)
   (symbol ^is <sym> ^span-pool external ^tag <which>)
   - (symbol ^is <sym> ^span-pool internal ^tag <which>) > -1.0
-->
   (transmit-make 1.3 symbol ^span-pool internal ^is <sym> ^tag <which>
                  ^rehearse yes))


(p refresh-symbol-from-buffer
   (goal ^is do-ds ^status active)
   (symbol ^is <s> ^span-pool external ^tag <which>)
   {(symbol ^is <s> ^span-pool internal ^tag <which>) <t>} > -1.0
-->
   (transmit 0.5 <t>))

;;;
;;; acquire the key
;;;

(p initiate-move-to-key
   (goal ^is do-ds ^status active)
   (symbol ^span-pool internal ^tag target)
   (symbol ^span-pool external ^tag target)
   - (symbol ^tag key)
   - (command)
-->
   (make command ^span-act 1.0 ^do move-to-key ^tag move ^span-pool motor))


;;;
;;;  What if we get the key and we've lost the target?
;;;
```

```
(p initiate-rehearse-key
   (goal ^is do-ds ^status active)
   (symbol ^tag key)
   - (symbol ^tag target)
   - (goal ^is rehearse) > -1.0
-->
   (transmit-make 1.0 goal ^is rehearse ^parent do-ds ^status active))


(p move-back-to-target
   (goal ^is do-ds ^status active)
   (symbol ^tag key ^span-pool external)
   (rehearse)   ;; vs. (goal ^is rehearse)
   - (command)
-->
   (make command ^span-act 1.0 ^do move-to-target ^tag move ^span-pool motor))


;;;
;;; do the comparison
;;;

(p compare-yes
   (goal ^is do-ds ^status active)
   (symbol ^is <sym1> ^tag target)
   (symbol ^is <sym1> ^tag key)
   {(command ^do press-yes) <mc>} < 0.0
-->
   (transmit 1.0 <mc>))


(p compare-yes-buffer
   (goal ^is do-ds ^status active)
   (symbol ^is <sym1> ^span-pool external)
   (symbol ^is <sym1> ^tag rhsl)
   {(command ^do press-yes) <mc>} < 0.0
-->
   (transmit 1.0 <mc>))


(p compare-no
   (goal ^is do-ds ^status active)
   (symbol ^is <sym1> ^tag target)
   (symbol ^is <> <sym1> ^tag key)
   {(command ^do press-no) <mc>} < 0.0
-->
   (transmit 1.0 <mc>))
```

```
;; handling restarts

(p wait
   {(restart) <rstrt>}
 -->
   (remove <rstrt>))

;;;
;;; rehearsal
;;;


(p build-rehearse-subgoal
   (goal ^is do-ds ^status active)
   {(symbol ^span-pool internal ^is <s> ^rehearse yes ^tag key) <isym>}
   - (goal ^is rehearse) > -1.0
-->
   ; (modify <isym> ^rehearse yes)
   (transmit-make 1.0 goal ^is rehearse ^parent do-ds ^status active))


(p build-rehearse-motor
   (goal ^is rehearse ^status active)
   {(symbol ^is <s> ^span-pool internal ^rehearse yes) <isymb>}
   ; - (symbol ^is <s> ^span-pool external)
   - (rehearse ^span-pool motor ^item <s>) > -1.0
-->
   (modify <isymb> ^rehearse no)
   (transmit-make 1.0 rehearse ^span-pool motor ^item <s>))


(p refresh-rehearse-motor
   (goal ^is rehearse ^status active)
   {(rehearse) <rehrs>} < 0.0
-->
   (transmit 1.0 <rehrs>))


(p process-rehearse
   (goal ^is rehearse ^status active)
   {(rehearse ^span-act <actv> ^item <sym>) <rhsl>}
   - (rehearse ^span-act {> <actv>})
-->
   (remove <rhsl>)
   (eval-exp (run-rehearse (quote <sym>))))

;;;; initialize declarative memory

(make command ^span-pool motor ^span-act -1.0 ^do press-yes ^tag done)
(make command ^span-pool motor ^span-act -1.0 ^do press-no ^tag done)
(make symbol ^span-pool external ^is A ^tag target)
(make-goal)
(make-garbage-node)
;(make-garbage-node)
```

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : (c)1996 Georgia Institute of Technology, All Rights Reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : ds-ext.lisp
;;; Version     : 1.0
;;;
;;; Description : External functions for model of Digit Symbol task
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :span)

(defvar *action-queue* nil)
(defvar *se-time* 250)            ;; time to extract symbol
(defvar *move-time* 200)           ;; time to move the eye
(defvar *latency* 0)

(defmacro while (test &body body)
  `(do ()
       ((not ,test))
     ,@body))

(defun sort-action-queue ()
  (setf *action-queue* (sort *action-queue* #'< :key #'first)))


(setf *warn-if-redefine* nil)
(defun timehook (the-time)
  (while (and *action-queue*
              (>= the-time (first (first *action-queue*))))
    (let ((do-what (second (pop *action-queue*))))
      (when *ptrace* (format t "~% =*= Yanking from queue =*=~%"))
      (eval do-what))))
(setf *warn-if-redefine* t)


(defun move-to-key ()
  (progn
    (mapc #'remove-from-wm (consistent-elements '(symbol ^ span-pool external)))
    (let ((new-entry (list (+ *span-clock* *se-time* *move-time* -10)
                           '(make symbol ^span-pool external ^is A ^tag key))))
      (push new-entry *action-queue*)
      (sort-action-queue))))
```

```
(defun press-yes ()
  (let ((latency (+ *span-clock* *se-time* 170)))
    (format t "~%=*= YES pressed at ~S =*=~%" latency)
    (setf *latency* latency)))


(defun press-no ()
  (format t "~%=*= NO pressed at ~S =*=~%" (+ *span-clock* *se-time* 170)))


(defun run-rehearse (symbol)
  (let ((create-action (list (+ *span-clock* 170)
                             (append
                              '(make symbol ^span-pool external ^tag rhsl ^is)
                              (list symbol))))
        (del-action (list (+ *span-clock* 310)
                          (list 'mapc '#'remove-from-wm
                                '(consistent-elements
                                  '(symbol ^ span-pool external ^tag rhsl))))))
  (push create-action *action-queue*)
  (push del-action *action-queue*)
  (sort-action-queue)))


(defmacro make-remove-rhsl ()
  `(list 'mapc 'mapc '#'remove-from-wm
        '(consistent-elements '(symbol ^ span-pool external
                                ^tag rhsl))))


(defmacro make-garbage-node ()
  `(make garbage ^span-pool internal
         ^span-act ,(decay (random 1.0) *se-time*)
         ^tag ,(gensym)))


(defmacro make-goal ()
  `(make goal ^span-pool internal ^span-act ,(random 1.0) ^is do-ds
         ^status active))


(defmacro run-ds (cycles)
  `(progn
     (setf *latency* 0)
     (load "SPAN:digitsymbol;ds2.span")
     (run ,cycles)
     *latency*))
```

```
(defmacro run-ds-quiet ()
  `(progn
     (setf *latency* 0)
     (load "SPAN:digitsymbol;ds2.span")
     (set-switch :trace-productions nil :trace-capacity nil)
     (run)
     *latency*))

(defun avg-times (num-runs)
  (gc)
  (let ((timelist nil))
    (dotimes (i num-runs timelist)
      (push (run-ds-quiet) timelist))
    (float (/ (apply #'+ timelist) (length timelist)))))
```

APPENDIX C.

COMPUTATION SPAN MODEL SOURCE CODE

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : ©1996 Michael D. Byrne, All Rights Reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : cspan.span
;;; Version     : 1.0
;;;
;;; Description : Productions for the computation span task, assuming that
;;;             : eveything is simply available on the display (i.e. no
;;;             : eye movements are made).
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :span)
(clean-up)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; set switches
(setf *action-queue* nil)
(set-switch :cycle-time 50)
;(set-switch :gamma-rate 1.0)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; declarative memory types
;;;
(load "SPAN:default-dmtypes.span")

(defDMtype number is position buffer rehearse)
          ;;; values for "position": op1, op2, result, answer1, answer2, etc.
(defDMtype operator is buffer rehearse)
          ;;; like "times" "plus" "minus" "divide"
(defDMtype math-fact op1 op2 operator value)
(defDMtype arrow position)
```

```
(defDMtype recall-flag howmany)  ;;; screen says "recall"
(defDMtype stop)

;; rehearsal types
(defdmtype num-code is id buffer rehearse next)
(defdmtype rehearse item)
(defdmtype lockout tag count)
(defdmtype next-item is helped)
(defdmtype change)

(load "SPAN:default-productions.span")



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Small flow control productions
;;;

(p cease-and-desist
   (stop)
-->
   (halt))


(p wait
   {(restart) <rstrt>}
 -->
   (remove <rstrt>))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Deciding what kind of thing, equation or recall.
;;;

(p switch-to-wait-for-recall
   {(goal ^is do-quest ^status active) <g>}
   (recall-flag ^span-pool external)
-->
   (modify <g> ^is wait-for-recall))


(p start-question-make
   (goal ^is do-quest ^status active)
   {(change) <c>}
   (number ^span-pool external ^is <n> ^position op2)
   - (goal ^is get-op2) > -1.0
-->
   (transmit-make 1.0 goal ^is get-op2 ^parent do-quest ^status active)
   (remove <c>))
```

```
(p start-question-activate
   (goal ^is do-quest ^status active)
   {(change) <c>}
   (number ^span-pool external ^is <n> ^position op2)
   {(goal ^is get-op2 ^status satisfied) <g>} < 0.0
-->
   (transmit-modify 1.0 <g> ^status active)
   (remove <c>))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; First, get that second operand started
;;;

(p build-num-code
   (goal ^is get-op2 ^status active)
   (number ^is <n> ^position op2)
   - (num-code ^is <n> ^id none) > -1.0
-->
   (transmit-make 1.0
                  num-code ^span-bias 1.3 ^is <n> ^id none ^rehearse yes
                  ^next none))


(p refresh-rehearsal
   (goal ^is rehearse ^status active)
   {(next-item ^is <n> ^helped no) <next>}
   {(num-code ^id <n> ^rehearse yes) <num>} < 0.0
-->
   (transmit 1.0 <num>)
   (modify <next> ^helped yes)
   (make lockout ^span-pool support ^tag rehearse))


(p refresh-num-code
   (goal ^is get-op2 ^status active)
   (number ^is <n> ^position op2)
   {(num-code ^is <n> ^id none) <num>} < 0.0
-->
   (transmit 1.0 <num>))


(p bump-nums
   (goal ^is get-op2 ^status active)
   (next-item ^is start)
   (num-code ^id none)
   {(num-code ^span-pool internal ^id {<> none}) <nc>}
-->
   (transmit 1.0 <nc>))
```

```
(p add-to-rehearsal-make
   {(goal ^is get-op2 ^status active) <g>}
   {(next-item ^is start) <nxti>}
   {(num-code ^id none) <newnum>}
   {(num-code ^next start) <last>}
   - (goal ^is compute-value) > -1.0
-->
   (bind <newid>)
   (modify <nxti> ^is <newid> ^helped no)
   (modify <newnum> ^id <newid> ^next start)
   (modify <last> ^next <newid>)
   (modify <g> ^status satisfied)
   (transmit-make 1.0 goal ^is compute-value ^parent do-quest ^status active))


(p add-to-rehearsal-activate
   {(goal ^is get-op2 ^status active) <g>}
   {(next-item ^is start) <nxti>}
   {(num-code ^id none) <newnum>}
   {(num-code ^next start) <last>}
   {(goal ^is compute-value) <compval>} < 0.0
-->
   (bind <newid>)
   (modify <nxti> ^is <newid> ^helped no)
   (modify <newnum> ^id <newid> ^next start)
   (modify <last> ^next <newid>)
   (modify <g> ^status satisfied)
   (transmit-modify 1.0 <compval> ^status active))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; computing the value
;;;

(p retreive-value
   (goal ^is compute-value ^status active)
   (number ^is <op1> ^position op1)
   (number ^is <op2> ^position op2)
   (operator ^is <oprtr>)
   {(math-fact ^op1 <op1> ^op2 <op2> ^operator <oprtr>) <mf>} < 0.0
-->
   (transmit 1.0 <mf>))


(p build-result
   (goal ^is compute-value ^status active)
   (math-fact ^value <rslt>)
   - (number ^position result ^is <rslt>) > -1.0
-->
   (transmit-make 1.3 number ^span-bias 1.3 ^position result ^is <rslt>
                  ^rehearse no))
```

```
(p refresh-result-1
   (goal ^is compute-value ^status active)
   (math-fact)
   {(number ^position result) <number>} < 0.0
-->
   (transmit 1.0 <number>))


(p got-result-new
   {(goal ^is compute-value ^status active) <g>}
   (number ^position result)
   - (goal ^is do-respond) > -1.0
-->
   (modify <g> ^status satisfied)
   (transmit-make 1.0 goal ^is do-respond ^parent do-quest ^status active))


(p got-result-activate
   {(goal ^is compute-value ^status active) <g>}
   (number ^position result)
   {(goal ^is do-respond) <resp>} < 0.0
-->
   (modify <g> ^status satisfied)
   (transmit-modify 1.0 <resp> ^status active))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; responding
;;;

;; The arrrow starts _above_ the answers, so move that first.
(p start-answer
   (goal ^is do-respond ^status active)
   (arrow ^position 0)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^span-pool motor ^do down-arrow
                  ^proc manual ^tag move))


(p refresh-result-2
   (goal ^is do-respond ^status active)
   {(number ^position result) <number>} < 0.0
-->
   (transmit 1.0 <number>))
```

```
(p match-answer-1
   {(goal ^is do-respond ^status active) <g>}
   {(number ^position result ^is <ans>) <num>}
   (arrow ^position 1)
   (number ^position answer1 ^is <ans>)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^do return-key
                  ^span-pool motor ^proc manual ^tag ans)
   (modify <g> ^status satisfied)
   (modify <num> ^position none))


(p not-match-1
   (goal ^is do-respond ^status active)
   (number ^position result ^is <ans>)
   (arrow ^position 1)
   (number ^position answer1 ^is <> <ans>)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^do down-arrow
                  ^span-pool motor ^proc manual ^tag ans))


(p match-answer-2
   {(goal ^is do-respond ^status active) <g>}
   {(number ^position result ^is <ans>) <num>}
   (arrow ^position 2)
   (number ^position answer2 ^is <ans>)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^do return-key ^span-pool motor
                  ^proc manual ^tag ans)
   (modify <g> ^status satisfied)
   (modify <num> ^position none))


(p not-match-2
   (goal ^is do-respond ^status active)
   (number ^position result ^is <ans>)
   (arrow ^position 2)
   (number ^position answer2 ^is <> <ans>)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^do down-arrow ^span-pool motor
                  ^proc manual ^tag ans))
```

```
(p match-answer-3
   {(goal ^is do-respond ^status active) <g>}
   {(number ^position result ^is <ans>) <num>}
   (arrow ^position 3)
   (number ^position answer3 ^is <ans>)
   - (command ^proc manual)
-->
   (transmit-make 1.3 command ^do return-key ^span-pool motor
                  ^proc manual ^tag ans)
   (modify <g> ^status satisfied)
   (modify <num> ^position none))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Recalls
;;;

(p switch-to-do-recall
   {(goal ^is wait-for-recall ^status active) <g>}
   (num-code ^is start ^buffer audio)
-->
   (modify <g> ^is do-recall)
   (make lockout ^span-pool support ^tag rehearse ^count 0))


(p kill-rehearse-start
   (goal ^is do-recall ^status active)
   (num-code ^is start ^buffer audio)
   {(num-code ^span-pool internal ^is start) <istart>}
-->
   (modify <istart> ^rehearse no))


(p recall-item-from-buffer
   (goal ^is do-recall ^status active)
   (num-code ^is {<num> <> start} ^buffer audio)
   - (command ^proc manual)
   - (number-key)
   - (lockout ^tag recall ^count <num>)
-->
   (transmit-make 1.3 number-key ^span-pool motor ^is <num>))


(p process-number-key
   {(number-key ^span-pool motor ^is <num>) <nk>}
-->
   (remove <nk>)
   (make lockout ^span-pool support ^tag recall ^count <num>)
   (eval-exp (press-num (quote <num>))))
```

```
(p release-recall-lock
   {(lockout ^tag recall) <lock>}
-->
   (remove <lock>))


(p advance-number
   (goal ^is do-recall ^status active)
   (number-key)
   - (command ^proc manual)
-->
   (transmit-make 1.1 command ^span-pool motor ^do return-key
                  ^proc manual ^tag pr))


(p refresh-advance
   (goal ^is do-recall ^status active)
   {(command ^do return-key) <key>} < 0.0
-->
   (transmit 1.0 <key>))


(p get-out
   (goal ^is do-recall ^status active)
   (next-item ^is start)
   (num-code ^is start ^rehearse no) < 0.0
   - (command ^proc manual)
-->
   (transmit-make 1.1 command ^span-pool motor ^do return-key
                  ^proc manual ^tag pr))


(p failed!
   (goal ^is << do-recall wait-for-recall >> ^status active)
   (next-item ^helped yes)
   - (num-code ^span-pool internal)
-->
   (write *FAIL)
   (eval-exp (recall-stop))
   (halt))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Rehearsal
;;;
```

```
(p build-rehearse-motor
   (goal ^is rehearse ^status active)
   {(next-item ^is <item>) <ni>}
   {(num-code ^id <item> ^is <n> ^rehearse yes ^next <next>) <num>}
   - (num-code ^is <n> ^buffer audio)
   - (rehearse)
   - (command ^proc vocal)
   - (lockout)
-->
   (modify <num> ^rehearse no)
   (modify <ni> ^is <next> ^helped no)
   (transmit-make 1.3 rehearse ^span-pool motor ^item <n> ^proc vocal))


(p refresh-rehearse-motor
   (goal ^is rehearse ^status active)
   {(rehearse) <rehrs>} < 0.0
-->
   (transmit 1.0 <rehrs>))


(p process-rehearse
   {(rehearse ^span-act <actv> ^item <num>) <rhsl>}
   - (rehearse ^span-act {> <actv>})
   - (lockout)
-->
   (make lockout ^span-pool support ^tag rehearse ^count 0)
   (remove <rhsl>)
   (eval-exp (run-rehearse (quote <num>))))


(p release-lock
   {(lockout ^tag rehearse) <lock>}
-->
   (remove <lock>))


(p transmit-from-rehearsal-buffer
   {(num-code ^is <n> ^span-pool internal) <num>} > -1.0
   (num-code ^is <n> ^span-pool external ^buffer audio)
-->
   (transmit 1.0 <num>))


(p clear-from-rehearsal-buffer
   (goal ^is rehearse ^status active)
   {(num-code ^is <n> ^span-pool internal ^rehearse no) <num>} > -1.0
   (num-code ^is <n> ^span-pool external ^buffer audio)
-->
   (modify <num> ^rehearse yes))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; end of productions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Will need a bevy of integer arithmetic facts
(build-math-facts)

;;; Reset the display
(reset *the-display*)

;;; make the goal
(make-goal)

;;; make the rehearse subgoal
(make-rehearse-goal)
(make-start-num)
(make next-item ^span-pool support ^is START ^helped no)

;;; make some garbage
(make-garbage-node)
(make-garbage-node)
(make-garbage-node)
(make-garbage-node)
```

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : ©1996 Michael D. Byrne, All Rights Reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : cspan-ext.lisp
;;; Version     : 1.0
;;;
;;; Description : External functions for the computation span task.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :span)

;;; globals
(defvar *action-queue* nil)
(defvar *question-times* nil)
(defvar *recall-times* nil)
(defvar *recall-acc* nil)


;;; managing the action queue
(defmacro while (test &body body)
  `(do ()
       ((not ,test))
     ,@body))


(defun sort-action-queue ()
  (setf *action-queue* (sort *action-queue* #'< :key #'first)))


(setf *warn-if-redefine* nil)
(defun timehook (the-time)
  (while (and *action-queue*
              (>= the-time (first (first *action-queue*))))
    (let ((do-what (second (pop *action-queue*))))
      (when *ptrace*
        (setf *print-pretty* nil)
        (format t "~% =*= Yanking from queue: ~S~%" do-what)
        (setf *print-pretty* t))
      (eval do-what))))
(setf *warn-if-redefine* t)


;;; external functions (link production RHS's to display)
```

```
(defun return-key ()
  (enter-key *the-display*))

;;; down-arrow
(defun down-arrow ()
  (down-arrow-key *the-display*))

;;; press-num
(defun press-num (the-num)
  (number-key *the-display* the-num))



;;; MAKE-MATH-FACT      [Function]
;;; Description : Makes an individual math fact.

(defun make-math-fact (op1 op2 oprtr val)
  (eval `(make math-fact ^span-act -1.0 ^span-bias 1.3
               ^op1 ,op1 ^op2 ,op2 ^operator ,oprtr ^value ,val)))


;;; BUILD-MATH-FACTS      [Function]
;;; Description : Builds the entire database of math facts that we'll need.
;;;
(defun build-math-facts ()
  ; addition
  (dotimes (i 9 nil)
    (dotimes (j 9 nil)
      (let ((the-val (+ i j 2)))
        (make-math-fact (1+ i) (1+ j) '+ the-val))))
  ; subtraction
  (dotimes (i 9 nil)
    (dotimes (j 9 nil)
      (let ((the-val (- i j)))
        (when (> the-val 0)
          (make-math-fact (1+ i) (1+ j) '- the-val))))))


;;; RUN-REHEARSE      [Function]
;;; Description : Adds the appropriate items to the action queue for the
;;;             : rehearsal of one element.  First, a clear of all elements
;;;             : in the audio buffer, then a MAKE of the verbal code for
;;;             : the number, and finally a clear of that verbal code after
;;;             : some time has elapsed.
```

```
(defun run-rehearse (number)
  (let ((post-del-action (list (+ *span-clock* 310)
                               `(remove-consis-elems
                                 '(num-code ^ buffer audio ^is ,number))))
        (create-action (list (+ *span-clock* 170)
                             (append
                              '(make num-code ^span-pool external
                                ^buffer audio ^is)
                              (list number))))
        (prep-del-action (list (+ *span-clock* 120)
                               '(remove-consis-elems
                                 '(num-code ^ buffer audio)))))
    (push create-action *action-queue*)
    (push prep-del-action *action-queue*)
    (push post-del-action *action-queue*)
    (sort-action-queue)))


;;;;;;;;;;
;;;
;;; MAKE macros for initialization of the model

(defmacro make-garbage-node ()
  `(make garbage ^span-pool internal
         ^span-act ,(random 1.0) ^tag ,(gensym)))

(defmacro make-goal ()
  `(make goal ^span-pool internal ^span-act ,(random 1.0) ^is do-quest
         ^status active ^parent nil))


(defmacro make-rehearse-goal ()
  `(make goal ^span-pool internal ^span-act ,(random 1.0) ^is rehearse
         ^status active ^parent nil))

(defmacro make-start-num ()
  `(make num-code ^span-pool internal ^span-act ,(random 1.0) ^is start
         ^id START ^rehearse yes ^next START))

;;;;;;;;;;
;;;
;;; Runtime management


(defun question-stop ()
  (push (- *span-clock* (first *question-times*))
        *question-times*))


(defun recall-stop ()
  (if (equal (targets *the-display*)
             (recalls *the-display*))
    (progn
      (push 1 *recall-acc*)
```

```
      (push (- *span-clock* (first *question-times*))
            *recall-times*))
    (push 0 *recall-acc*))
  (push 0 *question-times*))


(defun list-avg (the-list)
  (float (/ (apply #'+ the-list) (length the-list))))


(defun finish-run ()
  (format t "~% Proportion correct recalls: ~S" (list-avg *recall-acc*))
  (format t "~% Average question time: ~S ms"
          (list-avg (ccl::remove 0 *question-times*)))
  (format t "~% Average recall time (~S): ~S ms~%"
          (length *recall-times*) (list-avg *recall-times*)))


(defmacro avg-runs (num-runs gamma span)
  `(progn
     (setf *recall-acc* nil)
     (setf *question-times* '(0))
     (setf *recall-times* nil)
     (initialize *the-display*)
     (while (< (length *recall-acc*) ,num-runs)
       (format t "~%~% =-=-=-= run ~S =-=-=-=~%~%" (length *recall-acc*))
       (run-cspan-quiet ,gamma ,span))
     (format t "~%~% =-=-=-=-=-=-=-=-=-=-=-=-=-=-=")
     (format t "~% Data for ~S runs of span ~S, gamma ~S"
             (length *recall-acc*) ,span ,gamma)
     (finish-run)))


(defmacro run-cspan (span cycles)
  `(progn
     (load "SPAN:cspan;cspan.span")
     (set-switch :trace-productions nameonly)
     (setf (curr-len *the-display*) ,span)
     (run ,cycles)))


(defmacro run-cspan-quiet (gamma span)
  `(progn
     (load "SPAN:cspan;cspan.span")
     (set-switch :trace-productions nil :trace-capacity nil :gamma-rate ,gamma)
     (setf (curr-len *the-display*) ,span)
     (run 350)))


(defmacro pt (what)
  `(set-switch :trace-productions ,what))
```

```
;;;  -*- mode: LISP; Package: SPAN; Syntax: COMMON-LISP;  Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Author      : Mike Byrne
;;; Copyright   : ©1996 Michael D. Byrne, all rights reserved
;;; Address     : Georgia Institute of Technology
;;;             : School of Psychology
;;;             : Atlanta,Georgia 30332-0170
;;;             : byrne@cc.gatech.edu
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Filename    : new-cspan-display.lisp
;;; Version     : 1.0
;;;
;;; Description : CLOS code to handle the functions of the device (or display)
;;;             : for the Computation Span task.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :span)

(defvar *the-display* nil)


(defclass cspan-display ()
  ((trial-list :accessor trial-list)
   (arrow :accessor arrow)
   (curr-len :accessor curr-len)
   (targets :accessor targets)
   (recalls :accessor recalls)
   (stim-list :accessor stim-list)
   (outcome-list :accessor outcomes)))


(defmethod initialize ((self cspan-display))
  "Initialize the thing"
  (setf (trial-list self) nil)
  (setf (curr-len self) 1)
  (setf (targets self) nil)
  (setf (recalls self) nil)
  (setf (outcomes self) nil)
  (load "SPAN:cspan;stimuli.pfsl"))


(defmethod reset ((self cspan-display))
  (setf (targets self) nil)
  (setf (recalls self) nil)
  (setup-question self))
```

```
(defmethod down-arrow-key ((self cspan-display))
  "Moves the arrow down in the display"
  (incf (arrow self))
  (remove-consis-elems '(arrow ^ span-pool external))
  (eval `(make arrow ^span-pool external ^position ,(arrow self))))


(defmethod setup-question ((self cspan-display))
  "Sets up the state and WM for a question display"
  (setf (arrow self) 0)
  (make arrow ^span-pool external ^position 0)
  (let ((trial-list (pop (stim-list self))))
    (make-screen-num (first trial-list) 'op1)
    (make-screen-num (third trial-list) 'op2)
    (make-screen-num (fourth trial-list) 'answer1)
    (make-screen-num (fifth trial-list) 'answer2)
    (make-screen-num (sixth trial-list) 'answer3)
    (eval `(make operator ^span-pool external ^buffer visual
                 ^is ,(second trial-list)))
    (make change ^span-pool external)
    (push (first (last trial-list)) (targets self))
    (print-q-display trial-list)))


(defun print-q-display (scrn-list)
  (format t "~%-----------------------------------")
  (format t "~%|    ~S ~S ~S   =  ?   [~S ~S ~S]      |"
          (first scrn-list) (second scrn-list) (third scrn-list)
          (fourth scrn-list) (fifth scrn-list) (sixth scrn-list))
  (format t "~%-----------------------------------~%"))


(defmethod finish-question ((self cspan-display))
  "Remove WM items, reset the arrow, and check if we're done."
  (remove-consis-elems '(number ^ span-pool external ^buffer visual))
  (remove-consis-elems '(arrow ^ span-pool external))
  (remove-consis-elems '(operator ^ span-pool external))
  (setf (arrow self) nil)
  (question-stop)
  (if (= (length (targets self)) (curr-len self))
    (setup-recall self)
    (setup-question self)))


(defmethod setup-recall ((self cspan-display))
  "All we need to do is show the recall flag, right?"
  (make recall-flag ^span-pool external)
  (print-r-display (curr-len self)))


(defun print-r-display (len)
  (format t "~%-----------------------------------")
  (format t "~%|   Recall:  (~S)                 |" len)
  (format t "~%-----------------------------------~%~%"))
```

```
(defmethod number-key ((self cspan-display) the-num)
  "Number-key is a response, so push it onto the recalls list."
  (push the-num (recalls self)))


(defmethod enter-key ((self cspan-display))
  "If we're in a question trial, end it.  Else advance the cursor."
  (if (null (arrow self))
    (advance-return self)
    (finish-question self)))


(defmethod advance-return ((self cspan-display))
  "Did we make it?"
  (when (equal (length (recalls self)) (curr-len self))
    (make stop ^span-pool support)
    (recall-stop)
    (format t "~% ###> Recalled these items: ~S~%" (recalls self))
    (format t "       Wanted these items: ~S~%~%" (targets self))))


(defmethod setup-new-trial ((self cspan-display))
  "Increment the trial length and clean up other variables."
  (incf (curr-len self))
  (setf (targets self) nil)
  (setf (recalls self) nil)
  (setf (outcomes self) nil)
  (setup-question self))


(defun make-screen-num (num pos)
  "Makes a DME for numbers on the screen"
  (eval `(make number ^span-pool external ^is ,num ^buffer visual
               ^position ,pos)))

(setf *the-display* (make-instance 'cspan-display))
(initialize *the-display*)
```

REFERENCES

Aitkenhead, A. M., & Slack, J. M. (1985). *Issues in cognitive modeling.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (1974). Retrieval of propositional information from long-term memory. *Cognitive Psychology, 6,* 451–474.

Anderson, J. R. (1976). *Language, memory, and thought*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., & Bower, G. H. (1973). *Human associative memory.* Washington, DC: Winston and Sons.

Anderson, J. R., Matessa, M., & Douglass, S. (1995). The ACT-R theory and visual attention. In J. D. Moore & J. F. Lehman (Eds.) *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pp. 61–65. Mahwah, NJ: Erlbaum.

Anderson, J. R., Reder, L. M., & Lebiére, C. (in press). Working memory: Activation limitations on retrieval. To appear in *Cognitive Psychology*.

Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. In K. W. Spence & J. T. Spence (Eds.), *The psychology of learning and motivation* (Vol. 2). New York: Academic Press.

Baars, B. J. (1992). *Experimental slips and human error: Exploring the architecture of volition*. New York: Plenum Press.

Babcock, R. L. (1994). Analysis of adult age differences on the Raven's Advanced Progressive Matrices test. *Psychology and Aging, 9,* 303-314.

Baddeley, A. D. (1986). *Working memory.* Oxford: Oxford University Press.

Baddeley, A. D., Eldridge, M, Lewis, V., & Thompson, N. (1984). Attention and retrieval from long-term memory. *Journal of Experimental Psychology: General, 113,* 518–540.

Baddeley, A. D., & Hitch, G. (1974). Working memory. In G. H. Bower (Ed.), *The psychology of learning and motivation* (Vol. 8). New York: Academic Press.

Bartlett, J. C. (1993). Limits on losses in face recognition. In J. Cerella, J. Rybash, W. Hoyer, & M. L. Commons (Eds.), *Adult information processing: Limits on loss.* San Diego: Academic Press.

Beach, K. D. (1978) The role of external mnemonic symbols in acquiring an occupation. In M.M. Gruneberg, P.E. Morris, and R.N. Sykes (Eds.) *Practical aspects of memory: Current research and issues* (vol. 1., pp 342–346). New York: Wiley.

Birren, J. E., & Fisher, L. M. (1995). Aging and speed of behavior: Possible consequences for psychological functioning. *Annual Review of Psychology, 46,* 329-353.

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text editing skill: A cognitive complexity analysis. *Human Computer Interaction, 5,* 1-48.

Byrne, M. D. (1994). Integrating, not debating, situated action and computational models: Taking the environment seriously. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society,* 118–123. Hillsdale, NJ: Lawrence Erlbaum.

Byrne, M. D., & Bovair, S. (in press). A working memory model of a common procedural error. Forthcoming in *Cognitive Science*.

Campbell, J. I. D., & Charness, N. (1990). Age-related declines in working memory skills: Evidence from a complex calculation task. *Developmental Psychology, 26,* 879-888.

Cantor, J., & Engle, R. W. (1993). Working memory capacity as long-term memory activation: An individual differences approach. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 19,* 1101-1114.

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

Carlson, R. A., Sullivan, M. A., & Schneider, W. (1989). Practice and working memory effects in building a procedural skill. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 15,* 517–526.

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,* 404-431.

Casner, S. M. (1994). Understanding the determinants of problem-solving behavior in a complex environment. *Human Factors, 36,* 580–596.

Cerella, J. (1985). Information processing rates in the elderly. *Psychological Bulletin, 98,* 67-83.

Cerella, J. (1990). Aging and information-processing rate. In J. E. Birren, & K. W. Schaie (Eds.) *Handbook of the psychology of aging* (3rd. Ed.). San Diego: Academic Press.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4,* 55-81.

Cherry, K. E., & Park, D. C. (1993). Individual difference and contextual variables influence spatial memory in younger and older adults. *Psychology and Aging, 8,* 517-526.

Chi, M. (1978). Knowledge structures and memory development. In R.S. Siegler (Ed.), *Children's thinking: What develops?* Hillsdale, NJ: Erlbaum.

Cohen, N., & Squire, L. R. (1980). Preserved learning and retention of pattern analyzing skills skills in amnesia: Dissociation of knowing how and knowing that. *Science, 210,* 207–210.

Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological Bulletin, 104,* 163–191.

Craik, F. I. M. (1977). Age differences in human memory. In J. E. Birren, & K. W. Schaie (Eds.) *Handbook of the psychology of aging*. New York: Van Nostrand Reinhold.

Daneman, M. (1991). Working memory as a predictor of verbal fluency. *Journal of Psycholinguistic Research, 20,* 445–464.

Daneman, M., & Carpenter, P. A. (1980). Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior, 19,* 450–466.

Daneman, M., & Green, I. (1986). Individual differences in comprehending and producing words in context. *Journal of Memory and Language, 25,* 1–18.

Davis, G. A., & Ball, H. E. (1989). Effects of age on comprehension of complex sentences in adulthood. *Journal of Speech and Hearing Research, 32,* 143-150.

Dobbs, A. R., & Rule, B. G. (1989). Adult age differences in working memory. *Psychology and Aging, 4,* 500-503.

Dolan, C. P., & Smolensky, P. (1989). Tensor production system: A modular architecture and representation. *Connection Science, 1,* 53–68.

Doorenbos, R. B. (1994). Production matching for large learning systems. Technical report CMU-CS-95-113, Carnegie-Mellon University, Pittsburgh, PA.

Duchek, J. M., & Balota, D. A. (1993). Sparing activation processes in older adults. In J. Cerella, J. Rybash, W. Hoyer, & M. L. Commons (Eds.), *Adult information processing: Limits on loss*. San Diego: Academic Press.

Ekstrom, R. B., French, J.W., Harman, H. H., & Derman, D. (1976). *Manual for kit of factor-referenced cognitive tests.* Princeton, NJ: Educational Testing Service.

Engle, R. W., Cantor, J., & Carullo, J. J. (1992). Individual differences in working memory and comprehension: A test of four hypotheses. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 18,* 972–992.

Engle, R. W., Conway, A. R. A., Tuholski, S. W., & Shisler, R. J. (1995). A resource account of inhibition. *Psychological Science, 6,* 122-125.

Ericsson, K. A., & Kintsch, W. (1995). Long-term working memory. *Psychological Review, 102,* 211–245.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Rogers, W. A., Fisk, A. D., & Hertzog, C. H. (1994). Do ability-performance relationships differentiate age and practice effects in visual search? *Journal of Experimental Psychology: Learning, Memory, and Cognition, 20,* 710–738.

Foos, P. W., & Wright, L. (1992). Adult age differences in the storage of information in working memory. *Experimental Aging Research, 18,* 51-57.

Fowler, B. (1994). P300 as a measure of workload during a simulated aircraft landing task. *Human Factors, 36,* 670–683.

Gardner, H. (1985). *The mind's new science: A history of the cognitive revolution.* New York: Basic Books.

Giambra, L. M. (1989). Task-unrelated-thought frequency as a function of age: A laboratory study. *Psychology and Aging, 4,* 136-143.

Gick, M. L., Craik, F. I. M., & Morris, R. G. (1988). Task complexity and age differences in working memory. *Memory & Cognition, 16,* 353-361.

Gerard, L., Zacks, R. T., Hasher, L., & Radvansky, G. A. (1991). Age deficits in retrieval: The fan effect. *Journal of Gerontology: Psychological Sciences, 46,* 131-136.

Goldman, S. R., & Varma, S. (1996). CAPing the construction-integration model of discourse comprehension. To appear in C. Weaver, S. Mannes, & C. Fletcher (Eds.), *Discourse comprehension: Models of processing revisited* (pp. 337–358). Hillsdale, NJ: Erlbaum.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Human-Computer Interaction, 8,* 237–309.

Greeno, J. G., & Moore, J. L. (1993). Situativity and symbols: Response to Vera and Simon. *Cognitive Science, 17,* 49–60.

Hamm, V. P., & Hasher, L. (1992). Age and the availability of inferences. *Psychology and Aging, 7,* 56-44.

Hasher, L., Stoltzfus, E. R., Zacks, R. T., & Rypma, B. (1991). Age and inhibition. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17,* 163-169.

Hasher, L., & Zacks, R. T. (1988). Working memory, comprehension, and aging: A review and a new view. In G. H. Bower (Ed.), *The psychology of learning and motivation* (Vol. 22, pp. 193-225). San Diego: Academic Press.

Hitch, G. J. (1978). The role of short-term working memory in mental arithmetic. *Cognitive Psychology, 10,* 302-323.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery*. Cambridge, MA: MIT press.

Hutchins, E. (1990). The technology of team navigation. In J. Galegher, R. E. Kraut, & C. Egido (Eds.). *Intellectual teamwork: Social and technological foundations of cooperative work* (pp. 191–220). Hillsdale, NJ: Lawrence Erlbaum.

Jensen, A. R. (1982). The chronometry of intelligence. In R. J. Sternberg (Ed.), *Advances in the psychology of of human intelligence,* vol. 1, pp. 255–310. Hillsdale, NJ: Lawrence Erlbaum.

Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension: An individual differences approach. *Psychological Review, 99,* 123-148.

Just, M. A., Carpenter, P. A., & Hemphill, D. D. (1996). Constraints on processing capacity: Architectural or implementational? In D. Steier & T. Mitchell (Eds.) *Mind matters: A tribute to Allen Newell,* 141–178. Mahwah, NJ: Lawrence Erlbaum.

Keppel, G., & Underwood, B. J. (1962). Proactive inhibition in short-term retention of single items. *Journal of Verbal Learning and Verbal Behavior, 1,* 153–161.

Kieras, D. E., & Meyer, D. E. (1994). The EPIC architecture for modeling human information-processing: A brief introduction. (EPIC Tech. Report No. 1, TR-94/ONR-EPIC-1). Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1995). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. (EPIC Tech. Report No. 4, TR-95/ONR-EPIC-1). Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365-394.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1995). Predictive engineering models using the EPIC architecture for a high-performance task. In *Human Factors in Computer Systems: Proceedings of CHI'95,* pp. 11–18. New York: Addison Wesley.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review, 95,* 163-182.

Kirlik, A., Miller, R. A., & Jagacinski, R. J. (1993). Supervisory control in a dynamic and uncertain environment: A process model of skilled human-environment interaction. *IEEE Transactions on Systems, Man, and Cybernetics, 23,* 929–952.

Kitajima,M., & Polson, P. G. (1992). A computational model of skilled use of a graphical user interface. In *Proceedings of CHI'92* (pp. 241–249).

Kliegl, R., & Lindenberger, U. (1993). Modeling intrusions and correct recall in episodic memory: Adult age differences in encoding list context. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 19,* 617–637.

Kotovsky, K., & Kushmerick, N. (1991). Processing constraints and problem difficulty: A model. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive Psychology, 17,* 248-294.

Kyllonen, P. C. & Christal, R. E. (1990). Reasoning is (little more than) working-memory capacity. *Intelligence, 14,* 389–433.

Kyllonen, P. C. (1995). Aptitude testing inspired by information processing: A test of the four-sources model. *Journal of General Psychology, 120,* 375–405.

Lebiére, C., & Anderson, J. R. (1993). A connectionist implementation of the ACT-R production system. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society,* p. 635–640.

Lebiére, C., Anderson, J. R., & Reder, L. M. (1994). Error modeling in the ACT-R production system. In A. Ram & K. Eiselt (Eds.) *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 555–559. Hillsdale, NJ: Erlbaum.

Light, L. L. (1991). Memory and aging: Four hypotheses in search of data. *Annual Review of Psychology, 42,* 333–376.

Light, L. L., & Capps, J. L. (1986). Comprehension of pronouns in younger and older adults. *Developmental psychology, 22,* 580–585.

MacDonald, M. C., Just, M. A., & Carpenter, P. A. (1992). Working memory constraints on the processing of syntactic ambiguity. *Cognitive Psychology, 24,* 56–98.

Mannes, S. M., & Kintsch, W. (1991). Routine computing tasks: Planning as understanding. *Cognitive Science, 15,* 305-342.

McClelland, J. L., Rumelhart, D. E., & the PDP Research Group. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition,* vol 2. Cambridge, MA: MIT Press.

Melton, A. (1963). Implications of short-term memory for a general theory of memory. *Journal of Verbal Learning and Verbal Behavior, 2,* 1–21.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63,* 81-97.

Morrell, R. W., & Park, D. C. (1993). The effects of age, illustrations, and task variables on the performance of procedural assembly tasks. *Psychology and Aging, 8,* 389 399.

Morris, R. G., Gick, M. L., Craik, F. I. M. (1988). Processing resources and age differences in working memory. *Memory & Cognition, 16,* 362-366.

Morrow, D., Altieri, P., & Lierer, V. (1992). Aging, narrative organization, presentation mode, and referent choice strategies. *Experimental Aging Research, 18,* 75-84.

Morrow, D., Lierer, V., Altieri, P. A. (1992). Aging, expertise, and narrative processing. *Psychology and Aging, 7,* 376–388.

Newell, A. (1972). A theoretical exploration of mechanisms for coding the stimulus. In A. W. Melton & E. Martin (Eds.), *Coding processes in human memory* (pp. 373–434). Washinton, DC: Winston and sons.

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

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nissen, M. J., Knopman, D. S., & Schacter, D. L. (1987). Neurochemical dissociation of memory systems. *Neurology, 37,* 789-794.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review, 88,* 1-15.

Norman, S., Kemper, S., & Kynette, D. (1992). Adults' reading comprehension: Effects of syntactic complexity and working memory. *Journal of Gerontology, 47,* B258–265.

Park, J., & Kanwisher, N. (1994). Negative priming for spatial locations: Identity mismatching, not distractor inhibition. *Journal of Experimental Psychology: Human Perception and Performance, 20,* 613-623.

Peterson, L. R., & Peterson, M. J. (1959). Short-term retention of individual verbal items. *Journal of Experimental Psychology, 58,* 193–198.

Puckett, J. M., & Stockburger, D. W. (1988). Absence of age-related proneness to short-term retroactive interference in the absence of rehearsal. *Psychology and Aging, 3,* 342-347.

Raven, J. C., Court, J. H., & Raven, J. (1983). Manual for Raven's progressive matrices and vocabulary scales: Advanced Progressive Matrices Sets I and II. London: H. K. Lewis.

Reason, J. T. (1990). *Human error.* New York: Cambridge University Press.

Reber, P. J., & Kotovsky, K. (1992). Learning and problem solving under a memory load. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*.

Reder, L. M., & Anderson, J. R. (1980). A partial resolution of the paradox of interference: The role of integrating knowledge. *Cognitive Psychology, 12,* 447–472.

Reitman, J. S. (1974). Without surreptitious rehearsal, information in short-term memory decays. *Journal of Verbal Learning and Verbal Behavior, 13,* 365–377.

Rosenbloom, P. S., Newell, A., & Laird, J. E. (1991). Toward the knowledge level in Soar: The role of the architecture in the use of knowledge. In K. VanLehn (Ed.) *Architectures for intelligence* (pp. 75–114). Hillsdale, NJ: Lawrence Erlbaum.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition* (vol. 1). Cambridge, MA: MIT Press.

Salthouse, T. A. (1985). *A theory of cognitive aging*. Amsterdam: North-Holland.

Salthouse, T. A. (1988). Initiating the formalization of theories of cognitive aging. *Psychology and Aging, 3,* 3–16.

Salthouse, T. A. (1991a). *Theoretical perspectives on cognitive aging*. Hillsdale, NJ: Erlbaum.

Salthouse, T. A. (1991b). Mediation of adult age differences in cognition by reductions in working memory and speed of processing. *Psychological Science, 2,* 179–183.

Salthouse, T. A. (1992a). Influence of processing speed on adult age differences in working memory. *Acta Psychologica, 79,* 155–170.

Salthouse, T. A. (1992b). Why do adult age differences increase with task complexity? *Developmental Psychology, 28,* 905-918.

Salthouse, T. A. (1992c). Working-memory mediation of adults age differences in integrative reasoning. *Memory & Cognition, 20,* 413-423.

Salthouse, T. A. (1993). Influence of working memory on adult age differences in matrix reasoning. *British Journal of Psychology, 84,* 171-199.

Salthouse, T. A. (1994a). The aging of working memory. *Neuropsychology, 8,* 535 543.

Salthouse, T. A. (1994b). The nature of the influences of speed on adult age differences in cognition. *Developmental Psychology, 30,* 240–259.

Salthouse, T. A. (1996). The processing-speed theory of adult age differences in cognition. *Psychological Review, 103,* 403–428.

Salthouse, T. A., & Babcock, R. (1991). Decomposing adult age differences in working memory. *Developmental Psychology, 27,* 763-776.

Salthouse, T. A., & Coon, V. E. (1994). Interpretation of differential deficits: The case of aging and mental arithmetic. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 20,* 1172–1182.

Salthouse, T. A., & Kersten, A. W. (1993). Decomposing adult age differences in symbol arithmetic. *Memory & Cognition, 21,* 699–710.

Salthouse, T. A. & Meinz, E. J. (1995). Aging, inhibition, working memory, and speed. *Journals of Gerontology, Series B: Psychological Sciences and Social Sciences, 50B,* P297–307.

Salthouse, T. A., Mitchell, D. R. D., Skovronek, E., & Babcock, R. L. (1989). Effects of adult age and working memory on reasoning and spatial abilities. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 15,* 507-516.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures.* Hillsdale, NJ: Lawrence Erlbaum.

Schneider, W. & Detwiler, M. (1987) A connectionist/control architecture for working memory. In G.H Bower (Ed.), *The psychology of learning and motivation* (Vol. 21, pp. 54–119). New York: Academic Press.

Schneider, W. (1993). Varieties of working memory as seen in biology and in connectionist/control architectures. *Memory and Cognition, 21,* 184–192.

Schneider, W., & Oliver, W. L. (1991). An instructible connectionist/control architecture: Using rule-based instructions to accomplish connectionist learning in human time scale. In VanLehn, K. (Ed.), *Architectures for intelligence* (pp. 113–145). Hillsdale, NJ: Lawrence Erlbaum.

Searle, J. R. (1992). *The rediscovery of the mind.* Cambridge, MA: The MIT Press.

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

Skinner, B. F. (1966). An operant analysis of problem solving. In B. Klienmuntz (Ed.), *Problem solving: Research, method, and theory*, pp. 225-57. New York: John Wiley.

Squire, L. R. (1987). *Memory and brain.* New York: Oxford University Press.

Steele, G. L. (1990). *Common Lisp: The language* (2nd ed.). Bedford, MA: Digital Press.

Stine, E. A. L., Wingfield, A., & Myers, S. D. (1990). Age differences in processing information from television news: The effects of bisensory augmentation. *Journal of Gerontology: Psychological Sciences, 45,* 1-8.

Strayer, D. L., & Kramer, A. F. (1990). An analysis of memory-based theories of automaticity. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 16,* 291–304.

Suchman, L. (1987). *Plans and situated actions: The problem of human-machine communication*. Cambridge, England: Cambridge University Press.

Sullivan, M. P., & Faust, M. E. (1993). Evidence for identity inhibition during selective attention in old adults. *Psychology and Aging, 8,* 589-598.

Touretzky, D. S., & Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science, 12,* 423–466.

Towse, J. N., & Hitch, G. J. (1995). Is there a relationship between task demand and storage space in tests of working memory capacity? *Quarterly Journal of Experimental Psychology, 48,* 108–124.

Tun, P. A., Wingfield, A., & Stine, E. A. L. (1991). Speech-processing capacity in young and older adults: A dual-task study. *Psychology and Aging, 6,* 3-9.

Turner, M., & Engle, R. W. (1989). Is working memory capacity task dependent? *Journal of Memory and Language, 28,* 127-154.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions.* Cambridge, MA: MIT press.

Van der Linden, M., Brédart, S., Beerten, A. (1994). Age-related differences in updating working memory. *British Journal of Psychology, 85,* 145-152.

Verhaeghen, P., Marcoen, A., & Goossens, L. (1993). Facts and fiction about memory aging: A quantitative integration of research findings. *Journal of Gerontology: Psychological Sciences, 48,* 157-171.

Vicente, K. J., & Rasmussen, J. (1990). The ecology of human-machine systems II: Mediating "direct perception" in complex work domains. *Ecological Psychology, 2,* 207-249.

Weismeyer, M. D. (1992). An operator-based model of human covert visual attention. Doctoral dissertation, University of Michigan, CSE-TR-123-92.

Wechsler, D. (1981). *Manual for the Wechsler Adult Intelligence Scale—Revised.* New York: Psychological Corporation.

Zhang, J., & Norman, D. A. (1994) Representing distributed cognitive tasks. *Cognitive Science, 18,* 87–122.

VITA

Michael Dwyer (Mike) Byrne was born to Michael Dwyer, Sr. and Barbara Jean Byrne in Beaumont Hospital in Royal Oak, Michigan on September 8, 1969. The family lived in Detroit for about two years before moving to the Rochester suburb. In 1976, the family relocated again, this time to the Edina suburb of Minneapolis, Minnesota. In 1977, younger brother Benjamin Mitchell joined the clan. Mike graduated from Edina High School in 1987 and started at the University of Michigan, Ann Arbor in the fall of the same year, determined to become an aerospace engineer. Plans changed somewhat during college and Mike graduated from Michigan in 1991 with a Bachelor of Science in Engineering and a Bachelor of Arts in Psychology, working in David Kieras's lab his last two years in Ann Arbor. Summers of 1990 and 1991 were spent living in Silicon Valley working for Apple and Claris. This was followed by starting graduate school at Georgia Tech in Atlanta in the Fall of 1991. There he earned a Master of Science in Psychology in 1993 and a Master of Science in Computer Science in 1995. After completion of his Ph.D. at Georgia Tech, he will begin a postdoctoral fellowship at Carnegie-Mellon University.