NORTHWESTERN UNIVERSITY

The Effects of Structural Context on Priming

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTORATE OF PHILOSOPHY

Field of Linguistics

By

Meredith Jean Larson

EVANSTON, ILLINOIS

June 2010

ABSTRACT

The Effects of Structural Context on Priming

Meredith Larson

Previous research has found that the recent processing of a linguistic form (e.g. word or syntactic pattern) facilitates its reuse. A separate line of research has found that the appearance of a linguistic form in certain structural contexts (e.g. the focus position of a cleft sentence) can increase the likelihood of a form's reuse. However, these two lines have not explored whether the structural context in which a recently-processed form occurred mediates the facilitatory effects of recent processing. I contend that such mediation exists. Specifically, I propose that the way a structural context is processed affects how memory represents the processing event and how the linguistic forms associated with that structural context are represented in memory. I further contend that differences in these representations affect the subsequent accessibility of the forms.

I present a series of priming studies that support this proposal by showing that the facilitatory effects of a form's recent processing are attenuated when the form occurred in particular structural contexts. By holding time constant and varying only the structural context in which a lexical or syntactic form occurred, I demonstrate that some structural contexts undermine forms' reuse. Specifically, speakers are slower to identify lexical primes occurring in the internal complements of nouns (e.g. the bolded word in "David knew the fact that the man **kissed** Sophia") relative to primes occurring in other structural contexts (e.g. a relative clause

"David knew the man who **kissed** Sophia" or a main clause "As David knew, the man **kissed** Sophia"). Similarly, speakers exhibit less-stable structural priming for primes occurring in the internal complements of verbs ("David knew that the man **kissed** Sophia") relative to primes occurring in other structural contexts.

To clarify the source of structural context's affects on priming behavior, I present a novel activation-based model of language processing. My model describes how linguistic forms are retrieved and manipulated during processing and how the memory traces of linguistic forms are affected by structural contexts. During processing, the processor retrieves encoded memory traces for target linguistic forms. Features of these memory traces, such as how recently they were created and the number of forms (e.g. other words) associated with the memory trace, affect the processor's ability to reuse the target form. I argue that processing some structural contexts (e.g. those containing argument clauses such as the internal complements of verbs) leads to memory traces with more competing forms than the traces generated during the processing of other structural contexts (e.g. those containing adjunct clauses such as relative clauses). The differing number of competing forms associated with the memory trace stems from the processing of different structural contexts and ultimately affects the processor's ability to reuse recently encountered linguistic forms.

ACKNOWLEDGEMENTS

> "Go ahead and go," Haze said, "but remember that the truth don't lurk around every street corner." ~ *Flannery O'Connor, Wise Blood.*

Despite my desire to be parsimonious, I am, ultimately, a sentimentalist. This makes writing acknowledgments particularly difficult. Part of the problem with writing this section is that I have waited much too long to express my gratitude and, as such, have some pent-up thanks to express. Over the past seven years in particular, I have received a great deal of assistance and support from various sources. To these, I now turn to give my sincere thanks.

I begin with my advisor and committee chair, Professor Brady Clark. Brady, you have been a source of steady guidance, patience, wisdom, and humor. You were committed not only to my research but also to my growth as an academic and individual. Without your insight, I would not have found the basis of my argument nor the technical expertise to develop it. Without your patience, I would have been overwhelmed by my impatience and tendency for frustration. And most of all, without your wit and humor, I would have neglected my favorite pastime: laughing. You made sure I mastered simple things I should have mastered long ago (e.g. the use of *hierarchical* and *specifically*) and complex things I could not have mastered without careful guidance (e.g. being able to explain complex models of language processing). There were times when you were more committed to my work and my success than I was. Your encouragement as I wrote grant proposals and fellowship packets made the process more rewarding, and part of the joy of getting the awards was knowing that I was also bringing recognition to your hard work.

When I first started toying around with ideas for my dissertation, you went out of your way to meet with me and brainstorm, thereby helping me to believe in the possibilities of my ideas. However, there were many times over the past couple years when I seriously doubted my work. In these moments, I knew I could trust in your guidance and perspective and that if I listened, I would arrive safely at my destination. Thank you. Your extensive comments, corrections, and suggestions improved the quality of my work and my writing and, moreover, have prepared me to advise and assist those with whom I will work. I hope to take many of your qualities and apply them in my role as an advisor for others. Simply put: I feel lucky to have had you as my advisor, and I am sincerely thankful for all you have done on my behalf.

I am also grateful to the members of my committee. Professor Matt Goldrick, you were the first professor at Northwestern with whom I worked closely to develop research. It was in your psycholinguistics class that I discovered structural priming research, and it was through our work in phonotactics that I developed my original dissertation idea (which is, of course, a far cry from the current proposal). In your classroom and lab meetings, I learned the importance of carefully dissecting arguments and keeping thorough notes. Both of these skills have been invaluable during the dissertation process, and I have tried to pass them along to other students. Beyond these practical elements of academia, you have also demonstrated to me that one can be staunchly empirical without having to give up Monty Python, Daleks, or a soft spot for Bob the robot. Professor Sid Horton, I'd like to thank you for your insights into experimental design and materials. Your experience saved me countless hours and kept me from failed experiments. Furthermore, your chance comments (e.g. "So this is like the classic 'fan effect,' right?" and

"Have you thought where you might publish your findings after you're done writing?") shed light on what I was actually proposing and reminded me that there is more to 'finishing' the dissertation than just submitting it to the graduate school. Professor Kathryn Bock, without your original work in 1986, I could not have started this project. Your groundbreaking research in structural priming and your work with Professor Zenzi Griffin on the time course of structural priming were the basis of my original proposal. Having you on my committee brought me directly into the heart of the debate, and I benefited greatly from your experience and insight. Your enthusiasm and curiosity were contagious, and your interest in my results and conclusions were motivating.

I was lucky to be a part of the linguistics department at Northwestern. Each of my professors here has been of enormous importance, but I wish to mention a couple by name. Professor Gregory Ward, when I applied to Northwestern, I hoped to work with you, and I was lucky to have had the chance to through our work together in the TCP group. You demonstrated how to lead effectively and how to approach situations with assurance, intellectual rigor, a keen wit, and sensitivity to the environment. And thanks to you, I now make sure that everyone distinguishes between the use of *between* and *among*. Dr. Julie Moore, thank you for the opportunity to develop as a teacher and assistant administrator in the ESL program. With your mentorship and trust, I found confidence in the classroom, and through the roles you gave me, I have met life-long friends. You showed me how much someone can accomplish if she plans carefully and remains open to unforeseen possibilities.

Even before coming to Northwestern, I had the support and good council of many

professors. Deserving of special thanks are Professor Donald Hardy and Professor Betty Birner. Hardy, thank you for first noticing my love of grammar and the necessity of graduate school and for having the good humor and perspective to help me find my way through graduate school (relatively) sane and intact. Betty, thank you for being the inspiration for coming to Northwestern and for debating the nature of negation and the devil and the nature of reference and God. There are other professors who have also helped me through either personal encouragement or technical/theoretical expertise. Specifically, I would like to thank Professors Holly Branigan, Victor Ferreira, Zenzi Griffin, Martin Pickering, Hannah Rhode, and Masaya Yoshida for their feedback and encouragement.

I also wish to thank my fellow graduate students for their companionship and collaboration. In particular, I am grateful to Yaron McNabb, who was my first friend and colleague at Northwestern. I cannot imagine how I could have navigated the trials of Northwestern or the Chicago cultural scene without your guidance. Thank you for your continued syntax support, culinary expertise, keen insight into human nature, and Andrew Bird. Thanks also to the graduate student members of the Structure and Meaning work group—Matt Berends, Lewis Gebhardt, and Honglei Wang—for your input; to Lisa Hesterberg for making the recordings; to Jen Alexander, Melissa Baese-Berk, Rachel Baker, Ross Baker, Alex Djalali, Robert Deland, Ryan Doran, Ken Konopka, Jessica Spencer, Celina Troutman, and Kristin Van Engen for your help with and feedback on my numerous experiments and various ideas. In addition to the graduate students in the linguistics department, I would like to thank the graduate students I met from other departments at Northwestern. I am especially grateful to the students I

worked with in the ESL program. I started as your instructor and/or your assistant director, but you became my friends and colleagues. I am particularly grateful to Soo Jeong An, Sohyeon Shim, the officers from the Abu Dhabi language immersion program, and the ISI students of 2009.

But there is more to life than the university. There is, of course, my family. Without you, I literally would not be here. Mom and Dad, thanks for putting me through college and for putting up with me through high school, but moreover, thank you for having faith in me and being proud of me. Seth, you are undoubtedly my favorite brother. I'm glad we didn't buy that condo when we decided to share a place in Chicago seven years ago, but I am delighted that we were able to live together in the most awesome apartment in Rogers Park and that I had your support close at hand during my graduate program. Sage, most aunts and nieces don't get to live with one another as we did. Your presence and joy for life made the darkest of graduate-student days bright. To my cousin Amanda, thank you for offering a refugee in Washington for dissertating and for being the impetus of finding work in the federal government. Without your suggestions, I don't know what direction I would be heading as I leave graduate school.

And there are my friends, who often times might as well be family. Julie Kortum, my life-long friend and now fellow graduate school survivor, thank you for being my biggest cheerleader and my confidant throughout this process. Aimee Hall, I'm glad we were fellow fellows and that our fellowship has followed us this far. Sarah Hartfield, your perspective and compassion helped me recalibrate many many times. Thank you. And then there are the children I was lucky enough to tutor. Lily Par, Biak Thang, Stephen Thawung, and January Sun—you and

your families—have helped me to rediscover the joy of asking questions. Watching you learn how to read, write, and interact in an English showed me the strength of curiosity, play, and hard work. Thank you for calling me *teacher*.

Finally, I wish to thank Irene Sakk, the department assistant for all she has done to keep things moving forward smoothly and Northwestern University and The Graduate School for financial support through research grants and the dissertation year fellowship. I am also grateful for the people and places that became aspects of my daily life and graduate school ritual. Without the familiarity and comfort of my surroundings (and the people that populated my haunts), writing would have been much more difficult. I am grateful for Lake Michigan and its beauty; Andrew Bird and his music; Diet Pepsi and its caffeine; Panera and its friendly staff, bagels, and free wifi; and NPR. Without these little things to fill in the spaces, I would not be a whole person.

An ESL student of mine asked me how to express deep, deep thanks in English. He wanted to know if there was some special phrase or word we used to communicate our gratitude other than "Thank you." The best I could come up with was "I am truly grateful. Thank you." This was not very satisfying then, and it is still not satisfying. Perhaps someday we will correct this oversight in our lexicon, but for now: I am truly grateful to all of you and for each of you. Thank you.

GLOSSARY OF TERMS

ACTIVE CONTROL THOUGHT-RATIONAL (ACT-R): a general model of human cognition that accounts for behaviors ranging from language processing to arithmetic.

ACTIVATION (WEIGHT): the record of a linguistic form's history of use, including its baseline, or resting activation weight, plus any activation boost due to recent processing minus a function of decay.

ASSOCIATIVE ACTIVATION: the boost a form receives from retrieval cues in the current context (Anderson and Schuun 2000).

BASE LEVEL ACTIVATION: a numerical value associated with a declarative chunk that denotes its history of use and is calculated using the formula

$$Bi = \ln \sum_{j=1}^{n} tj^{-d}$$

BUFFER SYSTEM: a system of temporary stores in working memory for information from long-term memory.

CENTRAL EXECUTIVE: a work space in working memory that manages the flow of information that comes from subsidiary slave systems (Baddeley & Hitch 1974).

CONTEXT EFFECT: the hypothesis that the structural context in which a linguistic form occurs affects subsequent use of the form.

CONTROL STATE BUFFER: one of the buffers in the ACT-R architecture (Anderson 1993 *inter alia*); tracks the overall goal of the task (e.g. 'produce sentence with transitive verb').

DECAY: a constant function of deactivation for a linguistic form the onset of which begins when the form is no longer being processed.

DECLARATIVE CHUNKS (LINGUISTIC): the memories that comprise declarative knowledge; memory templates for various grammatical units, which are labeled according to their maximal projections; each chunk consist of feature-value pairs, e.g. 'num : SG' would be the feature 'number' and the values 'singular.'

DECLARATIVE CHUNK BUFFER: one of the buffers in the ACT-R architecture (Anderson 1993 *inter alia*); holds whatever particular declarative chunk is involved with the current goal state; can be empty given the current state of processing.

DECLARATIVE KNOWLEDGE: the facts we know, such as the meaning of a particular word; generally consciously accessible and can be learned explicitly.

EPISODIC BUFFER: one of the subsidiary slave systems that feeds information to the center executive; a temporary store of episodic information such as the temporal ordering of events (Baddeley 2000).

IDENTITY PRIMING (also **REPETITION PRIMING):** the processing facilitation an item receives because the same lexical item (lemma or lexeme) was recently encountered.

LEXICAL PRIMING: the facilitation in processing a lemma or lexeme receives due to the processing of lemma or lexeme.

LONG-TERM MEMORY (LTM): the part of memory that holds all previously experienced

encounters (e.g. previously processed words); is unlimited; feeds information to working memory and receives information from working memory.

PHONOLOGICAL LOOP: one of the subsidiary slave systems that feeds information to the center executive; a temporary store of phonological information such as the linear order of sounds in a speech stream (Baddeley 2000).

POPPING (!POP!): the production rule that removes a chunk from the retrieval buffer making it available for unification; occurs only when the chunk contains no open values in its feature-value pairs.

PRIMING: the facilitation a form receives in processing due to having recently been processed.

PRIMING ACCORDING TO RICE (PRICE): the processing of both a prime form and its structural context affects how the form is represented, and differences in these representations affect subsequent priming behavior.

PROBLEM STATE BUFFER: one of the buffers in the ACT-R architecture (Anderson 1993 *inter alia*); holds the particular subgoal being addressed (e.g. 'produce subject for transitive verb' or 'produce a determiner phrase').

PROCEDURAL KNOWLEDGE: the knowledge necessary for performing actions; this knowledge is represented as a series of production rules that are not consciously accessible to the individual.

PRODUCTION RULES: the memories that form procedural knowledge; take the form of condition-action units or IF-THEN statements that specify goals, change states, and often lead to the creation of new subgoals; learned via analogy and are acquired implicitly.

PRODUCTION RULE STRENGTH: a numerical value that reflects a production rule's history of use and is calculated using the equation

$$S_p = \ln \sum_{j=1}^{n} tj^{-d}$$

RECENCY EFFECT: the hypothesis that linguistic forms that have occurred recently exert more influence over subsequent behavior than those that have not occurred recently.

RECENT INTERACTION WITH CONTEXT EFFECT (RICE): the hypothesis that the effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed.

RELATION: any additional boost in activation weight that a declarative chunk receives from the other chunks in its context.

RETRIEVAL: the process that selects memories and brings them into focus; is affected by relation, activation level, and a function of random noise for the retrieval if chunks; is affected by utility, strength, and a function of random noise for the retrieval of production rules.

RETRIEVAL BUFFER: one of the buffers in the ACT-R architecture (Anderson 1993 *inter alia*); holds declarative chunks retrieved from memory that are currently involved processing.

SEMANTIC PRIMING (both 'associative' and 'semantic'): the facilitation in processing that a word, referent, or concept receives due to the processing of a related word, referent, or concept.

SLAVE SYSTEMS: subsidiary buffers for the central executive buffer in the Baddeley and Hitch (1974) buffer system model.

SPREADING ACTIVATION: activation boost that results from the processing of a related linguistic form, for example, having processed the word *cat* will activate the sounds linked to the form (/k/, /æ/, /t/) and the semantic information linked to the form (e.g. PET).

STRUCTURAL PRIMING: the facilitation in processing that a structural alternate receives due to the processing of a structural alternate.

STRENGTH: see PRODUCTION RULE STRENGTH.

TOTAL ACTIVATION WEIGHT: a numerical value that reflects the activation of a declarative chunk as determined by its base level activation plus any additional weight from its context (see RELATION). It is calculated using the equation

$$A_i = B_i + \sum_j W_j S_{ji}$$

UNIFICATION: the process by which two structures are merged to generate a new, more specified structure that contains the union of all the feature-value pairs of the original structures.

UNIFICATION CHAIN: a series of unification cycles that occur during the resolution of a single subgoal structure.

UNIFICATION CYCLE: a successful unification operation during the processing of a sentence.

UTILITY: a numerical value that reflects the expected gain associated with firing a production rule minus

the expected cost associated with the rule. It is calculated using the formula

$$\mathbf{U = PG - C}$$

VISUOSPATIAL SKETCHPAD: one of the subsidiary slave systems that feeds information to the center executive; a temporary store of visual and spatial information such as shapes and their location in the environment (Baddeley 2000).

WORKING MEMORY (WM): the part of memory that holds currently active information, is limited, receives information from long-term memory, and feeds information to long-term memory.

DEDICATION

Against the assault of laughter nothing can stand.
~ *Mark Twain*

To my brother, Seth, and my niece, Sage.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 CHAPTER

Introduction

> What is courageous in one setting can be foolhardy in another and even cowardly in a third. ~ *Joseph Epstein*

Consider an arabesque rug, with its many colors—like words—combining to make intricate, recurrent patterns. Whether a particular color appears light or dark depends on the colors around it, and although we have a memory for the overall pattern, the contribution of any single strand often eludes us. Similarly, language processing constantly weaves words and structures into dynamic, interconnected patterns—the interaction among the linguistic forms determining the way a sentence is comprehended or produced. Because of this interconnectedness, no single form can be completely decontextualized or divorced from the way it was initially processed or from the larger structural pattern in which occurred. The context in which a form occurs and our memory for this occurrence affect the way each word is interpreted, organized, and subsequently recalled.

Although the recall of a sentence or linguistic form can be explicit, as in remembering a particular name, it is often implicit. The verbatim recall of surface details such as the exact ordering of adjectives and syntactic structure of a sentence after a few seconds has long been regarded as impossible, though the meaning of a sentence may be longer lasting (Altman 2001; Anderson, Budiu, & Reder 2001; Caplan 1972; Craik & Tulving 1975; Jarvell 1971; Kintsch

1974, 1998a,b; Sachs 1967). Still, the surface details and syntactic structure of a sentence influence subsequent behavior with the linguistic forms that comprised the sentence, and this influence of the sentence's structure affects both immediate and long-term behavior. A sentence, its structure, or even the particular words within it do not need to be explicitly recalled for them to shape subsequent behavior. Simply encountering or processing a sentence changes the way speakers later use the words and structures that occurred in the larger sentence.

One common linguistic behavior that demonstrates this type of change is called PRIMING. Priming refers to the facilitation a form receives in processing due to having recently been processed. It occurs at every level of language processing: the phonological, lexical, syntactic, conceptual, etc. Priming may even play a role in language learning and language change (e.g. Becker, Moscovitch, Behrmann, & Joordens 1997; Bock 1986a, b; Bock & Griffin 2000; Bock & Kroch 1989; Jäger & Rosenbach 2008a, b; Dell 1986; Hutchinson 2003; Lucas 2000; Pickering & Ferreira 2008; McNamara 2005). For example after reading the sentence "Sophia is my tabby cat," speakers respond more quickly to the word *cat*, or to a semantically-related word (e.g. *dog*), or phonologically-related word (e.g. *hat*) (see McNamara 2005 for a review of lexical/semantic priming). The basic observation is that after processing a linguistic form—either through comprehension or production—speakers respond to the same form or related forms more quickly.

However, in many models of priming (e.g. Pickering & Branigan 1998), the amount of

time (delay) between the priming event and the subsequent target event matter.[1] When the priming and target events are close to one another (e.g. the priming event immediately precedes the target), the prime form is more likely to influence linguistic performance in the target task than if the two were separated by many other events. Generally speaking, the closer the two events are, the stronger the priming effect is. This 'recency' effect has long been noted as a relevant factor in determining whether and to what degree a linguistic form influences behavior (e.g. Bock 1986b; Bjork & Whitten 1974; Brennan & Clark 1996; Deese & Kaufman 1957; Murdock 1962; Davelaar, Goshen-Gottstein, Haarmann, Ashkenazi, & Usher 2005; Howard & Kahana 1999; McNamara 2005; Pickering & Branigan 1998), leading to the hypothesis below:

> **Recency Effect**
> Linguistic forms that have occurred recently exert more influence over subsequent behavior than those that have not occurred recently.

Although recency is a relevant factor in priming, it is not the only factor. Other aspects, such as the overall frequency of a form, also affect the extent to which a form primes.[2] Still, the amount of time between the prime and target event is a common predictor of the amount of priming.

Another factor that affects a form's influence on subsequent behavior is how the form is processed in relation to its larger structural context. The structural context in which a form occurs can affect subsequent processing both in production and comprehension. For instance, structural context can affect verb agreement errors (e.g. "the code to the alarms were stolen", see

[1] In models of priming that focus more on the implicit learning and long-term effects of priming (e.g. Bock & Griffin 2000), time is less relevant. They contend that the effects of priming are long-lasting and stable though there is a short-term lexical boost.

[2] For example, both experimental data and computational models have found that less-frequent linguistic forms show greater priming effects then more-frequent forms (e.g. Anderson & Schuun 2000; Chang, Dell, Bock, & Griffin 2000; Ferreira 2003; Jaeger & Snider 2008; Van Rijn & Anderson 2003).

Bock & Cutting 1992; Bock, Nicol, & Cutting 1999; Bock & Miller 1991; Pearlmutter, Garnsey, & Bock 1999 *inter alia*), speech onset latencies and prosodic contours (e.g. Ferreira 1991, Kleinhow & Smith 2000, Krivokapic 2007, Watson & Gibson 2004), the ordering of constituents (Arnold, Wasow, Losongco, & Ginstrom 2000; Wasow 2002), and the resolution of pronouns and gaps (e.g. Clifton, Kennison, & Albrecht 1997; Nicol & Swinney 1989; Sturt 2003). For example, cleft structures like English *it*-clefts and *wh*-clefts (in (1) and (2) respectively) facilitate anaphor resolution. The proposed reason for this facilitation is that cleft structures produce stronger, more distinct memory representations for elements in their focused positions (bolded) than for those in 'deemphasized positions' (italics). These 'deemphasized' positions refer to the elements that are structurally subordinate (i.e. embedded) and pragmatically backgrounded in focus structures.[1]

(1) It-cleft sentence
It was **the robin** that ate *the apple*.

(2) Wh-cleft sentence
What *the robin* ate was **the apple**.

Research suggests that being in the focus position can facilitate subsequent processing and response times for anaphors and can affect speakers' predictions for and completions of subsequent discourse (Almor 1999; Almor & Eimas 2008; Birch, Albrecht & Myers 2000; Birch & Garnsey 1995; Foraker & McElree 2007; Morris & Folk 1998). For example, Almor (1999) found faster reading times for noun-phrase anaphors (e.g. "the bird" in (3) and (4)) when their antecedents (e.g. "the robin") were in the focus position of a preceding cleft sentence.

[1] I will reserve the term *non-focus* positions to refer to elements that are in neither of these positions, such as "the robin" in the sentences "**The robin** ate the apple" or "The cat ate **the robin**."

(3) It was **the robin** that ate the apple. The bird seemed satisfied.

(4) What the robin ate was **the apple**. The bird seemed satisfied.

In the *it*-cleft (3), "the robin" is in focus position, whereas in the *wh*-cleft (4), it is in a deemphasized position. When speakers arrived at the noun-phrase anaphor "the bird," their reading times were shorter following sentences in which the antecedent ("the robin") was in focus position (sentences like (3)) versus when it was not in focus position (as in (4)). Had only recency mattered, then "the robin" in (3) and "the robin" in (4) should have led to the same facilitation (priming) at "the bird" because they both occurred in the same linear position, meaning they were the same number of words away from the target. The fact that a difference appeared suggests that reactivating an antecedent (prime) is easier when the antecedent occurred in the syntactic focus position in the priming sentence. The results of these studies lead to the following hypothesis, which I call the CONTEXT EFFECT.

> **Context Effect**
> The structural context in which a linguistic form occurs affects subsequent use of the form.

Given the hypotheses stated above, I contend that there are two crucial factors that predict a linguistic form's effect on subsequent linguistic performance:

i) when it was processed (recency) and
ii) how it and its context were processed (structural context)

In particular, I contend that there is a generally stable effect of recency, in that forms that were recently encountered influence behavior more than those not recently encountered, but this

general effect is mediated by processing differences caused by the different structural configurations in which primes can appear. This leads to the hypothesis best captured by the RECENT INTERACTION WITH CONTEXT EFFECT hypothesis of language processing:

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed.

In making this hypothesis, I focus on how the processing of a structural context mediates the effects of recency. As noted above, recency is a well-known predictor of a form's likelihood to influence subsequent linguistic behavior. Forms that have occurred recently are more likely to affect behavior than those that have not occurred recently. However, the way a sentences is processed (as is discussed in Chapter 2) affects the way linguistic forms (e.g. words and the rules that combine them to form grammatical units) are retrieved and unified. This, in turn, affects the accessibility of particular forms during subsequent processing tasks.

The claim that the structural context of a prime may affect subsequent behavior raises several important questions such as

(i) Does structural context affect the retrieval of both lexical and structural information similarly?

(ii) What should count as a relevant structural context (e.g. linear position or hierarchical position)?

and the closely-related question

(iii) Which contexts facilitate and which inhibit various linguistic behaviors?

The short answer to (i) is that the aforementioned research shows that the accessibility of semantic information may be enhanced by features of the structural context. However, whether

structural contexts also affect the way specific word forms or syntactic patterns influence subsequent performance is unknown. The short answer to (ii) and to (iii) is that we don't know for sure, but we have a potential starting point. Previous work in historical linguistics, language modeling, and theoretical linguistics suggest that matrix and embedded positions are different and that this distinction may be useful to begin our exploration for the relevant contexts (Lightfoot 1991, Pearl 2005, Pearl & Weinberg 2007, Pintzuk 1999, Pintzuk & Taylor 2006, Stockwell & Minkova 1991). For example, Lightfoot (1991) and Pearl and Weinberg (2007) argue that linguistic forms occurring in embedded VPs are not informative to language learners and that forms occurring in matrix clauses are privileged. Likewise, Pintzuk (1999) found that matrix and embedded positions show different patterns of language change, namely that matrix clauses make greater use of innovative forms than embedded ones. This suggests that clausal boundaries and hierarchical patterns may mediate the subsequent use of linguistic forms.

However, it is not perfectly clear why embedding should affect the accessibility of linguistic forms. Furthermore, various forms of embedding may have different effects on processing. Previous research in both comprehension and production have found differences between the processing of relative clauses and both the internal complements of nouns (henceforth *noun complement clauses*) and the internal complements of verbs (henceforth *verb* complement clauses) (e.g. Boland 2005; Chambers, Tanenhaus, & Magnuson 2004; Clifton Speer, & Abney 1991; Gibson 1998, 2000, 2003; Gibson, Desmet, Grodner, Watson & Ko 2005; Grodner & Gibson 2005; Hudgins & Cullinan 1978; Kennison 2002; McElree & Griffith 1995; Shaprio, Oster, Garcia, Massey, & Thompson 1992; Trueswell, Tanenhaus, & Garnsey 1994;

Tanenhaus, Spivey-Knowlton, Eberhard, & Sedivy 1995; van Gompel, Pickering, & Traxler 2001; Watson, Breen, & Gibson 2006; Watson & Gibson 2004) as well as for relative clauses and adverbial clauses (e.g. Gayraud & Martinie 2008). This research suggests that relative clauses are generally more difficult to comprehend than noun or verb complement clauses and that relative clauses are more integrated with other elements of the sentence than adverbial clauses. Given these findings, it may not be simply a matter of embedding that affects accessibility but also the form of embedding.

The possible answers to questions about the relevant contexts and their possible effects are profuse. In order to limit the number of questions, in this dissertation, I focus on how elements of the structural context mediate the effects of recency by decreasing the likelihood of priming. In particular, I explore how the processing of different structural contexts (e.g. matrix clauses and relative clauses) affects the priming behavior of linguistic forms that appear within them.[1]

To further limit the scope of this dissertation, I explore the effects of structural context on priming only on lexical items (lexical priming) and sentence structures (structural priming). One reason for exploring these two types of priming is that reactivating a specific lexical item or syntactic structure may be different from reactivating a semantic referent, as explored in the focus research mentioned above (Almor 1999; Almor & Eimas 2008; Birch, Albrecht & Myers 2000; Birch & Garnsey 1995; Foraker & McElree 2007; Morris & Folk 1998). These three types of reactivation occur at different levels of language processing, with the activation of a referent occurring at the conceptual level, a word occurring at the lexical level, and a syntactic form occurring at the combinatorial level.

[1] I do not consider non-structural forms of context, such as discourse-level information structure.

The studies that I present in Chapters 3 and 4 support the RICE hypothesis. I demonstrate that primes occurring within certain structural contexts have less of an effect on subsequent production than primes occurring in other structural contexts. In Chapter 3, I show that lexical primes in noun complement clauses show less priming than those occurring in verb complement clauses, relative clauses, or matrix clauses. For example, the bolded word in (5) facilitates response time at the target word (italicized) less than the same word in the other structural contexts ((6) – (8)).

(5) Prime in the internal complement of a noun (noun complement clause)
The manager reported the fact that the secretary **bought** the supplies for the owner.
Target: *bought*

(6) Prime in the internal complement of a verb (verb complement clause)
The manager reported that the secretary **bought** the supplies for the owner.
Target: *bought*

(7) Prime in a relative clause
The manager liked the secretary who **bought** the supplies for the owner.
Target: *bought*

(8) Prime in a matrix clause
The manager left the request, and the secretary **bought** the supplies for the owner.
Target: *bought*

In Chapter 4, I show that structural primes occurring in verb complement clauses show less priming than those occurring in relative clauses or matrix positions. For example, the bolded double object form of the dative alternation[1] in (9) is less likely to be repeated in subsequent productions than the same construction in sentences (10) – (12).

[1] The dative alternation is the variable ordering of objects following dative verbs such as *give, hand, show*, and *buy.* The two alternates are the double object form "promised the duchess the rubies" and the prepositional dative form "promised the rubies to the duchess."

(9) Prime form in the internal complement of a verb (verb complement clause)
The report declared the fact that the duke promised the duchess the rubies.

(10) Prime form in a matrix clause preceded by and adverbial clause
As report declared, the duke promised the duchess the rubies.

(11) Prime form in a matrix clause with subject-modifying relative clause
The duke who liked the king promised the duchess the rubies.

(12) Prime form in relative clause
The king liked the duke who promised the duchess the rubies.

The structure of the dissertation is as follows. In Chapter 2, I present the motivation for the RICE hypothesis as well as introduce the model of memory and language processing from which the predictions I explore in Chapter 3 and 4 are derived. Chapters 3 and 4 present studies that test the predictions of the model that I present in Chapter 2 for lexical priming and structural priming respectively. Chapter 5 contains a discussion of the overall results and future avenues of exploration.

2 CHAPTER

The Importance of RICE

Active Evil is better than *Passive Good*. ~ *William Blake*

The RICE hypothesis contends that subsequent use of linguistic form is affected both by when a linguistic form was previously processed (henceforth *recency*) and by how it was processed in relation to the larger structural context in which it occurred (henceforth *structural context*). In particular, RICE states that recently encountered forms influence subsequent performance but that this tendency is mediated by the structural context in which the form occurred during the recent processing.

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed.

In the current chapter, I present evidence—both theoretical and behavioral—that offers support for the general claims that both recency and structural context affect subsequent linguistic behavior (section 1). In section 2, I argue for the union of research on the effects of recency and structural context under RICE. In section 3, I present the components of the model of memory I use throughout the current work. In this section, I further develop the RICE hypothesis and its predictions about structural contexts and their effects, given the model presented in section 3. Section 4 concludes with a summary of the chapter and a comparison of the RICE hypothesis and the standard account of priming.

1. The effects of recency and structural context on linguistic behavior

A commonly made observation is that linguistic forms that were recently encountered are likely to be reused or to reoccur. This tendency can be found at all levels of language processing, from sounds, to words, to sentence types. For example, Levelt and Kelter (1982) found that speakers tend to reuse the words and patterns of their interlocutors. When speakers in their study heard a question such as "At what time do you close?," they were more likely to say "At five-o'clock" than if they had first been asked "What time does your store close?" The speakers in the study used the same pattern (i.e. the prepositional phrase "at . . .") as the one they recently encountered. Levelt and Kelter explained this behavior as a form of word-matching. Others have built on this finding and contend that it is not just a matter of reusing the same words but also reusing the same syntactic structures independent of particular words (e.g. Bock 1986b). This sort of matching behavior may arise for a multitude of reasons, ranging from establishing common ground, to aligning discourse and coordinating interlocutors, or even to preserving registers (e.g. Garrod & Anderson 1987, Tannen, 1987, Weiner & Labov 1983). It may be a form of implicit learning or a source of language change (Bock & Kroch 1989; Bock & Griffin 2000; Ferreira & Bock 2006; Jäger & Rosenbach 2008a, b).

Although many explanations for such behavior have been put forth, there is one common assumption about it: recent processing of a linguistic form prepares, or primes, the processor to reuse it. That is, the processing of a form facilitates the subsequent processing of the same or a similar form. McNamara (2005) defines such facilitation as PRIMING, "an improvement in

performance in a perceptual or cognitive task, relative to an appropriate baseline, produced by context or prior experience" (p 3). Prior experience can refer, for example, to the number of times a person has encountered a particular word throughout his or her life or to how recently a person encountered a particular word. The effects of frequency and recency have been noted by numerous studies, and they both can affect the way words are understood and sentences are parsed (e.g. MacDonald, Just, & Carpenter 1992; McKoon & Ratcliff 1998; Trueswell, Tanenhaus, & Garnsey 1994). The RICE hypothesis is primarily concerned with recency and its effects on subsequent reuse of a linguistic form.

Although the RECENCY EFFECT (i.e. the tendency of recently processed linguistic forms to facilitate subsequent behavior) is a crucial factor for priming, it is not specific to linguistic performance. Recency effects have been noted by cognitive psychologists for decades. Language users tend to retrieve items at the end of word lists more quickly and more reliably during free recall tasks than items in other parts of word lists (e.g. Bjork & Whitten 1974; Deese & Kaufman 1957; Murdock 1962; Davelaar, Goshen-Gottstein, Haarmann, Ashkenazi, & Usher 2005; Howard & Kahana 1999). Since the early days of this line of research, theorists have argued that recency effects arise because particular words or syntactic patterns are still active in memory. In other words, the memory (conscious or unconscious) of the encounter is still accessible (e.g. Atkinson & Shiffrin 1971).

Outside of word-list recall, recency effects can be found in facilitated response times to categorization tasks, lexical decisions, item recognition, pronunciation tasks, and sentence completion tasks (Bock 1986a, b; McNamara 1992, 2005; Rips, Shoben, & Smith 1973 *inter*

alia). Processing a word can facilitate the processing of the same word (e.g. Perea & Rosa 2002), a rhyming word (e.g. Meyer, Schvaneveldt, & Ruddy 1974), a morphologically similar word (e.g. Plaut & Gonnerman 2000, Veríssimo & Clahsen 2009), or an associatively or semantically related word (e.g. Forster 1981; McKoon & Ratcliff 1992; McNamara 2005; Perea & Rosa 2002; Ratcliff & McKoon 1994; Wang, Dong, Ren, & Yang 2009). For example, processing a structural form (e.g. the passive construction "The house was struck by lightning") can facilitate the reuse of the same construction (e.g. Bock & Loebell 1990, Pickering & Ferreira 2008, Weiner & Labov 1983).

Three commonly studied forms of priming (Bock 1986; Branigan, Pickering, Liversedge, Stewart, & Urbach 1995; Estival 1985; Jaeger & Snider 2008*;* Pickering & Ferreira 2008; Snider 2008) include:[1]

- LEXICAL PRIMING (e.g. 'identity' or 'repetition' priming): facilitation in the processing of a lemma or lexeme due to the recent processing of the same lemma or lexeme (Chapter 3)
- SEMANTIC PRIMING (both 'associative' and 'semantic'): facilitation in the processing of a word due to the recent processing of a related word (Chapter 3)
- STRUCTURAL PRIMING (sometimes called SYNTACTIC PRIMING or STRUCTURAL PERSISTENCE): facilitation in the use of a syntactic form due to the recent processing of the form (Chapter 4)

Each of these types of priming focuses on a different level of language processing, but each of them reaches a similar conclusion: recency matters.

This leads to the question of why recency matters. As mentioned previously with respect to list learning, the claim is that forms that have occurred recently are easier to recall because

[1] Phonological priming is not considered in the present work.

they are still 'salient' or 'active' in memory. However, there is evidence that suggests that more than just recency affects a form's saliency or activation. The structural context in which the form occurred also matters (henceforth the STRUCTURAL CONTEXT EFFECT). Previous research has found that elements of the structural context can serve to heighten a form's activation making it easier to retrieve and, hence, more likely to facilitate subsequent behavior (Birch & Garnsey 1995; Clifton, Kennison, & Albrecht 1997; McKoon, Ratcliff, Ward, & Sproat 1993; Nicol & Swinney 1989; Spivey, Tanenhaus, Eberhard, & Sedivy 2002; Sturt 2003; Swinney 1979; van Berkum, Brown, & Hagoort 1999). For example, research on focus constructions, such as *it*-clefts (as in (1)) and *wh*-clefts (as in (2)), has found that words in focus positions (the bolded expressions below) lead to greater facilitation at the target word (i.e. the subject of the continuation sentence in italics below) than words in 'deemphasized' positions, i.e. words/phrases in structurally subordinated, pragmatically backgrounded positions in cleft sentences, such as the italicized words in (1) and (2)[1] (Almor 1999).

(1) *It*-cleft sentence
It was **the robin** that ate *the apple*.

Continuation: The bird/The fruit…

(2) *Wh*-cleft sentence
What *the robin* ate was **the apple**.

Continuation: The bird/The fruit…

Had recency been the only factor affecting the speed at which speakers responded to the noun-phrase anaphors, then "the robin" in (1) and (2) should have facilitated responses to "the bird"

[1] The term 'deemphasized' or 'defocused' has been used to describe the non-clefted information (Birch, Albrecht, & Myers 2000).

equally. Likewise, the "the apple" in (1) and (2) should have facilitated responses to "the fruit" equally. However, there was a difference: primes in focus positions facilitated processing more than those in deemphasized positions. In other words, "the robin" in the focus position in the *it*-clause (1) facilitated responses to "the bird" more than "the robin" in the deemphasized position in the *wh*-clause (2). Henceforth, I refer to this facilitation for items in focus positions the FOCUS EFFECT. Other studies have also found focus-position effects for performance differences in tasks ranging from lexical decisions to story continuations (Almor & Eimas 2008; Birch, Albrecht, & Myers 2000; Birch & Garnsey 1995; Foraker & McElree 2007; Morris & Folk 1998). The boosted activation of a linguistic expression that appears in a focus position makes that linguistics expression easier to locate or reactivate when a semantically related expression (e.g. a coreferential pronoun) is subsequently processed.

However, there are notable exceptions to the focus effect. For example, Birch, Albrecht, and Meyers (2000) found that forms in focus positions (e.g. (3)) were not recognized any faster than those in neutral positions (e.g. (4)), indicating that focus position in and of itself may not affect the retrievability of specific words.

(3) Antecedent in focus position
It was **the mayor** who refused to answer a reporter's question.

(4) Antecedent in neutral position
The mayor refused to answer a reporter's question.

However, they did find that linguistic expressions presented in focus positions showed greater long-term effects than those in either neutral or deemphasized positions. These findings suggest

that focus effects may arise only after a delay and that they may be sensitive to pragmatic and task demands.

Almor and Eimas (2008) also found that focus effects are not as consistent or as stable as previously assumed. They found that focus effects can disappear depending on the task one is using. Using both repeated noun-phrase anaphors (i.e. the antecedent and the noun-phrase target were the exact same phrase as in (5)) and non-repeated noun-phrase anaphor (as in (6)), they tested the effects of focus constructions in both an auditory lexical decision task and a recall task.

(5) Repeated noun-phrase anaphor
It was **the bird** that ate the apple. The bird seemed very satisfied.

(6) Non-repeated noun-phrase anaphor
It was **the robin** that ate the apple. The bird seemed very satisfied.

They found facilitation for focus constructions in the lexical decision task, regardless of whether the noun-phrase anaphor was a repeated phrase or a non-repeated phrase. However, they suggest that there may also have been a first-mention effect aiding in the lexical decision. In the recall task, they found no effect of focus position for either repeated or non-repeated noun-phrases, though they did find that repeated noun-phrases led to worse recall overall. They argue that, although there may be facilitation for reading times or lexical decisions at the noun-phrase anaphor following focus constructions, recall and long-term representations of the antecedent are not affected by whether the antecedent occurred in focus position or deemphasized position. They state that "focus only affects memory under conditions which are not typical of regular discourse" (e.g. when the anaphor is the exact same phrase as the antecedent rather than a pronoun or other expression) (p 222).

What we can take away from the above work on focus constructions is that being in a focus position facilitates performance in some tasks relative to being in a deemphasized position. Other structural contexts, such as different forms of embedding, may affect behavior in radically different ways.

2. Why we need RICE

Previous work on recency effects and structural context effects suggest that the processing of a linguistic form influences subsequent use of that form. Firstly, if a form has been recently encountered, it is more likely to reoccur or to be responded to more quickly than if it hadn't occurred recently. Secondly, if a form occurs in some structural configurations such as the focus position of a focus construction, subsequent behavior with the form may also be facilitated. However, these two lines of research have not been integrated. As a consequence, we do not have a complete picture of how the processing of a linguistic form affects subsequent linguistic behavior. Work that explores recency effects often does not manipulate the structural context in which the prime forms occur. At the same time, work on structural context effects does not manipulate recency.

At the end of the day, all we really have is an indication that recent processing or being in certain structural contexts aids in subsequent processing, and we are left with unanswered questions, such as

(i) Does the processing of a structural context affect different types of priming in the same way, meaning that if the structural context aids in lexical priming does it also aid in structural priming?

and

(ii) Which structural contexts facilitate priming and which structural contexts inhibit priming?

The studies presented in Chapters 3 and 4 address these questions and explore the following hypothesis:

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed

The RICE hypothesis states that structural configurations create patterns of word retrieval and integration. Differences in these patterns of retrieval and integration affect the subsequent accessibility of memories for particular forms that appear within those structural configurations. Differences in subsequent behavior/performance with a form arise due to differences in the levels of accessibility, which are ultimately affected by the way memory encodes and retrieves processing events. Before demonstrating how structural context may mediate the general facilitating effects of recent priming, I present the model of language processing and memory from which RICE is derived. In section 3, I cover some of the basic components of activation-based models of language processing. After presenting this, I demonstrate how these elements interact to affect linguistic behavior.

3. Memory

One of the most common assumptions theorists and researchers have about memory is that it has a limited processing capacity but virtually limitless storage capacity. Although we can store a life's worth of memories, we can consciously access or use only a finite set of memories at any

particular time (Miller 1956). These limitations have led to a distinction between the aspects of memory that are more long-lasting and harder or slower to retrieve (i.e. LONG-TERM MEMORY, LTM) and those that are more fleeting and easier or quicker to manipulate. This second aspect of memory is often referred to as SHORT-TERM MEMORY (STM) or WORKING MEMORY (WM) (e.g. Allen & Baddeley 2009; Baddeley 1986, 1992; Baddeley & Hitch 1974; Baddeley & Logie 1999; Becker 1994; Cowan 1999, 2001; Ericsson & Kintsch 1995; Halford, Wilson, & Phillips 1998; Hitch & Logie 1996; Just & Carpenter 1992; Just, Carpenter, & Keller 1996; King & Just 1991; Lewis 1996; Miyake & Shah 1999; see Cowan, Nelson, & Chen 2009 for a dissenting opinion about the distinction). For ease of discussion and because terms such as 'working memory' and 'long-term memory' are so ubiquitous, I use these terms to refer to the memories that are limited in number and currently active or in use (working memory, WM) and those that may be unlimited in number but are not currently active or in use (long-term memory, LTM).

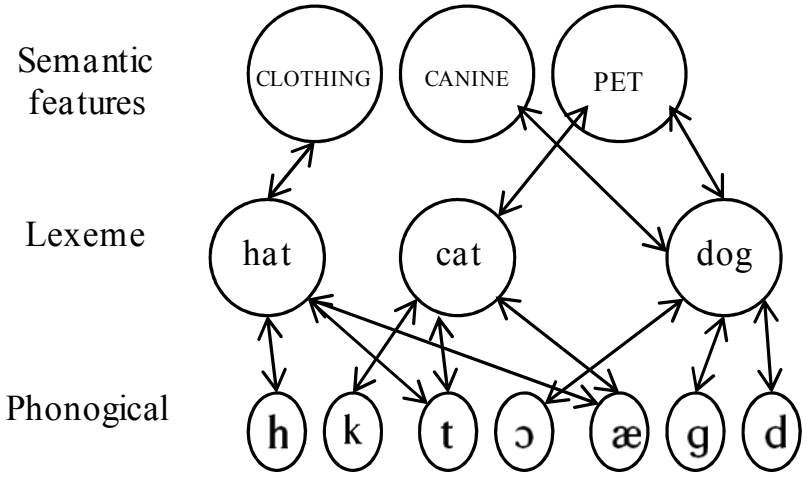
3.1 Activation-based models of memory and language processing and priming

The model of language processing that I present in this section adopts many of the features and assumptions of activation-based processing models. This set of models assumes that because cognitive resources in WM are limited, WM cannot search LTM exhaustively. Rather, it relies on the relative activation of forms (e.g. linguistic forms) in LTM during processing (Collins & Loftus 1975; Dell, Chang, & Griffin 1999; Elman 1990, 1991; Garrett 1982, 1988; Kintsch 1988a,b; Levelt 1989). In these models, each linguistic form has an associated record of its history of use, called its ACTIVATION WEIGHT. This weight helps determine how easily a form is retrieved or how likely a form is to be retrieved. As the use of a form increases in frequency

(e.g. the form is repeatedly retrieved), its resting activation, or BASE LEVEL ACTIVATION, weight increases. This gradual accrual of weight accounts for phenomena such as frequency effects (e.g. more frequent words are processed more quickly than less frequent words) and long-term learning effects. Retrieval also has short-term effects on a form's activation weight. After being retrieved from memory, the form's activation weight increases momentarily. This activation can then spread from the specific form to related forms through a process called SPREADING ACTIVATION, or cascading activation (Collins & Loftus 1975, Dell 1986, Dell & Gordon 2003, Rapp & Goldrick 2000).

For instance, assume memory contains representations for the semantic, lexical, and phonetic information of words (represented below in Figure 2.1 as the 'semantic features,' 'lexeme,' and 'phonological' levels). Each node connects to many other nodes, and the same node can be shared by different words. Consider the small network below with its various nodes.

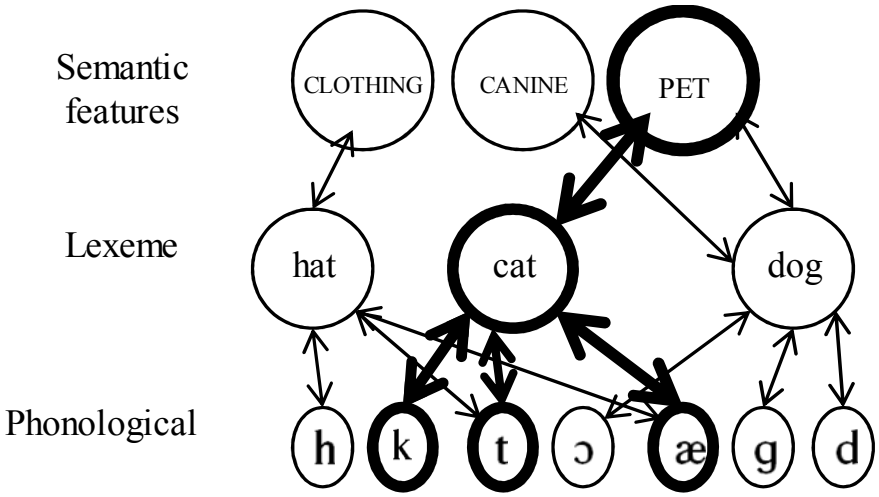
Figure 2.1: Representations of linguistic forms in memory



Here we see that both *cat* and *dog* link to the semantic node for PET, but only *dog* links to CANINE. Similarly, both *cat* and *hat* link to the phonological node /t/, but only *cat* links to /k/.

During the processing of "Sophia is my tabby cat," the word *cat* is retrieved. This retrieval activates the *cat* node (lexeme level)[1] along with its phonological and semantic units, as denoted by the bolded lines in Figure 2.2 below.

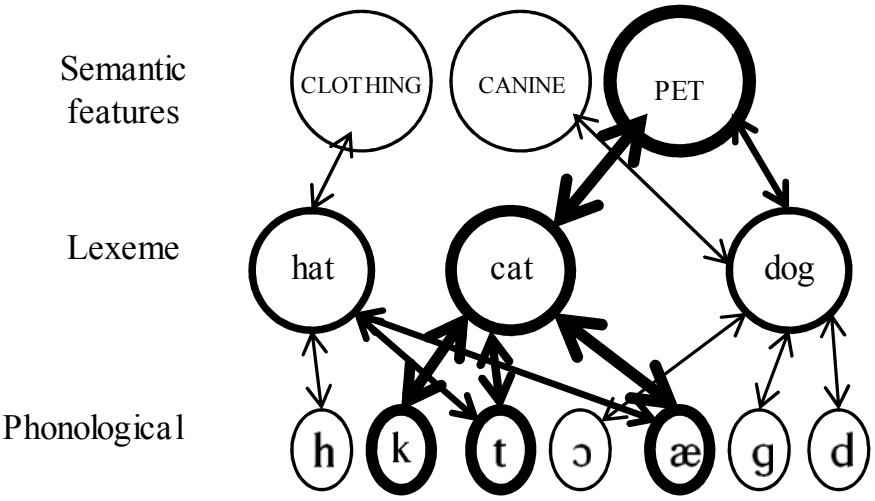
Figure 2.2: The activation of *cat*



In this figure, the nodes are connected to one another in a complex network with nodes at one level connecting to multiple nodes at other levels. The activation of one of these nodes spreads to the other connected nodes to which they are connected. These connected nodes receive a slight boost in their activation, as denoted in Figure 2.3 by the lesser-bolded lines and circles.

[1] The lexeme level refers to the level that contains the abstract unit for a word, e.g. WALK, that corresponds to the set of possible forms of a word, e.g. *walk, walked, walks*, which count as the lemma for the lexeme.

Figure 2.3: Spreading activation from *cat*



The activation of the word *cat* activates its associated semantic information (e.g. PET). Now that PET is active, other nodes that are connected to PET, e.g. *dog*, receive a slight increase in their activation. Subsequently, words that are connected to the retrieved word—either semantically or phonologically—show greater facilitation due to the slight increase in their activation. However, this activation boost (for both the retrieved form and the forms connected to it) wanes.

One important element of the type of model I am adopting is that the different levels of processing are connected and can, therefore, inform one another, but they are not fully dependent on each other. They do not need process information in lockstep (Allen & Badecker 1999, 2000; Dell 1986; Rapp & Goldrick 2000, 2004, *inter alia*). Processing that occurs at one level can proceed before the processing at another level is complete (e.g. Bock & Levelt 1994, Ferreira 2000, Ferreira & Swets 2002, Ferreira & Slevc 2007, Kempen & Hoenkamp 1987, Levelt 1989, Smith & Wheeldon 1999). For example, phonological encoding can begin before a sentence has been fully syntactically processed. These different levels of processing not only proceed

independently but also have independent effects as demonstrated by various speech errors (e.g. Dell 1986) and the independence of lexical and structural priming (e.g. Bock & Loebell 1990, Pickering & Branigan 1998). In this dissertation, I focus strictly on processing at the syntactic level and how type of processing affects the reuse of linguistic forms.

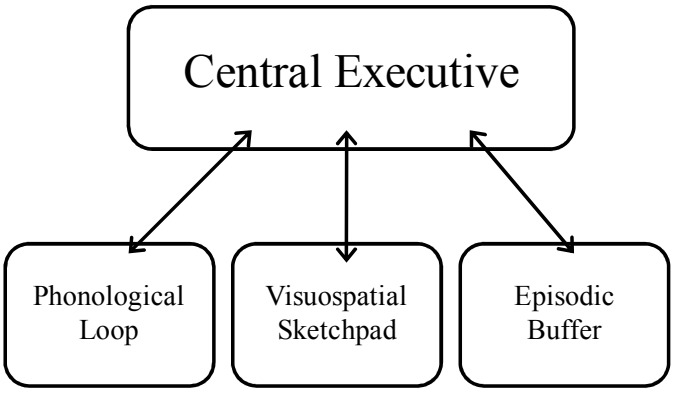
In addition to assuming separate levels of processing, activation-based models also assume that retrieved linguistic forms begin to deactivate soon after the processor has used them. This deactivation, or DECAY, of the memory trace occurs at a constant function leading to an exponential rate of decrease. As a form's activation boost decays and its weight moves back towards its initial state, its influence on subsequent linguistic performance decreases. Activation-based models use this pattern of activation, spread, and decay to explain linguistic behavior, such as priming. Because activation and decay are sensitive to time, priming behavior in such models is also sensitive to time, such that recently occurring forms are likely to exert more influence on behavior than other, comparable forms (e.g. Plaut & Gonnerman 2000; Seidenberg & McClelland 1989; Spivey 2007). Even models that focus on the long-term effects of priming (e.g. Chang 2008; Chang, Dell, & Bock 2006; Chang, Dell, Bock, & Griffin 2000) would allow for the most recent processing event to exert additional influence on subsequent performance (Kaschak 2007).

Although activation is key for the model of language processing I am adopting in this dissertation, other components of the model also play a role in explaining priming behavior. I now turn to these other components and their role in priming behavior.

3.2 Buffers

A major component of the language processing model that I am adopting in this dissertation is the BUFFER SYSTEM, which serves as the interface between LTM and WM. For processing to occur, WM and LTM need to interact. WM needs to be able to access and manipulate the content of LTM. The content of LTM needs to be able to acquire new memories and update the representations of old memories. The buffer system is where the interaction between LTM and WM takes place. Baddeley and Hitch (1974) proposed a model of memory consisting of a CENTRAL EXECUTIVE, which manages the flow of information, and subsidiary SLAVE SYSTEMS that hold the information for the central executive to exploit, as shown below in Figure 2.4.

Figure 2.4: Baddeley & Hitch (1974) model of working memory



These slave systems are temporary stores for particular types of information and act as short-term buffers for various types of information (Baddeley 2000):

- PHONOLOGICAL LOOP: stores phonological information such as the linear order of sounds in a speech stream

- VISUOSPATIAL SKETCHPAD: stores visual and spatial information such as shapes and their location in the environment
- EPISODIC BUFFER: stores episodic information such as the temporal ordering of events

These buffers take input from the environment and hold it long enough for the central executive to integrate and manipulate it.

Baddeley and Hitch's concept of buffers has been integrated into several different models of memory. For example, Anderson (1993) uses a similar system of buffers in his ACTIVE CONTROL THOUGHT-RATIONAL (ACT-R) model, a general model of human cognition that accounts for behaviors ranging from language processing to arithmetic (Anderson 1993, 2005; Anderson, Bothell, Byrne, Douglass, Lebiere, & Qin 2004; Anderson, Budiu, & Reder 2001; Anderson & Lebiere 1998; Anderson, Reder, & Lebiere 1996; Anderson, Lebiere, Lovett, & Reder 1998; Anderson & Schuun 2000; Lebiere & Anderson 1998; Lewis & Vasishth 2005; Lewis, Vasishth, & Van Dyke 2006; Matessa 2001; Matessa & Anderson 2000; Reitter 2008). In ACT-R models, cognitive processes, such as language processing, are treated as "a series of skilled associative memory retrievals modulated by similarity-based interference and fluctuating activation" (Lewis & Vasishth 2005, p 375). In other words, all cognitive behaviors depend on the ability to retrieve and combine memories based on previous experience in similar situations. This ability is modulated by interference from similar memories and the waxing and waning of activation (as discussed in the previous section and in section 3.3.2 to come). ACT-R, like the aforementioned models of language processing, assumes that processing is affected by the activation of forms and their interactions with the activations of other forms. In one of the

earliest versions of ACT-R, Anderson and Pirolli (1984) demonstrate its spreading activation architecture, and this architecture persist through subsequent versions of ACT-R as demonstrated by the equations used to estimate forms' activation (section 3.3.2). According to an ACT-R type model, if the processor is generating a sentence, it needs to retrieve specific words and combine them into grammatical patterns that conform to previously encountered grammatical patterns. The processor uses buffers as temporary stores for this information.

I adopt the following buffers from ACT-R (Anderson 1993):

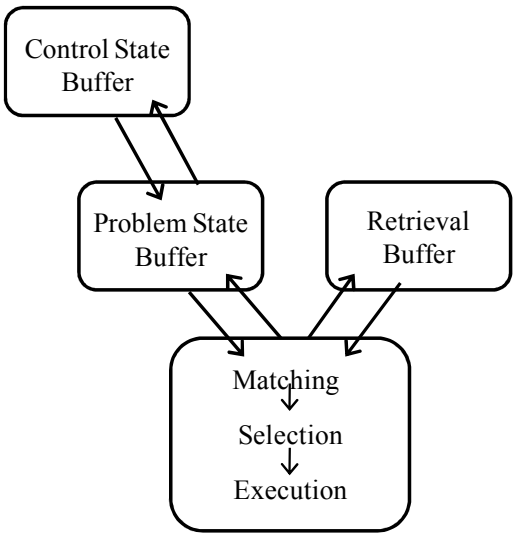
(i) the CONTROL STATE BUFFER, which tracks the overall goal of the task (e.g. 'process sentence');

(ii) the current goal or PROBLEM STATE BUFFER, which holds the particular subgoal being addressed (e.g. 'process determiner phrase (DP)' or 'process verb phrase (VP)'); and

(iii) the RETRIEVAL BUFFER, which refers to the buffer that holds whatever particular word is involved with the current goal state. This buffer could also be empty depending on whether a word has been retrieved recently or not.

In my model, I exploit these buffers to demonstrate the interactions among linguistic forms and the goals of language processing. Figure 2.5 presents the buffer system that I adopt in this dissertation.

Figure 2.5: Buffer system



In my model along with Anderson's (1993) ACT-R model, the control state buffer is connected to the problem state buffer. These two buffers feed information to one another. The retrieval buffer holds any recently retrieved declarative chunk (as defined below). Both the retrieval buffer and the problem state buffer are connected to a work area (i.e. the box that contains "Matching→ Selection→ Execution") that evaluates the needs of the current processing state based on the contents of the problem state buffer and the retrieval buffer. This work area allows the processor to determine how best to resolve the current problem through a process of matching, selecting, and executing production rules (section 3.3.2 below). To understand how linguistic information in LTM interacts with the buffer system, we first need to understand how this information is represented in memory.

3.3 Representations of knowledge in memory

Two types of knowledge are commonly distinguished: DECLARATIVE and PROCEDURAL

knowledge.[1] DECLARATIVE KNOWLEDGE contains the facts we know, such as the meaning of the word *sage.* These memories can be learned explicitly and are represented as DECLARATIVE CHUNKS, sets of feature-value pairs. For example, a declarative chunk corresponding to the word *sage* may have the following representation:

***Sage* Chunk**

$$\begin{pmatrix} \text{isa : noun} \\ \text{orth : } \textit{sage} \\ \text{semantic : spice} \end{pmatrix}$$

This chunk contains the feature-value pair of 'isa : noun' (meaning "is a noun").The chunk also contains the feature-value pair 'semantic : spice.' This feature-value pair provides the conceptual information, e.g. a word's semantic classification. There may be other semantic features that are also represented in a chunk's feature-value pairs.

PROCEDURAL KNOWLEDGE, on the other hand, is the knowledge necessary for performing actions, such as linguistic structure building (e.g. building a determiner phrase) or riding a bicycle. In ACT-R-inspired models, this knowledge is represented as a series of PRODUCTION RULES. These production rules take the form of condition-action units or IF-THEN statements that specify goals, change buffer states, and often lead to the creation of new subgoals. To illustrate, a production rule for retrieving a DP might have the following form:

IF the goal is to build a DP and no determiner has been selected

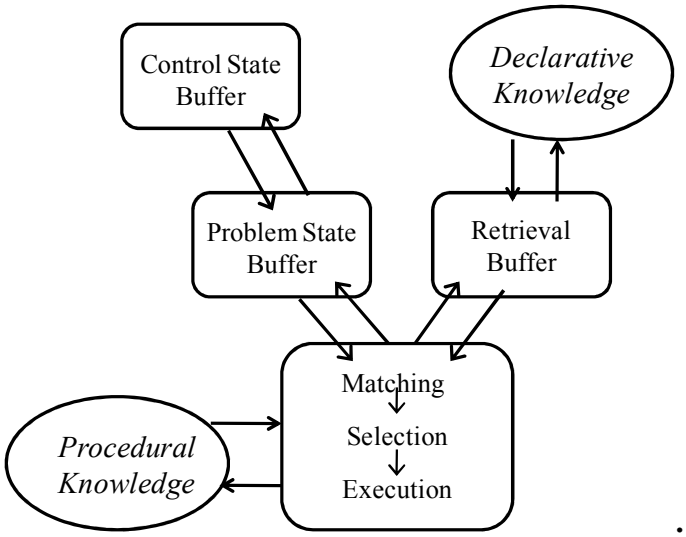
THEN retrieve a determiner phrase from LTM.

Both declarative knowledge and procedural knowledge feed into the buffer system

[1] For the current discussion, I again borrow from Anderson's ACT-R model (1993, 2005), as it is useful in subsequent discussions about language processing.

presented in Figure 2.5, as amended below in Figure 2.6

Figure 2.6: Revised buffer system

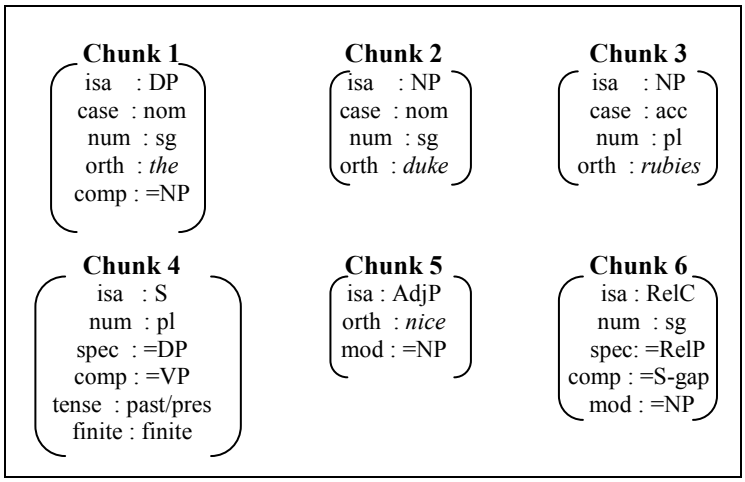


One good reason for keeping the distinction between declarative and procedural knowledge clear is that there is neuropsychological evidence for a distinction between declarative knowledge and procedural knowledge (Ogawa, Inui, & Ohba 2008; Ullman 2001, 2004; Ullman, Corkin, Coppola, Hickok, Growdon, Koroshetz, & Pinker 1997; Stowe, Broere, Paans, Wijers, Mulder, Vaalburg, & Zwarts 1998).

3.3.1 A closer look at chunks and rules

Declarative chunks: As mentioned above, the representations of declarative memories are called *chunks.* These chunks are sets of feature-value pairs. The feature-value pairs express information such as whether the form takes an argument and of what type or any other features necessary for case and agreement (Lewis & Vasishth 2005). For example, consider the following

six chunks.[1]



These six chunks are sets of feature-value pairs. The feature-value pairs state the type of chunk ('isa' meaning "is a") according to its syntactic category (e.g. 'isa : S' means "is an S(entence)-chunk") along with features relevant for agreement (e.g. 'num : sg' means "number is singular").[2] Some of the above chunks also have open (unresolved) values, which act as place holders for values yet undefined. For example, in the S-chunk (Chunk 4), there are two open values: 'spec : =DP' and 'comp : =VP.' Although these values are open, meaning that the specific content is not yet determined, there is information stating what the syntactic category of the complement should be (e.g. 'VP'). The '=' denotes that the value is unspecified, and the XP following the '=' denotes the type of chunk whose values would satisfy the open value. Chunks 5

[1] All the additional chunks used in this and later chapters in the dissertation are in Appendix 2A both in their long forms and their abbreviated forms.

[2] I adopt the use of features such as 'comp' and 'spec' from the work of Lewis & Vasishth (2005). Likewise, the use of agreement values, such as the feature-value pair in the S-chunk is inspired by Lewis & Vasishth (2005).

and 6 contain the feature 'mod,' which refers to the type of element they modify.[1] These two chunks must modify nouns. However, other chunks (e.g. 'isa : AdvP') can modify VPs, Ss, or ADJs. Note that not all chunks have open values. Some chunks, e.g. the Chunks 2 and 3, have all the values of their feature-value pairs filled. I return to the importance of having filled (specified) or unfilled (unspecified) values in section 3.4 below.

According to ACT-R, there are two ways to add new chunks to long-term memory. They can either be learned explicitly or through the successful resolution of a problem. The first option is akin to memorizing multiplication tables or the names of children. The second is akin to solving an arithmetic problem or comprehending a novel sentence. In the first case, the chunk is learned as a whole (e.g. memorizing the fact "9x3=27") without having to make reference to the computations that lead to the solution. In the second case, the processor had to compute the solution using pre-existing templates or chunks and combining them, e.g. manipulating chunks for the numbers (i.e. 9 and 3) and the operations relevant to multiplication. For this type of processing, the processor needed to retrieve individual chunks, hold them in memory, integrate them, and reach an answer. The output of this process is then sent to LTM and becomes a declarative chunk that could be retrieved in whole during subsequent task that require the multiplication of 9 and 3.

For the purposes of the current work, I focus on the second type of learning, namely the formation of new declarative chunks through the manipulation of independent, pre-existing chunks and the operations to combine them. During the processing of a sentence, the processor

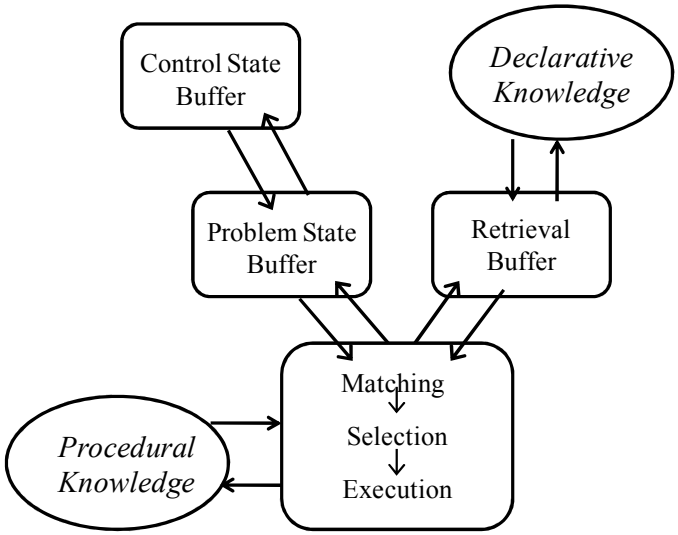
[1] This 'modification' feature is inspired by Sag, Wasow, & Bender's (2003) 'mod' feature and by Kromann's (2004) use of an 'amod' feature. In both of these approaches, the features indicate the syntactic category of the form that an adjunct can modify.

must retrieve individual words and combine them, similar to how the processor needs to retrieve individual numbers to calculate a multiplication problem. After the sentence has been processed, it can be retrieved as a unit from memory. The assumption is that the processor solves problems (e.g. 'process sentence') by constructing novel phrases and clauses. These phrases and clauses use pre-existing linguistic forms (e.g. the words *the* and *cat* are combined to generate 'the cat') to generate new forms that can be used for subsequent processing. That is, the product of this processing (the new phrases or clauses) ultimately becomes a chunk that is itself retrievable as a whole.

Production rules: Production rules, unlike declarative chunks, are not a collection of feature-value pairs. Rather, they take the form of condition-action units or IF-THEN statements that change goals ('process sentence'→ 'process subject DP'), add chunks to buffers ('retrieve DP-chunk'), and remove chunks from buffers ('pop NP-chunk') (Anderson & Schuun 2000). One can think of chunks as being the items in the buffers and of production rules as being the procedures that move the chunks through the buffers. When attempting to select a production rule in a given context, the processor first checks the buffers states against the IF parts of the possible rules. The processor calculates each rule's strength, which is roughly its activation, and utility, which is how likely it is to satisfy the current goal minus the cost of applying the rule (see also section 3.3.2 below) (Anderson & Lebiere 1998).

The use of production rules can be broken into three steps: the matching phase, the selection phase, and the execution phase as shown in Figure 2.6 repeated here:

Figure 2.6: Revised buffer system



In this figure, we see that declarative knowledge is accessed via the retrieval buffer, and procedural knowledge is accessed via a work space joined to both the problem state buffer and the retrieval buffer. The MATCHING PHASE of this process refers to the process of checking the buffers (i.e. the problem state and retrieval buffers) and comparing them to the condition parts (IF statements) of each production rule. After the processor has matched the IF-statements to the current status of the buffers, it has a set of possible production rules to choose from. Each of these rules contains IF-statements that are true of the current state.[1] During the SELECTION PHASE the processor compares the production rules in the set of production rules output by the matching phase and determines the utility of each rule. This step determines which production rules are most likely to satisfy the needs of the goal while incurring the least amount of cost (section 3.3.2). During the EXECUTION PHASE, the processor deploys one of the rules that resulted from

[1] There is always a factor of random noise that could lead to imperfect matching. Ultimately, a rule that is not valid given the contents of the buffer or that does not satisfy the goals of the processor could be selected and fired due to this noise and, hence, lead to a potential error. However, I do not consider this factor in the current dissertation.

the selection phase, amends the buffers as determined by the rule, and tracks the success of the rule in achieving (or leading to the achievement of) a goal.

For example, say the processor has checked the buffers and matched them to the following condition:

IF the goal is to build a determiner phrase and the retrieval buffer is empty

There may be multiple actions that could follow such as:

THEN retrieve a DP headed by an indefinite article

THEN retrieve a DP headed by an definite article

THEN retrieve a proper name

The processor then selects among the rules (e.g. 'if the goal is to build a subject phrase and the retrieval buffer is empty, then retrieve a definite determiner phrase' versus 'if the goal is to build a subject phrase and the retrieval buffer is empty, then retrieve an indefinite determiner phrase'). This selection process is affected by a number of factors including pragmatic pressures and the relative activation of the rules. The second of these is the primary focus of the current work and is explained in greater detail in section 3.3.2 below. Once a rule has been selected, it is executed (fired), and the processor notes whether it successfully satisfied the goal in the problem state buffer.

In what follows, I focus on three general functions of production rules that arise in one fashion or another in various ACT-R models: retrieve, push, and pop.[1]

(i) RETRIEVAL RULES find chunks in long-term memory and place them into the retrieval buffer.

[1] Different versions of ACT-R assume different numbers of rules. Still, there are a few common elements across the various ACT-R instantiations, and I refer to them as 'retrieval,' 'push,' and 'pop' rules.

(ii) PUSH RULES change the problem state by adding new subgoals.

(iii) POP RULES remove chunks from the retrieval buffer.

Let us consider, now, the actual form of three production rules: a retrieve rule, a push rule, and a pop rule.

A retrieval rule checks the state of the retrieval buffer against the problem state buffer. If the retrieval buffer is empty and there is a current problem in the problem state that needs resolution, a retrieval rule fires to retrieves a chunk that may resolve the problem. The actual selection of one chunk in lieu of another depends on the chunks' activation weights relative to one another and their overlap with the current needs of the processor (section 3.3.2). Table 2.1 contains an example of a 'retrieve' production rule.[1]

Table 2.1: Production rule for retrieving a DP-chunk ('retrieve' rule)

Syntax of Production Rule	English Description
(process-sentence=goal>	Production rule to satisfy primary goal
	IF the goal chunk is
isa : S	of the type sentence
num : sg	and the sentence requires a singular subject
spec : =DP	and it contains an open value for a DP
comp : =VP	and it contains a open value for a VP
tense : past	and the tense is past
finite : finite	and the form is finite
=retrieval>	AND IF the retrieval buffer
isa : nil	is currently empty
==>	THEN
+retrieval>	**request a retrieval**
isa : DP	**of a chunk that is of type DP**
)	

[1] The production rules for this dissertation along with their names (their 'abbreviated forms') are listed in Appendix 2 B.

The left-hand column in the table above shows the production rule syntax, and the right-hand column contains its plain-language translation. This production rule matches the problem state buffer (the =*goal*> part of the statement) and the retrieval buffer (the =*retrieval*> part of the statement). Given the current state of these buffers along with the current goal (e.g. 'process sentence') and the utility of one rule over another (section 3.3.2), the processor selects a rule, which is then executed. In the current example, the problem state goal is to process a sentence and the retrieval buffer is empty. The processor matches these states to the IF part of the possible rules and selects the most appropriate one given the conditions in the IF part of the rule, in this case a 'retrieve DP' rule. For ease of future presentation, rules such as this are shown only in their abbreviated form, which refers to the THEN part of the statement. For instance, the example in Table 2.1 is henceforth '*retrieve DP*.'

Table 2.2 contains the syntax for a push rule. Recall that push rules change the current problem state by taking chunks from the retrieval buffer and moving them to the problem state. A push rule fires when the chunk in the retrieval buffer contains one or more feature-value pairs that need to be resolved. For example, in Chunk 1 (repeated below), there is an open NP value.

Chunk 1
isa : DP
case : nom
num : sg
orth : *the*
comp : =NP

Because the value of the comp feature is unspecified, the DP-chunk is incomplete, and its completion becomes a new subgoal. The DP-chunk is pushed into the problem state buffer, and its subgoal (i.e. 'process NP') is placed on the stack of subgoals. Each new subgoal is stacked on top of the last subgoal (Anderson & Douglass 2001). They must be resolved in reverse order, the

most recent one being resolved before the subgoal prior to it. In the syntax below, the push rule takes a DP-chunk with an open value (i.e. =NP) and places it in the problem state until its open values are satisfied.

Table 2.2: Production rule for changing the probem state ('push' rule)

Syntax of Production Rule	English Description
=goal>	IF the goal chunk is
isa : S	of the type sentence
num : sg	and the sentence requires a singular subject
spec : =DP	and it contains an open value for a DP
comp : =VP	and it contains a open value for a VP
tense : past	and the tense is past
finite : finite	and the form is finite
=retrieval>	AND IF the retrieval buffer
isa : DP	currently contains an determiner phrase
orth : *the*	and contains *the* as its head
comp : =NP	and contains an open value for an NP
==>	THEN
!push!	**push the current chunk into the problem state,**
isa : DP	**making a new subgoal**
orth : *the*	
comp : =NP	

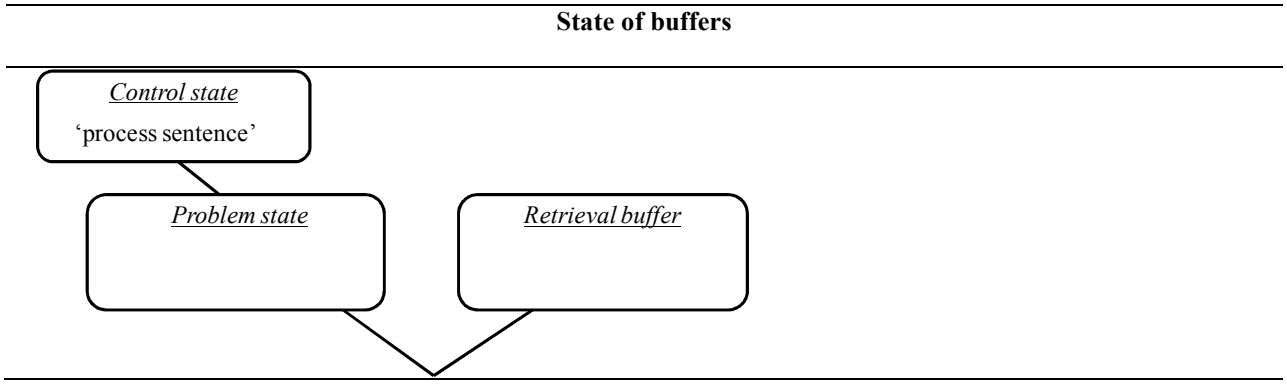
Table 2.3 contains an example of a pop rule, in particular, the rule that would pop an NP-chunk with no open values. Pop rules fire only when (i) there is a chunk in the retrieval buffer and (ii) this chunk has no open (unresolved) values. Pop rules take chunks from the retrieval buffer and send them to LTM. The particular pop-rule below removes an NP-chunk from the retrieval buffer.

Table 2.3: Production rule for popping an NP-chunk ('pop' rule)

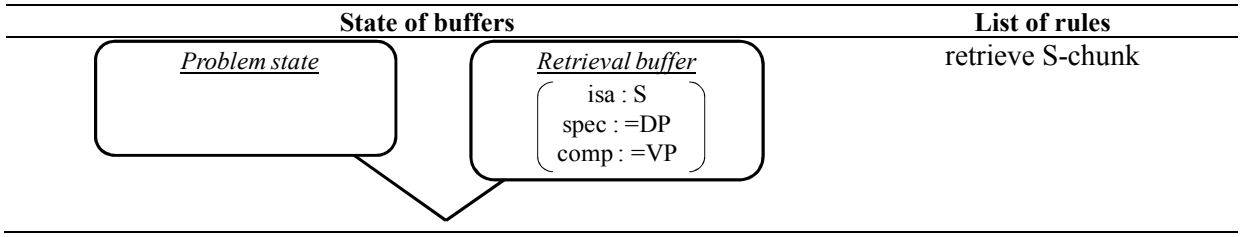
Syntax of Production Rule	English Description
=goal>	IF the goal chunk is
isa : DP	of the type determiner phrase
orth : *the*	and contains *the* as its head
comp : =NP	and contains an open value for an NP
=retrieval>	AND IF the retrieval buffer
orth : NP	currently contains an NP with no open values
head : *noun*	
==>	THEN
!pop!	**pop the contents of the retrieval buffer**
isa : NP	

Here we see that when the item in the retrieval buffer has no open values, it does not need to be sent to the problem state and can, rather, be popped.

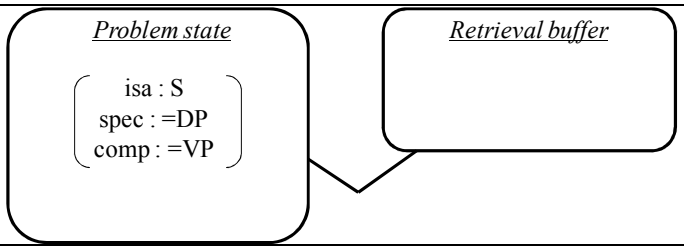
The choice among the rules is affected by the state of the buffers, and the states of the buffers are affected by the outcomes of the fired production rules. I now turn to how these production rules lead to changes in the buffers through the retrieval, pushing, and popping of chunks. Consider the diagram below. Here, we see that there is an initial goal in the control state buffer: 'process sentence.'



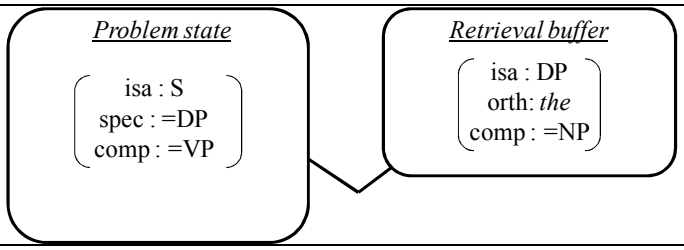
The processor checks the state of the problem state and retrieval buffers and sees that there is nothing currently in either. Using this information, the processor compares the results of the buffer tests (i.e. 'problem state and retrieval buffer = empty') to the production rules in LTM and identifies production rules that contains IF statements matching the current state. From these, it selects the rule that both matches the conditions of the buffers and is most likely to achieve the goal in the control state 'process sentence' at the least cost. Following rule selection, the processor fires the rule. In this particular case, the processor has selected and fired a 'retrieve S-chunk' rule and has placed the S-chunk into the retrieval buffer. From this point forward, we focus strictly on the contents of the problem state and retrieval buffer, so the control state buffer is no longer represented. We now keep track only of the contents of the problem state and retrieval buffers (left-hand column) and of the list of production rules used (right-hand column).



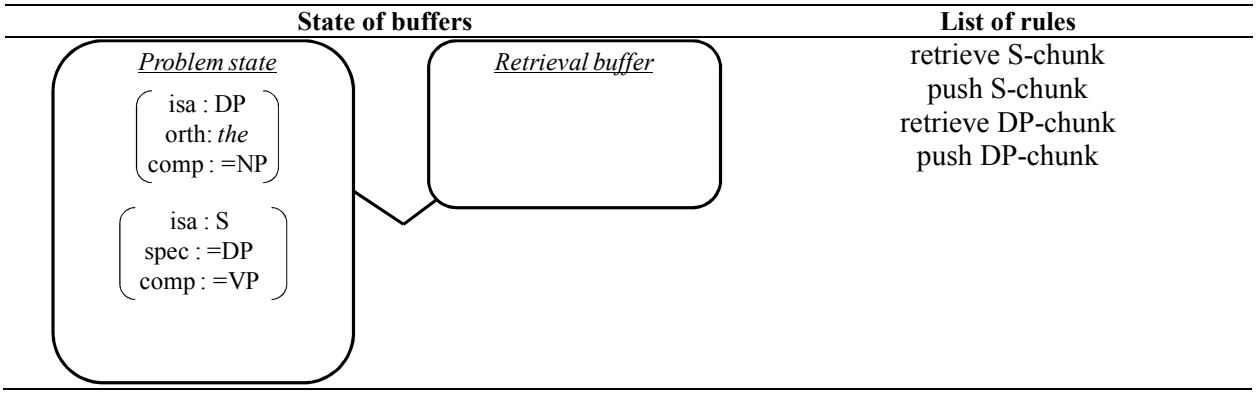
The S-chunk currently in the retrieval buffer has two open values: spec (=DP) and comp (=VP). As such, the S-chunk is incomplete and cannot be popped from the retrieval buffer. The processor again checks the two buffers, notes the S-chunks open values, and executes a rule to push the chunk into the problem state buffer, thereby adding a goal (subgoal) to the problem state.

State of buffers	List of rules
	retrieve S-chunk push S-chunk

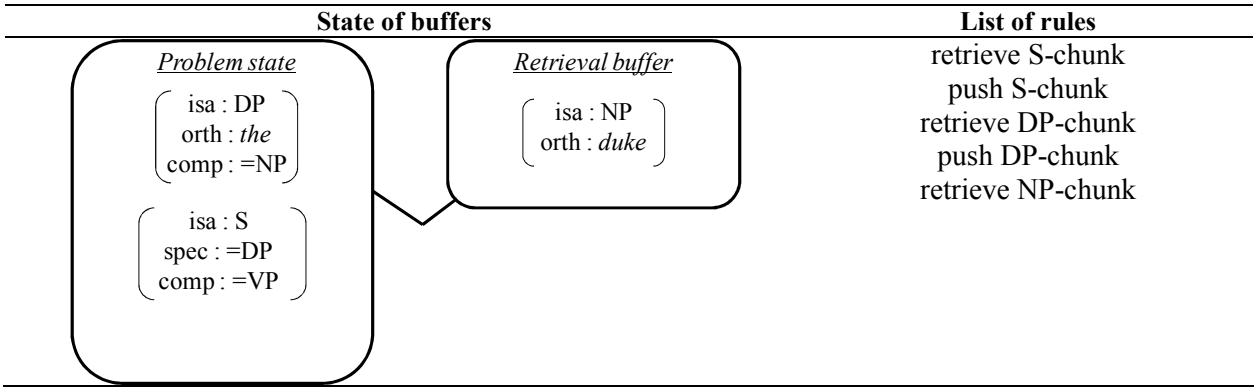
The processor checks the two buffers again and notices that the problem state has open values (subgoals) and that the retrieval buffer is empty. Using this information, the processor matches the results of the buffer checks against the possible rules, selects a rule, and executes it. In this case, a DP-chunk is added to the retrieval buffer.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk

Again, there is an open value in the chunk in the retrieval buffer, so the DP-chunk cannot be popped. The processor chooses a rule to push the chunk into the problem state buffer, adding yet another subgoal, i.e. to resolve (process) the open =NP.



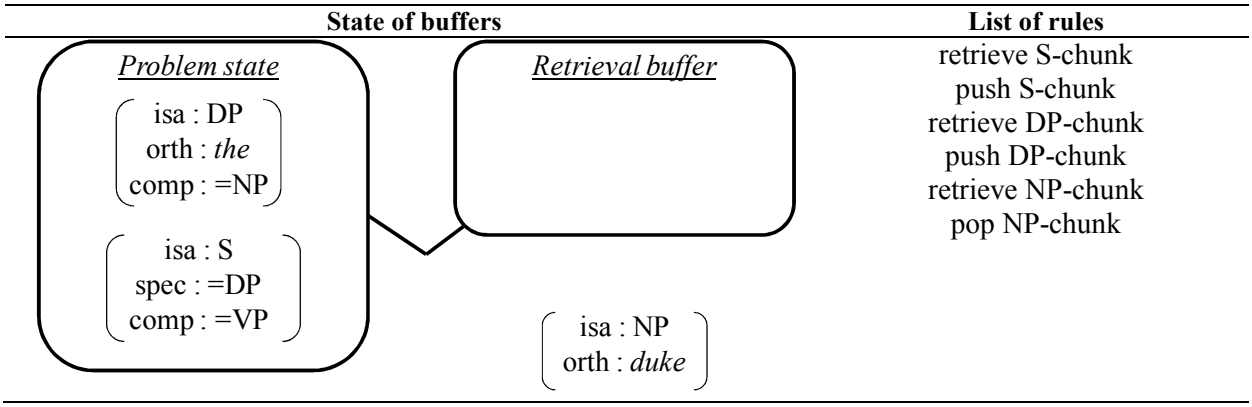
The processor notes that the current problem in the problem state is to resolve the open =NP and that there is currently nothing in the retrieval buffer, so it selects a production rule to retrieve an NP chunk. This rule fires, and an NP-chunk is selected and placed into the retrieval buffer.



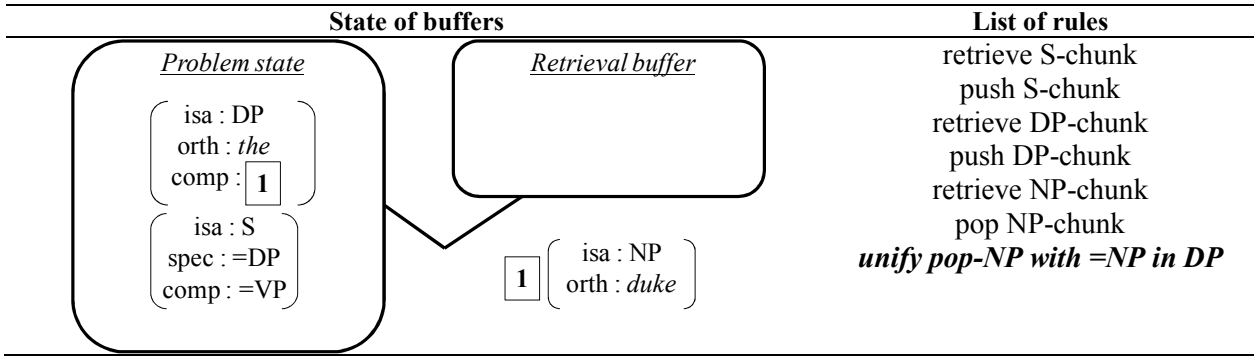
The NP-chunk has no open values, so the processor does not need to generate any new subgoals based on the features of the NP-chunk. The processor can release the chunk from the retrieval buffer using a pop rule. Pop rules can either affect the subgoal structure or not (Anderson & Lebiere 1998).[1] If the pop rule affects the subgoal structure, the values of the popped form can be

[1] Anderson & Lebiere (1994) identify six different combinations push rules, pop rules, and goal modification. The two combinations that involve popping include 'pop changed' and 'pop unchanged.' The value popped by a 'pop changed' rule can be passed from a subgoal to a parent goal through the 'subgoal return mechanism.'

passed up through the subgoal structure to the next subgoal in the stack of goals. For our purposes, this means that the values of the popped declarative chunk can be unified with the open values in the top-most chunk in the problem state. UNIFICATION is an operation that merges the feature-value bundles of one form (e.g. the NP) with the open values of another form (e.g. the =NP of the DP), as is discussed in detail in section 3.4.1. The unification of values modifies the subgoal structure by satisfying subgoals in the problem state. To illustrate, consider the popping of the NP-*duke*-chunk.



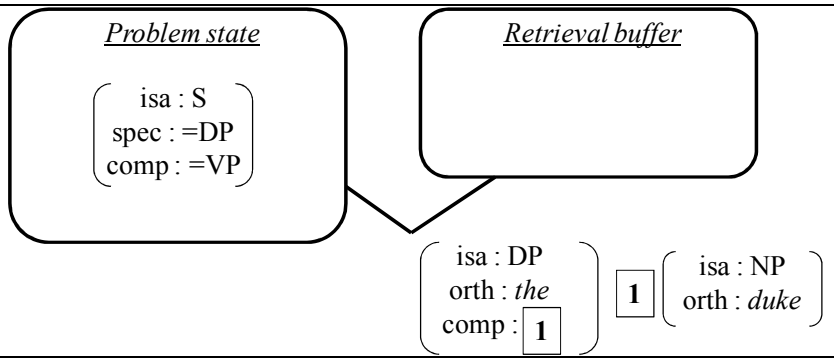
In this case, the top-most chunk (the DP chunk) stacked in the problem state buffer contains an open =NP value. This open value serves as a subgoal, i.e. 'process NP.' The popped NP-chunk can satisfy this subgoal by unifying its values with the open =NP value. This unification modifies the subgoal, as shown by the indexing below.



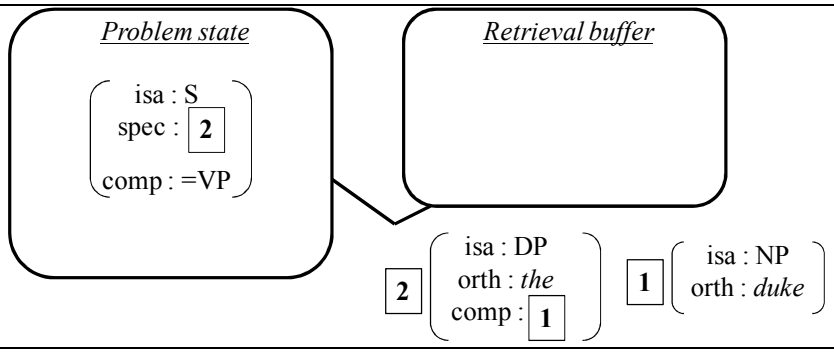
The 1 denotes the values that are now shared between the popped NP-chunk and the previously open =NP values in the DP-chunk. Specifically, the 1 indicates that the two values are in fact the same. I am using tags here in a similar fashion as unification-based approaches to grammar such as Sag, Wasow, & Bender (2003). In what follows, I show the chunks popped from the buffer system in the bottom right of each diagram to help us keep track of the various chunks that were used during the sentence's processing.

As processing proceeds, the chunk that results from one unification cycle can be incorporated into the next unification cycle. To clarify, consider the next step in the processing of our example above.

The DP-chunk has no open values, so its conditions are satisfied, and the processor can pop it.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk

The popped DP matches open values (=DP) in the S-chunk, so the pop rule can modify the subgoal 'process DP' in the S-chunk. The processor pops the DP-chunk and modifies the goal by unifying the DP-chunk's values and the open =DP value in the S-chunk. The 2 denotes that the values are now shared between the popped DP-chunk and the previously open =DP value in the S-chunk.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S***

However, there is still an open value in the S-chunk (i.e. =VP). Its conditions are not satisfied, so it cannot be popped from the retrieval buffer. The S-chunk stays in the problem state buffer. It cannot yet be removed from the stack of subgoals that must be satisfied before the sentence is fully processed.

In this manner, subgoals are added to the problem state and resolved by chunks popped from the retrieval buffer, all in pursuit of the main goal 'process sentence.' Processing is complete only when no more subgoals remain in the problem state buffer (e.g. when the 'spec : =DP' and 'comp : = VP' for the S-chunk have been resolved) and the final chunk resolves the primary goal (e.g. a complete sentences satisfies the 'process sentence' goal in the control state buffer). To summarize the various rules and chunks used in the processing of the DP in the example above, consider Table 2.4 below.

Table 2.4: Chunks and rules retrieved for processing a subject DP

Retreived chunks	List of rules
S-chunk	retrieve S-chunk
	push S-chunk
DP-*the*-chunk	retrieve DP-chunk
	push DP-chunk
NP-*duke*-chunk	retrieve NP-chunk
	pop NP-chunk
	unify pop-NP with =NP in DP
	pop DP-chunk
	unify pop-DP with =DP in S

In the left-hand column, we see the three chunks that were initially stored in long-term memory (LTM) and that were retrieved and used in the processing of the subject DP. In the right-hand column is the series of retrieve, push, and pop rules that were also retrieved from long-term memory and used. The bold-face expressions (e.g. '***unify pop-NP with =NP in DP***') denote the unifications of a popped form's values with an open value in another chunk. In subsequent chapters, I reuse this form of notation as shorthand for tracking the use of chunks and rules and the application of unification operations.

3.3.2 The retrieval and use of chunks and rules

Chunk Retrieval: According to activation-based models, such as the current one, chunks are selected based on their activation. Once a retrieval rule has been fired, the search through LTM for an appropriate chunk begins. To determine which form to retrieve, the processor compares chunks based on their activation weights using equations that estimate a chunk's BASE ACTIVATION WEIGHT and its TOTAL ACTIVATION WEIGHT. The base activation weight reflects a form's complete history of use as well as any additional boost due to recent use. The total activation weight includes the base activation weight and any additional activation a chunk receives from its relation to the current context, or its ASSOCIATIVE ACTIVATION. Henceforth, I refer to these two factors as a chunk's ACTIVATION (i.e. its base activation weight) and RELATION (i.e. the additional boost a chunk receives from the context). The following equation is used to determine the base activation weight for a chunk:

$$Bi = \ln \sum_{j=1}^{n} tj^{-d} \qquad \text{(Base) Activation weight}$$

In this equation for the base activation weight of item *i*, *n* is the total number of retrievals of *i, tj* is the amount of time *t* since its most recent retrieval *j*, and *d* is the constant function of memory decay (Anderson 1993, 1995; Lewis & Vasishth 2005). Thus, an item's activation is a summation of all of its retrievals *n* and the time since its last retrieval minus a function of decay. This activation score is the raw number associated with the chunk irrespective of the current processing. It is the weight that the chunk has simply due to the fact that it exists and that it has been processed at some point.

However, this is not the only factor that the processor uses to determine which chunk to retrieve. The relation of a particular chunk with other chunks in the context also affects the likelihood of a chunk's retrieval. For example, some chunks are likely to co-occur, so the use of one can increase the likelihood of the other's retrieval. However, the occurrence of chunks can also inhibit the retrieval of other chunks by creating interference. This interference can arise when there are many chunks associated with the same goal or processing event, leading to less activation for each chunk.

First, to capture this sense of relation and current context, we turn to the equation used to calculate a form's total activation weight.

$$A_i = B_i + \sum_j w_j s_{ji} \qquad \text{Total activation weight}$$

In this equation, we see the combination of the base activation weight *Bi* and additional factors. The first of these factors *Wj* refers to the weights associated with elements *j* of the goal, and the second factor *Sji* refers to the strength of connections between the chunk *i* and the elements *j*. *Wj* is not free but is determined by the formula *G*/*j*, where—as previously mentioned— *j* is the number of goal elements and *G* the amount of goal activation.[1] This goal activation defaults to 1. Thus, the weight associated with elements is determined by the number of elements necessary for a particular goal chunk. The more elements *j*, the less of the goal activation resources each individual element gets. This creates a 'fan effect,' meaning that the more elements the processor needs to evaluate, the slower or less-efficient the process is (Anderson 1974). When there are

[1] The assumption is that different goals have different numbers of associated features. For example, a one-column addition problem has fewer features than a three-column problem. However, the amount of WM resources remains constant over goals, so goals with more features distribute resources more thinly than those with fewer features.

only a few elements fighting for processing resources, each individual chunk's activation is higher that if there if there were many elements fighting for the resources. The total number of chunks can increase or decrease the amount of activation a chunk receives. The second key component of the total activation equation is the strength of the connections among the elements S*ji*. This factor is a measure of the weights of the connections among different forms. The purpose of S*ji* is to capture some element of contextual priming, i.e. where the current goal makes some chunks more relevant or salient.[1] This can also serve to increase or decrease the additional activation boost a chunk receives from the context.

With the addition of these factors, we can calculate the likelihood of a chunk's retrieval. Retrieval is based on both a chunk's own activation weight B_i and the amount of additional activation or interference it receives from the other elements in the context *j*. The third and final factor that affects the retrieval of a chunk is an amount of random noise (Lebiere & Anderson 1998). I do not consider this factor in any detail here.

Production Rule Retrieval: Just as the retrieval of a chunk is sensitive to its activation weight, so too is the selection of a particular production rule sensitive to the 'strength' of the rule. The term PRODUCTION RULE STRENGTH (henceforth STRENGTH) is used to describe the "probability and speed of application" of a production rule (Anderson 1993 p 52). This strength is sensitive to a rule's overall history of use, just as a chunk's activation weight is. The equation for determining a production rule's strength is given below.

[1] Although this is a relevant factor for determining activation weights, there is debate about its importance. Some contend that it is the least important aspect of activation, at least during the learning process (Anderson & Schuun 2000).

$$S_p = \ln \sum_{j=1}^{n} t_j^{-d}$$ Production strength

In this equation for the strength of a production rule *p*, *n* is the total number of uses of *p, tj* is the amount of time *t* since its most recent use *j*, and *d* is the rate of memory decay (Anderson & Schuun 2000, Lebiere 1998). This factor of decay, *d*, affects the accessibility of rules just as it affects the accessibility of chunks. For both chunks and rules, the onset of decay begins as soon as the processor is done using the particular chunk or rule. For chunks, the onset begins as soon as the chunk is popped from the retrieval buffer. For rules, the onset begins as soon as the rule has completed the THEN part of its statement. As a consequence, the activation/strength of chunks and rules is sensitive to recency of use, making them both susceptible to recency effects.

Just as we saw with the chunks, the retrieval and application of a particular rule is sensitive to the demands of the current context. A chunk's usefulness is determined by the current goal or subgoal. For chunks, I called the connection to context 'relation.' For rules, I use the term PRODUCTION RULE UTILITY, as adopted from Anderson (1993) (henceforth *utility*). Utility refers to the expected gain associated with firing a rule minus the expected cost associated with the rule and is determined by using the formula

$$\mathbf{U = PG - C}$$ (Production Rule) Utility

where *P* stands for the probability of success, *G* stands for the value of the particular goal, and *C* stands for the cost associated with implementing the rule. Determining the values of *P* and *C* depends on previous experience with the rule. *P* is estimated using the formula

$$\mathbf{P = qr/(1-(1-q)f)} \qquad \text{Probability of success}$$

where *q* is the likelihood that a rule achieves its intended effect (e.g. retrieving a NP), *r* is the likelihood that the rule leads to the completion of the larger goal (e.g. processing a sentence), and *f* is measures the decline in the probability of achieving the goal if the rule fails. C is estimated using the formula

$$\mathbf{C = a + b} \qquad \text{Associated cost}$$

where *a* is the cost associated with the rule itself and *b* is the cost associated with the rules that need to fire following the particular rule in order to achieve the larger goal. For example, if the goal is to process the subject of a sentence and a rule fires to

Both *q* and *a* are linked to the rule directly, whereas *r* and *b* must be estimated based on expected states and outcomes. To approximate the values for expected states, the processor considers the processing that has already occurred and the amount of processing that is likely to occur before the completion of the goal. For example, say that the processor is produce the subject of a sentence and that, given the current context, it has two equally as accessible rules: one that retrieves a proper name (e.g. *Andrew*) and one that retrieves a DP (e.g. *the*). If the processor uses the rule that retrieves the proper name, there are no additional steps necessary to complete the processing of the subject phrase. However, if the processor uses a rule that retrieves a DP-*the*, there are additional rules that must fire, i.e. a rule to retrieve an NP argument for the DP (e.g. DP-*the* and NP-*musician*). The processor can predict the number of rules that need to follow the selection of a particular rule based on its previous uses of a rule. For instance, the

processor can estimate that at least one more rule needs to fire following a DP-retrieval to generate a grammatical phrase based on the other DP-processing events it has previously encountered. Because of the stack-like composition of subgoals (Anderson & Douglass 2001), the processor can estimate the amount of processing prior to the current state by referring to the current subgoal structure. And because of its previous experience with similar processing events, the processor can estimate the amount of processing likely to occur after the current state. The amount of processing correlates with difficulty, and the more difficult a problem is, the less likely it is to be successfully completed. Consequently, the more processing that is necessary, the more costly the processing is.

For example, consider the processing of an equation such as 9x3. The least costly option that is also likely to resolve the goal 'compute equation' is retrieving the declarative chunk for the particular question, namely the chunk '9x3=27.' Another option that is low-cost but less likely to resolve the goal is random guessing. A third option that is more costly but also more likely to resolve the goal is to compute the equation step-by-step by retrieving each number and the method of computation individually. If each of the options has the same base level activation, the processor must rely on the estimated utility of the options to decide among them. Given these three choices, the processor is likely to choose the first, assuming the declarative chunk exists. Otherwise, the processor must choose between the other options, each of which has its pros and cons, one with low cost and low success, another with high cost and high success.

For these reason, both the number of rules necessary for processing a goal and each rule's history of success determine the likelihood of a rule's retrieval. To summarize, Anderson (1993,

p 63) states that the selection of a production rule is determined by the following factors:

a) The past history of use of various declarative chunks
b) The goal that is currently active
c) The elements in the current context
d) The complexity of the rule
e) The past frequency of use of the production rule
f) The past history of success of the production rule
g) The amount of effort put into solving the problem so far
h) The similarity between the goal state and the state resulting from applying the production rule
i) What other options for behavior are available

As is obvious given these numerous factors, the selection of production rules is sensitive to many aspects of the current context and previous experience. For our purposes, I reduce these factors to the two general factors mentioned above: strength and utility.

For both chunks and rules, there are two general factors contributing to the likelihood of retrieval: activation/strength and relation/utility. According to my use of these terms, a chunk's activation weight and a rule's strength are blind to the current context. They are simply scores based on the history of use of a form, regardless of whether the form was correctly used or successful. For example, if a word was erroneously retrieved (e.g. the speaker meant to say *cat* but instead said *hat*), the erroneously-retrieved word still receives a boost to its activation and may, therefore, end up with a slightly higher activation weight. Likewise, if a rule is selected but ultimately fails to achieve its goal, it still receives a boost in its strength simply because it was retrieved. The second set of factors, i.e. relation and utility, are more attuned to the current processing context. For example, chunks can receive additional weight from the other chunks in the context, and rules can receive additional weight from their history of success in a similar

processing event. In this way, both chunks and rules are sensitive to their history and the current context.

3.4 Unification and the processing of structural contexts

Now that we have linguistic representations (chunks and rules), a way to retrieve them (activation and relation for chunks, strength and utility for rules), and buffers to manipulate them, we need one final ingredient: a way to combine linguistic forms. I begin with a description of this operation, UNIFICATION, and then demonstrate how the outcomes of this operation reflect the processing of different structural contexts. These different structural contexts lead to different patterns of memory traces, which subsequently affect retrieval. These different patterns ultimately serve as the basis for structural context effects on structural priming.

3.4.1 Unification

We begin with the process of UNIFICATION, which is an operation by which two structures are merged to generate a new, equally specified or more specified structure (Jurafsky & Martin 2009; Shieber 1989). This new structure contains the union of all the feature-value pairs of the original structures. For successful unification, the two structures must have either complementary feature-value pairs or at least no conflicting feature-value pairs. For example, say that Structure 1 has the feature-value pair [X : a], and Structure 2 has the pair [X : a]. Structure 1 and 2 can be unified (as denoted by the ⊔) because the values of their features agree:

$$\mathbf{[X: a] \sqcup [X: a] = [X : a]}$$

This type of unification acts as a simple equality check that takes two feature-value pairs and returns the same feature-value pair. In other words, the processor checks each structure,

determines if they contain the same feature, and then determines if they have the same value. If the structures have the same features with the same values, then the two structures are equal and can be unified. Conversely, if the two structures have different values for the same feature, they cannot be unified. For instance, if Structure 1 has the feature-value pair [X : a] and Structure 2 has the pair [X : b], the unification fails:

$$\mathbf{[X: a] \sqcup [X: b] = } \textit{\textbf{fail}}$$

The union of these two structures fails because each structure contains a different value for the same feature.

Unification operations can also unify two structures that do not have overlapping feature-value pairs. For example, say that Structure 1 has the feature-value pair [X : a], and Structure 2 has the pair [Y : b]. Structure 1 and 2 can be unified:

$$\mathbf{[X: a] \sqcup [Y: b] = } \begin{pmatrix} \mathbf{X : a} \\ \mathbf{Y: b} \end{pmatrix}$$

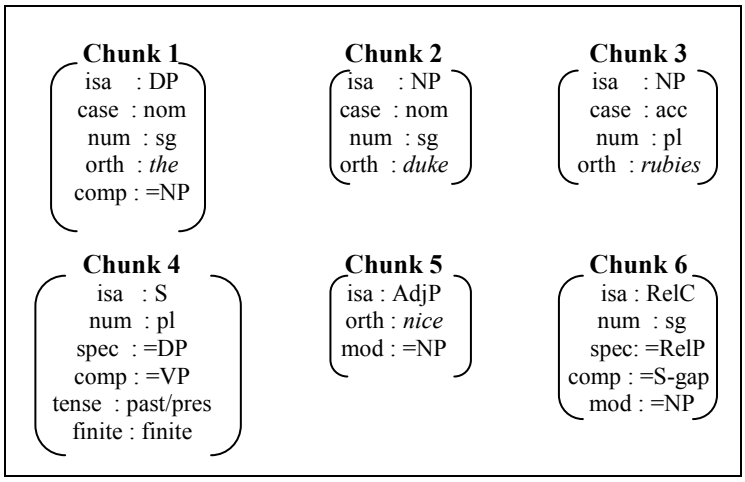
The reason these two structures can be unified is that they do not conflict. Each has different features with different values. Similarly, structures that have different levels of specificity can be unified. Take Structure 1 and Structure 2 below. Here we see that both have a feature for X, but Structure 2 leaves its value open, as denoted by the "[ ]."

$$\mathbf{[X: a] \sqcup [X: [\ ]] = [X : a]}$$

This open [ ] value is identical to the "=" notation in the discussion of chunks above. Here, the unification leads to a form in which the features are matched and the values are shared. The

resulting structure contains the feature that occurred in both structures (i.e. 'X') and the value that occurred in only one (i.e. 'a').

To make this clearer, consider again the chunks presented in section 3.3.1 above:



Let's say that the processor has retrieved Chunk 1 and chunk 2, which I abbreviate below:

isa: DP orth: *the* comp: =NP	isa: NP orth: *duke*

The DP-chunk has an open value (=NP) that is looking for something of the type NP. The processor has an active NP chunk. Because this chunk's 'isa' type (NP) matches the open value's type (=NP), the NP-chunk and open =NP value can be unified.

isa: DP orth: *the* comp: [1]	[1] isa: NP orth: *duke*

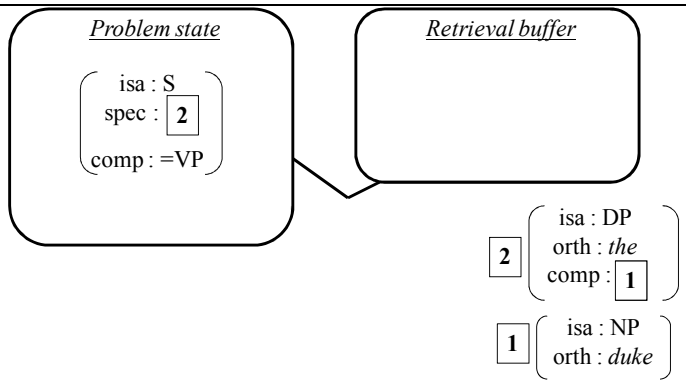
The second pair above denotes the post-unification representation. The indexing ([1]) indicates that the feature-value bundle (i.e. the NP-chunk's features and values) now serve as the feature-

values for the DP's comp (previously =NP). For the remainder of the dissertation, I describe this type of unification as the NP-chunk unifying with the =NP value of the DP-chunk's comp feature.

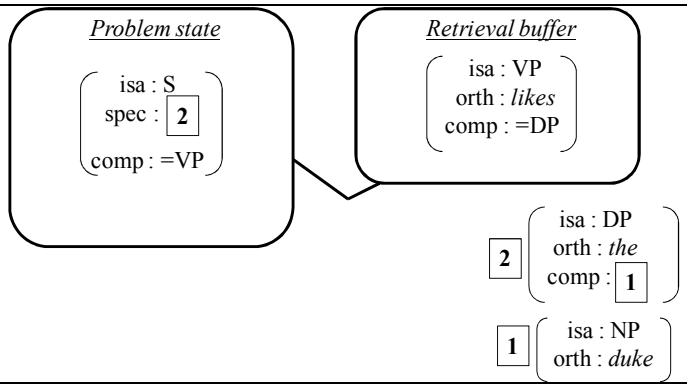
The unification of two forms occurs whenever the processor has a recently retrieved chunk and a currently active chunk whose feature-value pairs do not clash. For instance, in the example above, the recently retrieved NP chunk's values do not conflict with the currently active DP chunk's open =NP value, the NP-chunk can unify with the open =NP value. The unification operation, such as the NP and =NP unification above, applies throughout the processing of a sentence. For our purposes, I refer to each successful application of the unification operation as a UNIFICATION CYCLE. Every 'cycle' marks a step towards completing a goal or subgoal, such as 'process sentence' or 'process DP.' To complete this goal, the processor needs to move through many cycles—cycles that build NPs, DPs, VPs, etc. These cycles help us to count the amount of processing between any two points during the processing of a sentence. For instance, to process a DP-subject, there is one cycle that unifies the NP-chunk and the open =NP value of the DP-chunk and another cycle that unifies the DP-chunk with the open =DP value of the S-chunk. Thus, there are two cycles in the formation of a DP subject.

When the product of one unification cycle can be input for another unification cycle (as in the production of the unification of the NP-chunk and the open =NP value of the DP-chunk leading to the unification of the DP-chunk and the open =DP value of the S-chunk), the two cycles are linked, forming a chain. These UNIFICATION CHAINS include all the unification cycles that occur during the resolution of a single goal. Returning to our subject DP example above, we

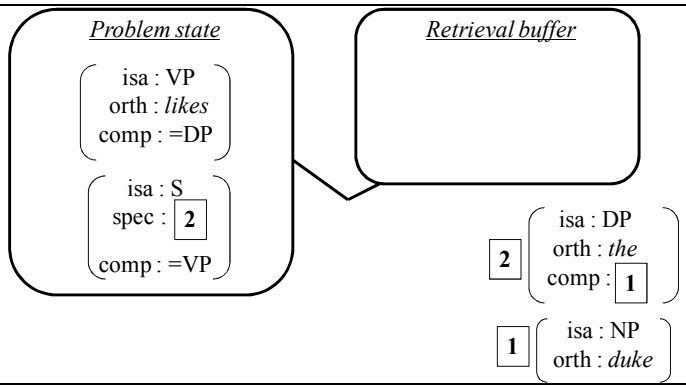
would say that there are two unification cycles in the unification chain for the completion of the 'process DP' subgoal of the S-chunk. Furthermore, all of the unification cycles that are involved in the processing of a subject DP and a predicate VP work to satisfy subgoals of the same goal ('process sentence'). Because these two subgoals (i.e. 'process DP' and 'process VP') stem from the same chunk (i.e. the S-chunk), they are part of the same subgoal structure as linked by the S-chunk. As such, the cycles necessary for the processing of the S-chunk's DP and its VP are part of the same unification chain. To illustrate, let us continue the processing of the S-chunk from section 3.3.1. I stopped the demonstration after the processing of the subject DP, as shown below.

State of buffers	**List of rules**
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S***

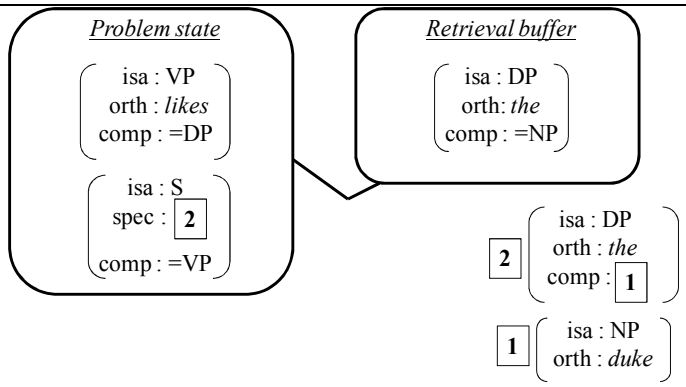
At this point, the subgoal 'process VP' becomes active. The processor chooses to retrieve the VP-*like*-chunk and places it in the buffer.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk

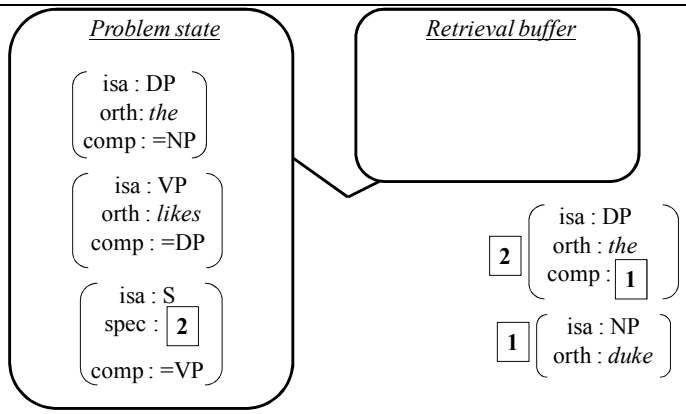
The VP-chunk has an open value, i.e. the open =DP chunk. A push rule fires and moves the VP-chunk to the problem state buffer so that its subgoal 'process DP' can be satisfied.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk

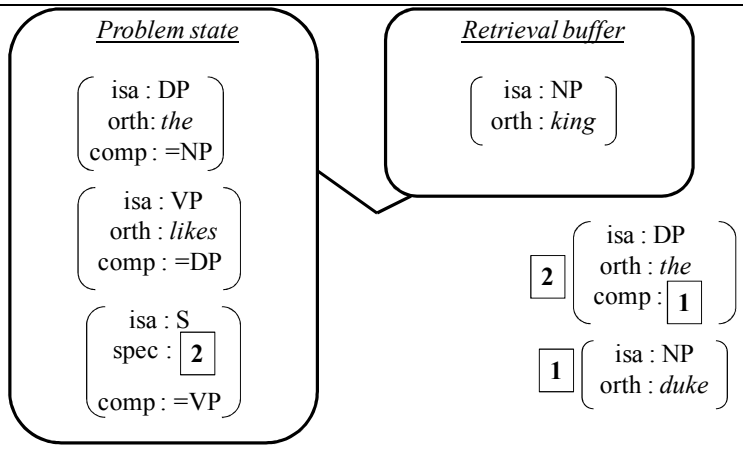
The processor notes the subgoal and the empty retrieval buffer and selects a retrieve-DP rule. The DP-chunk is placed in the retrieval buffer.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk

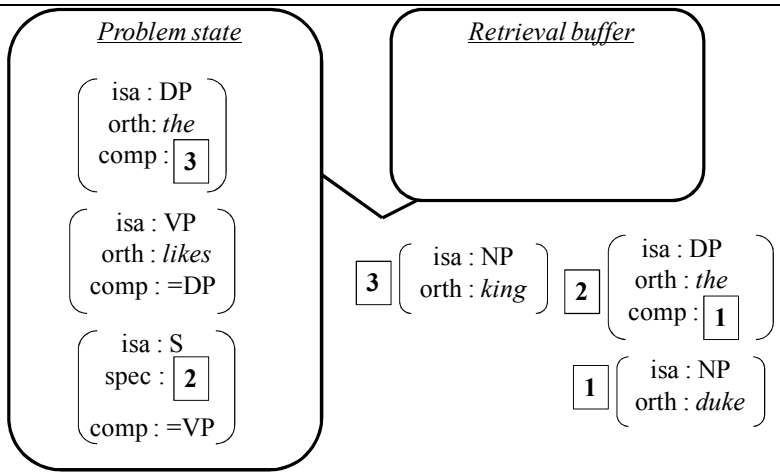
Another rule fires to move the chunk into the problem state due to its open =NP value.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk

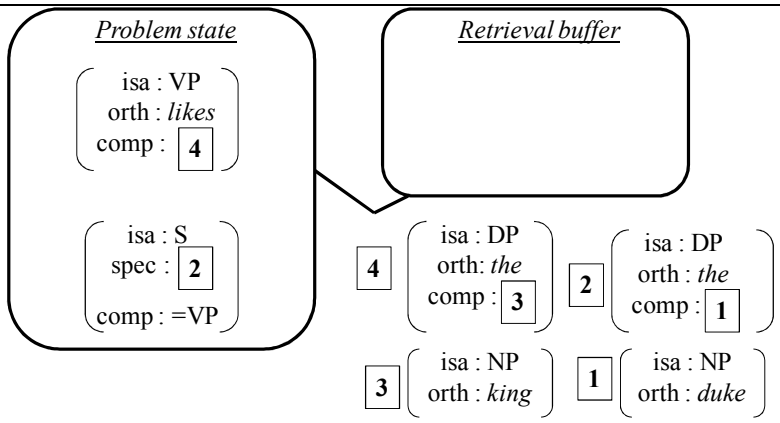
Then the processor fires a retrieve-NP chunk and places the NP-*king*-chunk into the retrieval buffer.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk

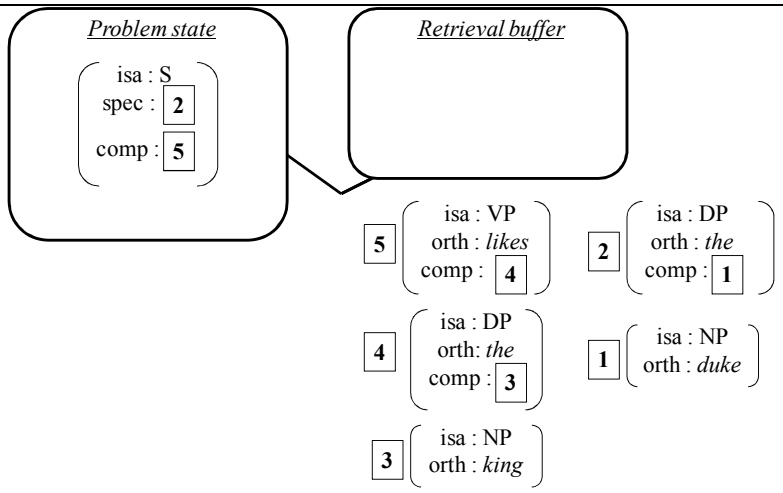
This chunk has no open values, so it is popped. It can unify with the open =NP value in the DP-chunk, so they are unified, thereby modifying the goal state.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP***

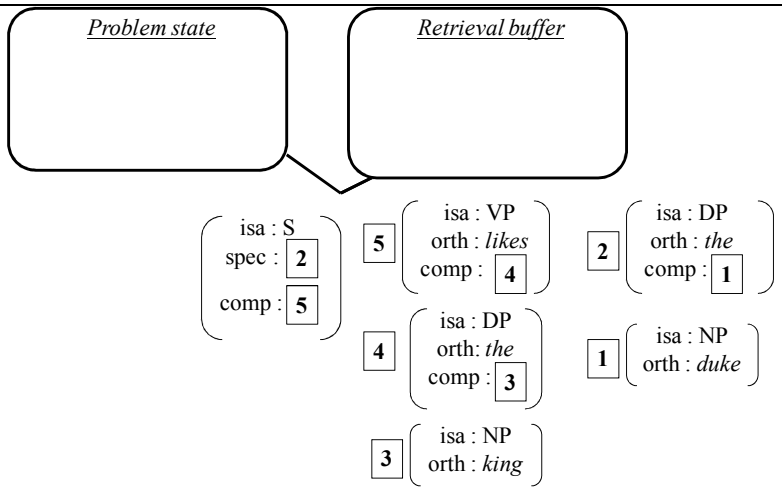
The DP-chunk is complete, so it is popped and then unified with the open =DP value in the VP-chunk.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in VP***

This unification satisfies the subgoals of the VP-chunk, so it can be popped. This allows for the VP-chunk values to be unified with the open =VP values in the S-chunk.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in VP*** pop VP-chunk ***unify pop-DP with =VP in S***

Now, all of the S-chunk's subgoals are satisfied, and the S-chunk can be popped

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S*** retrieve VP-chunk push VP-chunk retrieve DP-chunk push DP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in VP*** pop VP-chunk ***unify pop-DP with =VP in S*** pop S-chunk

Note that the indices associated with the S-chunk are 2 and 5. Each of these indexes satisfies one of the S-chunk's subgoals. During subsequent retrievals of the S-chunk, the processor retrieves the entire chain of unifications that built the 2 and the 5. In this way, the chains are unified under the S-chunk.

3.4.2 Arguments and adjuncts

As the processor retrieves and unifies chunks, it generates different sentences. These sentences can contain different numbers and patterns of arguments and adjuncts. The distinction between arguments and adjuncts is important to the model I am adopting because arguments are selected by lexical items (e.g. certain verbs such as *tell* may require one or more post-verbal arguments), and adjuncts are not. For example, the complement clause "that the man lied" is an argument of the noun *fact* in (7), whereas the relative clause "that the man told her" is an adjunct modifying *fact* in (8).

(7) Amanda knew the fact that the man lied.
(8) Amanda knew the fact that the man told her.

A lexical item's feature-value pairs contain information about whether arguments are necessary and, if they are, what the syntactic category of the argument must be. The information contained within the feature-values pairs of chunks ultimately affects the pattern of subgoals that arises during a sentence's processing, as is described in greater detail in section 3.4.3 below.

The processing of arguments and adjuncts and differences between them have been explored in detail in both linguistics and psycholinguistics (e.g. Ahrens 2003; Boland 2005; Boland, Tanenhaus, & Garnsey 1990; Boland, Tanenhaus, Garnsey, & Carlson 1995; Chambers, Tanenhaus, & Magnuson 2004; Chomsky 1981; Clifton, Speer, & Abney 1991; Demestre &

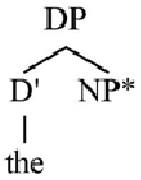
García-Albea 2004; Ferreira & Henderson 1990; Kaplan & Bresnan 1982; Kennison 2002; Shapiro, Oster, Garcia, Massey, & Thompson 1999; McElree & Griffin 1995; Tanenhaus, Spivey-Knowlton, Eberhard, & Sedivy 1995; Trueswell, Tanenhaus, & Garnsey 1994; Tutunjian & Bloand 2008; van Gompel, Pickering, & Traxler 2001).

A common finding is that arguments are processed more quickly than adjuncts, leading some to contend that arguments are primary, even when the argument is atypical (Boland 2005; Clifton, Speer, & Abney 1991; Shapiro, Oster, Garcia, Massey, & Thompson 1999). The difference between processing arguments and adjuncts may arise from frequency, in that arguments are more frequent than adjuncts based on overall use (Demestre & García-Albea 2004; Tutunjian & Boland 2008; van Gompel, Pickering, & Traxler 2001). Similarly, lexical knowledge of particular verbs may guide the initial syntactic parse to conform to each particular word's constraints, such constraints relating to its argument structure, subcategorization frames, and thematic role constraints (Boland 2005; Boland *et al.* 1990; Britt 1994; Chambers, Tanenhaus, Magnuson 2004; Tanenhaus *et al.* 1994). Context can also affect whether momentarily ambiguous phrases are parsed as arguments or adjuncts (Altman, Garnham, & Dennis 1992; van Berkum, Brown, & Hagoort 1999). For instance, Altman *et al.* (1992) presented participants sentences beginning with phrases like "The fireman told the woman that…" Their participants often parsed the *that* as marking a complement clause ("The fireman told the woman that he risked his life") although it could be marking a relative clause ("The fireman told the woman that he risked his life for to be happy"). However, this preference was sensitive to context. Readers were more likely to parse the *that* as being the head of a relative

clause if there were multiple potential referents in the context (e.g. there were two women, one who the fireman saved). The argument/adjunct distinction has also been found in production data. Arguments are more likely to be produced within the same intonational phrase as their selector, suggesting that the selector and the argument are processed as a unit (Gayraud & Martinie 2008; Watson, Breen, & Gibson 2006).

In a similar vein, syntactic frameworks such as Tree Adjoining Grammar (TAG), along with its lexically-based (LTAG) variant, argue that arguments and adjuncts are represented as structurally different within the grammar and that they are processed differently (Demberg & Keller 2008a,b, 2009; Ferreira 2000; Frank 1992, 2004; Frank & Badecker 2001; Joshi 1985; Joshi, Levy, & Takahashi 1975; Keller 2009; Kim, Srinivas, & Trueswell 2002). In TAG, the grammar consists of a collection of tree structures called ELEMENTARY TREES, such as the tree for a DP header by the determined "the" in Figure 2.7 below.

Figure 2.7: TAG tree for DP-*the*



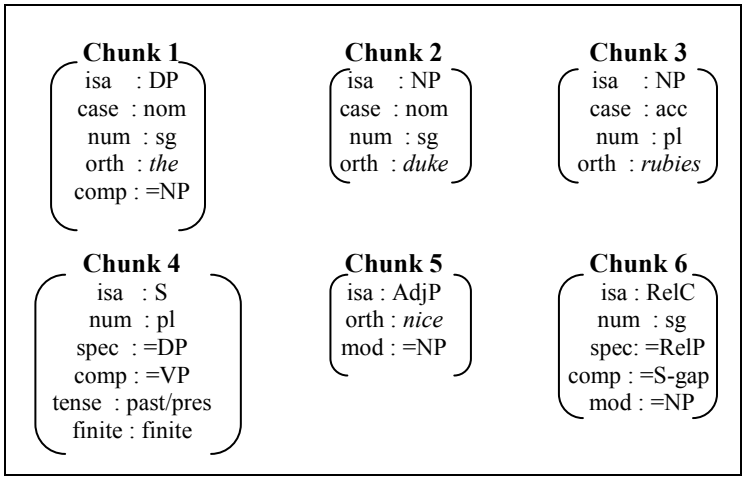
The lexical head, *the*, is present. However, the lexical head of the argument NP is not present. Only the syntactic category (i.e. NP) is noted on the tree. In TAG, dependencies between elements such as the dependency between determiners and nouns are represented in the trees in a manner similar to the open values in feature-value pairs, as discussed in section 3.3.1 above.

In sum, previous research suggests that any model of sentence processing must treat arguments and adjuncts differently. The model of language processing I adopt in this dissertation

captures the distinction between arguments and adjuncts through constraints on the interaction between declarative chunks and production rules. In the next section, I present a simplified example that illustrates how adjuncts and argument are differently processed by my model. Chapter 3 and 4 explore the differences between arguments and adjuncts in greater detail.

3.4.3 Unification of arguments and adjuncts

Unification always involves the unification of sets of feature-value pairs. For instance, a DP-chunk takes an NP argument, as denoted by the =NP in its 'comp' feature. Recall that this feature-value pair ('comp : =NP') states that the value of the feature 'comp' must be something of the type NP (i.e. 'isa : NP'). On the other hand, adjuncts are, by definition, not selected by any other element. Adjuncts place selectional restrictions on the forms with which they can unify, but they themselves are never required by another chunk. As a consequence, adjuncts are not syntactically restricted by the elements that they unify with. Consider again the declarative chunks presented in section 3.3.1.

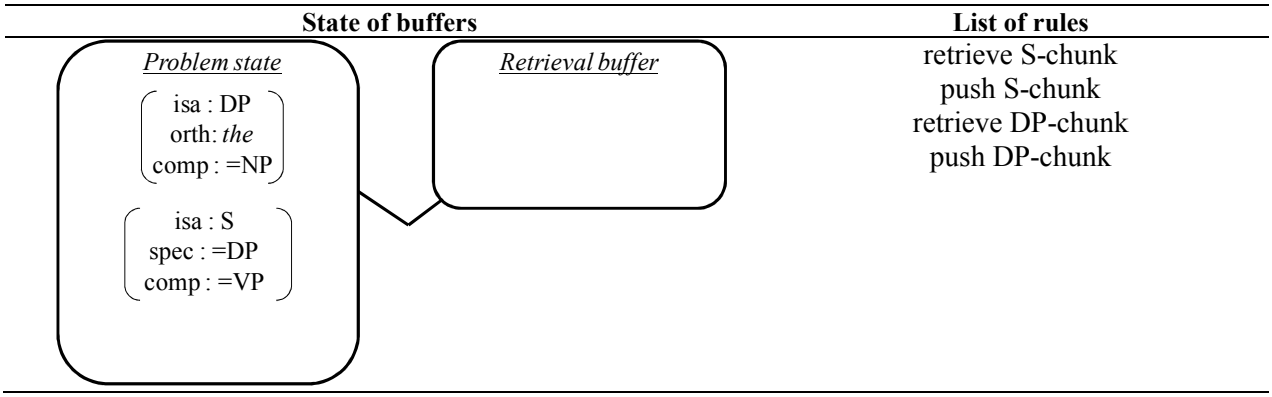


Chunks 1, 4, 5, and 6 all have open values for their complement (comp) feature. These open

values state what arguments are required by the chunk. Note that Chunks 5 and 6 have a 'mod' ('modification') feature.[1] This feature states the type of phrase the chunk modifies. However, unlike the other chunks, Chunks 5 and 6 (i.e. AdjP-chunk and RelC-chunk) are never required by any other chunk. No NP-chunk, for instance, requires an AdjP-chunk or RelC-chunk in the same way that a DP-chunk requires an NP-chunk.

Recall that when a chunk has no open values, it is popped from the retrieval buffer. At this point, it can go directly to long-term memory (LTM) without further processing or it can become available for unification with the next chunk in the problem state. When the next chunk in the problem state has open values that the popped chunk can resolve, the values of the popped chunk and the open value in the currently active chunk unify (see section 3.3.1 for a demonstration of the unification of popped chunks' values and open values in currently active chunks). However, if the popped chunk's feature-value pairs do not match any open values in the currently active chunk, the popped chunk proceeds to LTM.

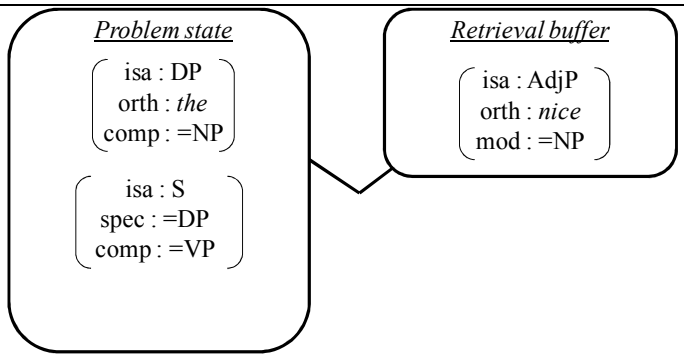
Consider the following:



[1] See Sag, Wasow, & Bender's (2003) and Kromann's (2004) for examples of uses of the 'mod' and 'amod' features.

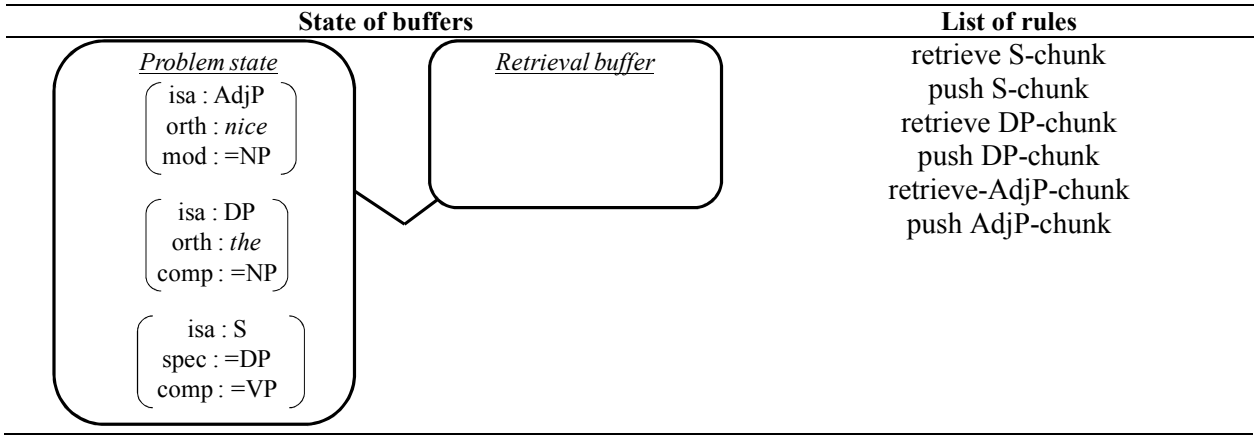
In the example presented above (section 3.3.1), after the DP-chunk was placed in the problem state buffer, the processor checked the status of the problem state buffer and the retrieval buffer, and found that the problem state had a subgoal to resolve the open =NP value (the subgoal 'process NP') and that the retrieval buffer was empty. In the previous example at this stage, the processor chose to retrieve the NP-*duke*-chunk. The utility of an NP-chunk given the current problem state is high, but this does not entail that the processor must retrieve an NP-chunk. Other pressures can lead to the retrieval of different types of chunks. For example, during comprehension, the processor may encounter an adjective. This input leads to the retrieval of an AdjP-chunk rather than an NP-chunk. Similarly, during production the processor may retrieve an AdjP-chunk due to semantic or pragmatic pressures such as pressure to identify a particular referent when there is a group of possible referents (e.g. the 'nice' duke rather than the 'tall' duke). The demonstration that follows attempts to encompass both sentence production and sentence comprehension.

Say that the processor retrieves the AdjP-*nice*-chunk rather than an NP-*duke*-chunk.

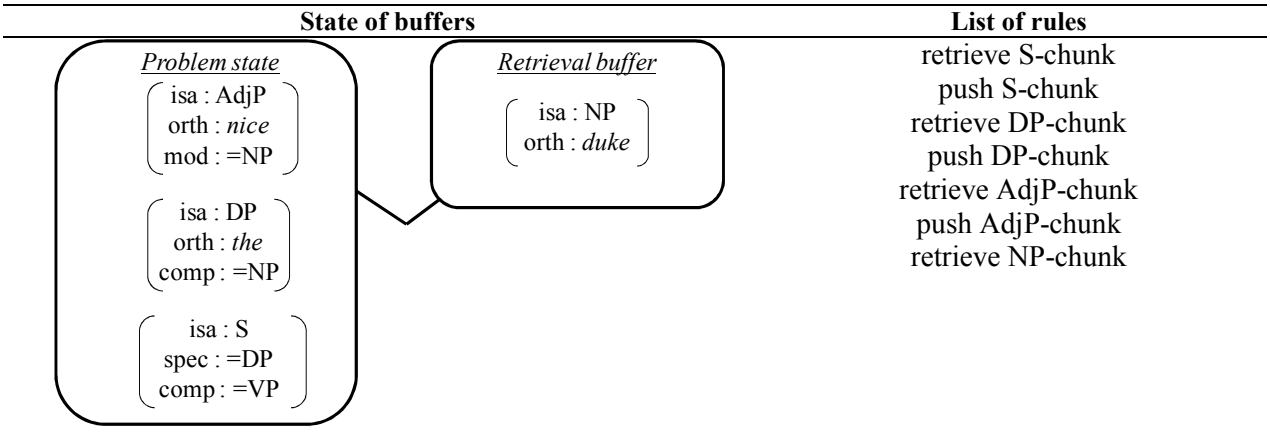
State of buffers	**List of rules**
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve-AdjP-chunk

There is an open, unresolved 'mod' value in the AdjP-chunk, i.e. =NP. The processor notes this

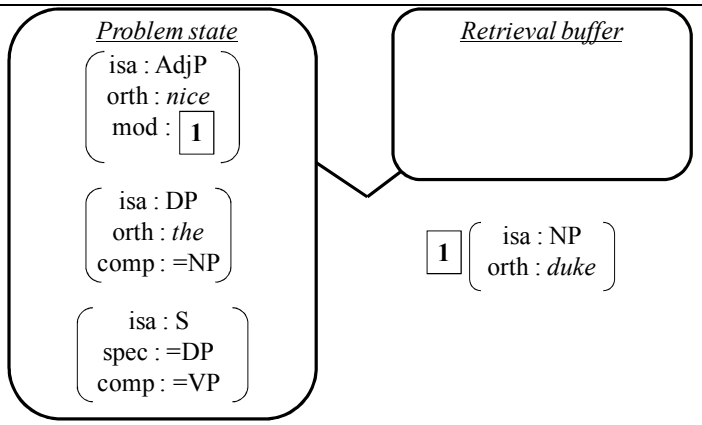
open value and fires a rule to place the AdjP chunk into the problem state until its 'mod' value is resolved.



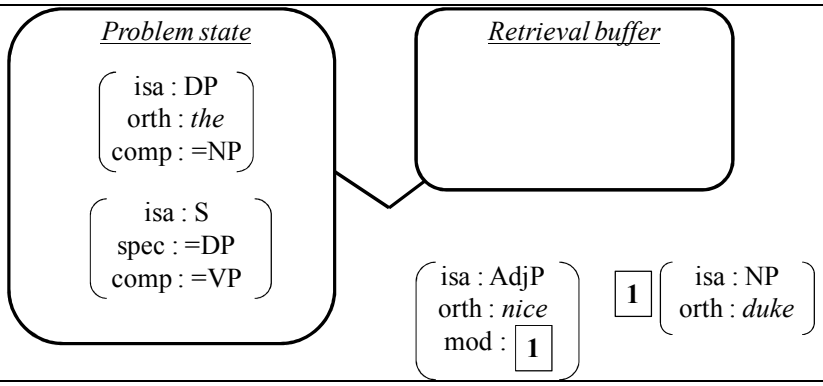
The processor checks the buffer and sees that there is a subgoal of processing an open =NP value and that the retrieval buffer is empty. The processor chooses a 'retrieve NP' rule, and places the NP in the retrieval buffer.



The NP-chunk has no open values, so it can be popped. Furthermore, because its matches open values (=NP) in the AdjP-chunk, the subgoal can be satisfied. This process is similar to the satisfaction of the DP-chunk's subgoal 'process NP,' as shown in the example in section 3.3.1 above.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve AdjP-chunk push AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in AdjP***

Now the open value in the AdjP is resolved, the subgoal 'process NP' is satisfied, and there are no more open values in the AdjP. The processor notes the state of the AdjP-chunk and selects a rule to pop it from the buffer system.

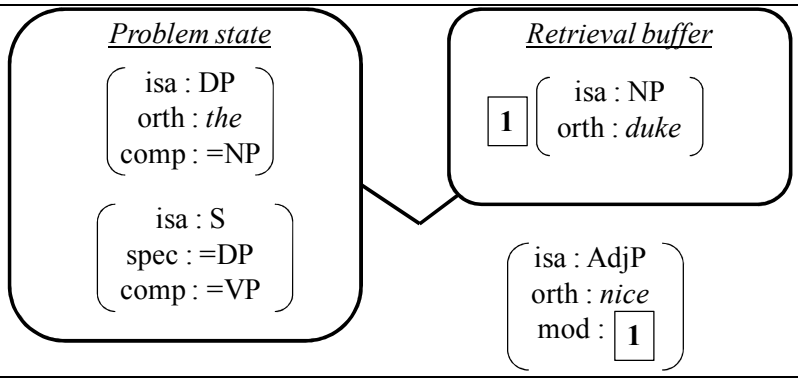
State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve-AdjP-chunk push AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in AdjP*** pop AdjP-chunk

However, unlike the previous examples, the AdjP-chunk cannot unify with the next chunk in the problem state because its values do not satisfy any unresolved, open values in the DP-chunk. Consequently, the AdjP-chunk goes to long-term memory. The processor moves to satisfy the next subgoal, i.e. the DP-chunk's 'process NP' subgoal

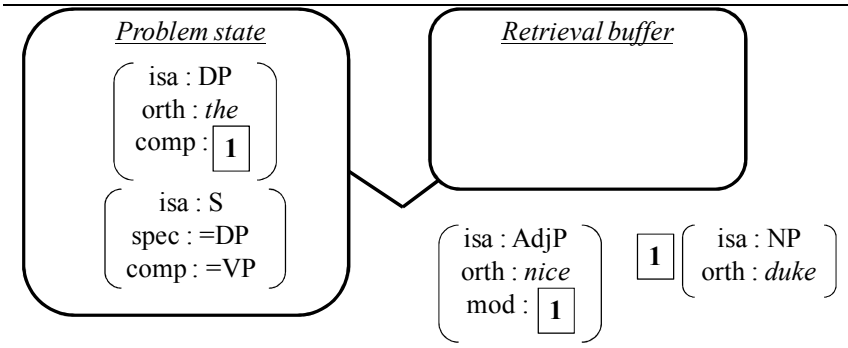
In the demonstration above, the AdjP-*nice duke*-chunk was sent to LTM. This is a consequence of the fact that the syntactic parsing of the unit was complete and that the phrase is

now a declarative chunk that can be recalled independently. This does not mean that the semantic processor cannot continue to hold the AdjP active. Semantic tracking and processing can occur independent of syntactic processing (section 3.1). Just because the syntactic module has satisfied its goals and has, hence, finished processing a particular form does not mean that other levels of processing (e.g. semantic) must also be done processing the form. My model of language processing assumes there are multiple levels of processing that are distinct—though integrated—such that they can function independently while still informing one another (e.g. Allen & Badecker 1999, 2000; Dell 1986; Roelofs 1992, 1993). I restrict my attention here to syntactic processing but leave open the possibility that linguistic forms that are no longer being processed syntactically can still be active semantically.

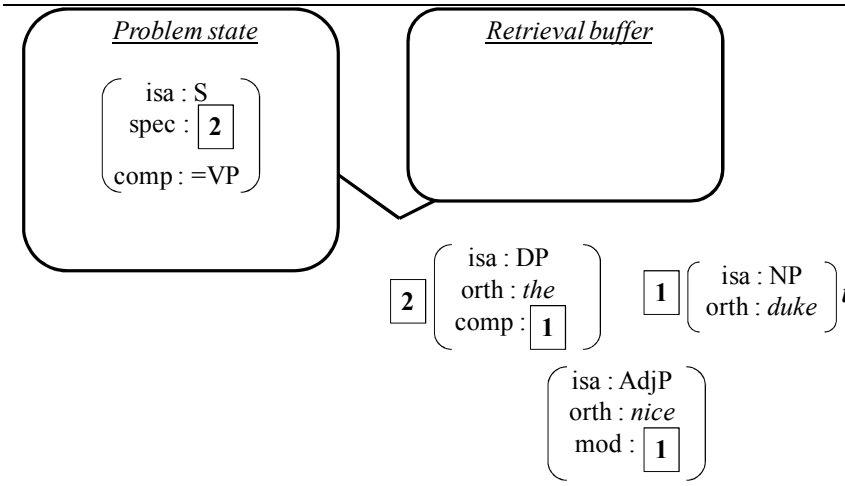
Returning to the processing of the current phrase, the processor still has a series of subgoals to satisfy, starting with the 'process NP' subgoal associated with the open =NP value in the DP-chunk. The processor selects a 'retrieve NP-chunk' rule, fires it, searches declarative memory for the most active and relevant chunk, finds the recently-used NP-*duke*-chunk, and places it into the retrieval buffer.

State of buffers	**List of rules**
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve AdjP-chunk push AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in AdjP*** pop AdjP-chunk retrieve NP-chunk

Note that the NP-*duke*-chunk still has the index that it received during its earlier unification with the AdjP-chunk. This reflects the fact that the NP-chunk's values and the previously open =NP's values are the same. Because the NP-chunk has no open values, it can be popped. Furthermore, because its values can unify with the open =NP value in the DP-chunk, the subgoal I 'process DP' is satisfied, as illustrated by the index 1.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve AdjP-chunk push AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in AdjP*** pop AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP***

The processing of the DP-chunk and the S-chunk proceed as they did in the previous example, as shown below with the popping of the DP and the unification of its values with the open =DP value in the S-chunk.

State of buffers	List of rules
	retrieve S-chunk push S-chunk retrieve DP-chunk push DP-chunk retrieve AdjP-chunk push AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in AdjP*** pop AdjP-chunk retrieve NP-chunk pop NP-chunk ***unify pop-NP with =NP in DP*** pop DP-chunk ***unify pop-DP with =DP in S***

However, because the AdjP-chunk did not unify with an open value of the DP-chunk, the DP-chunk and the AdjP-chunk do not occur in the same unification chain. Recall that each application of a unification operation counts as a unification cycle. The product of this unification cycle can participate in a subsequent unification cycle. For instance, as we saw in the example in 3.3.1, the product of the unification cycle that involved the unification of the NP-chunk and the open =NP value of the DP-chunk can then be unified with the open =DP value of the S-chunk. However, sometimes a unification cycle leads to formation of a unit that does not participate in a subsequent unification cycle, as with the case of the unification cycle that unified the NP-chunk with the open =NP value in the AdjP-chunk. In this case, the form is popped and sent directly to memory as a separate chain (Anderson & Lebiere 1998).

This is where the processing of an argument and an adjunct differ. After processing an adjunct phrase, such as the AdjP-chunk above, the unification chain associated with the adjunct ends. However, after processing an argument, as in the processing of the DP-chunk, the S-

chunk's subgoal still remains to be satisfied. As such, the unification chain does not end.

The important thing to keep in mind for what follows is that the processing of an argument isrequired by the argument's selector. Thus, an argument and its selector all occur in the same unification chain. However, an adjunct does not form part of the same unification chain as the element that the adjunct modifies.

In the example above, the AdjP-chunk and the DP-chunk do not occur in the same unification chain, as illustrated below.

Chain 1 ('process AdjP')	**Chain 2 ('process S-chunk')**
unify pop-NP with =NP in AdjP	*unify pop-NP with =NP in DP* *unify pop-DP with =DP in S*

The important thing to note here is that the processing of the AdjP-chunk and the processing of the DP-chunk led to the formation of two separate unification chains (see section 3.4.1). The processing of the AdjP-chunk, like the processing of all other adjunct chunks (e.g. RelC-chunks and AdvP-chunks), led to the formation of a separate unification chain. In contrast, because arguments are selected by elements in the problem state buffer, they are always unified with a chunk currently active in the problem state. In this way, the processing of arguments and adjuncts differs.

All the elements (e.g. chunks, rules, and unifications) that are necessary for the processing of an argument or an adjunct form a chain. After the processing of an adjunct, the chain associated with an adjunct ends because the adjunct is not required by anything in the problem state buffer. However, arguments work differently. The argument unifies with an open value in its selector. Subsequently, the unification chain associated with the argument can

continue to grow (for example, if the result of the unification is required by some element in the problem state buffer).

All the elements associated with the same unification chain are ultimately represented as one unit in memory. During subsequent retrieval when the processor attempts to retrieve a sentence, it retrieves the unification chains generated by the processing of the sentence. Ultimately, the composition of the chain (e.g. its length) affects the ability of the processor to locate specific elements within the chain.

The reason that properties of these unification chains affect access to specific elements associated with them is that they create retrieval structures and retrieval cues that facilitate the retrieval of the processing event. These retrieval structures refer to the way the retrieval cues (i.e. any stimulus that helps the processor to locate a particular memory) are organized into a stable structure (Ericsson & Kintsch 1995). The creation and composition of these structures and cues can affect subsequent access to memories. As such, the processor is concerned not only with the retrieval of linguistic forms but also the packing of individual linguistic forms for subsequent retrieval. This packing (also called 'chunking,' see Gernsbacher 1990, Kintsch 1989, Miller 1956) allows the processor to manipulate larger units (e.g. phrases or clauses rather than individual words), thereby maximizing the limited cognitive resources at its disposal.

Language processing requires the building and integrating of segments, leading to the creation of a structure to which linguistic forms (e.g. words) are adjoined (Kintsch 1988). This structure acts as a cue for subsequent retrieval (Baddeley, Hitch, & Allen 2009; Carpenter & Just 1988, 1989; Ericsson & Kintsch 1995). One way to delineate the size of these structures is to

make use of the subgoal structure. Specifically, when the popping of a chunk leads to the completion of a subgoal (as in the AdjP case where the popped chunk cannot feed directly into the next subgoal), it denotes the end of a unit's processing, creating a boundary for the retrieval structure. Thus, for our purposes, I contend that the size of these retrieval structures is determined by features of the subgoal structure. By packing information about the chunks retrieved or the rules fired into retrieval structures, the processor creates units and retrieval cues that allow for greater subsequent controlled access. Rather than having to reactivate each individual chunk or rule, the processor can retrieve larger structures and, thereby, streamline reanalysis. When the processor needs to verify that a specific chunk or production rule occurred, it can activate the retrieval structures and search them rather than having to individually reactivate every form that occurred. In this way, the processor uses unification chains to generate retrieval structures to expedite processing of a form within the chain.

Throughout the dissertation, I assume that the processor retrieves unification chains and uses these chains to, for example, verify that a particular word occurred. Because the initial processing leads to different patterns of subgoals and, hence, different types of unification chains, the processing of the structural context affects subsequent linguistic behavior.

In the following chapters, the reason for this effect becomes more clear, but as a foretaste, the unification chains affect subsequent performance by affecting the relation values for chunks and the utility values for production rules (section 3.3.2). Recall that both of these values reflect the connection between a particular chunk or rule and other chunks or rules in its context. When the processor retrieves a unification chain, it estimates the relation of a chunk to other chunks in

the unification chain. Likewise, it estimates the utility of a rule given the other rules in the chain. Depending on the number of other chunks and rules, the chunk's relation values and the rule's utility values fluctuate. I return to these arguments in Chapters 3 (for chunks) and 4 (for rules).

4. Looking back and looking forward

We began the chapter with the observations that both recency and structural context influence subsequent use of a linguistic expression, namely in the amount of priming or processing facilitation we find. The RICE hypothesis combines these two observations and claims that the way a linguistic form is processed within its larger structural context mediates the effects of recency on subsequent behavior. In the following chapters, I test this hypothesis first with lexical priming (Chapter 3) and then with structural priming (Chapter 4).

In each of these studies, I vary only the structural context in which a prime occurs and control for recency by controlling the amount of time or material between the prime and the target. For example, in the lexical priming study presented in Chapter 3, I control the number of syllables and milliseconds between the offset of the prime and the onset of the target. Prime words always occur in the same linear position (the underlined word in (9)-(12) below), but the larger structural context in (9)-(12) (bracketed) varied.

(9) Prime in matrix clause
The station received the call, and [the policeman <u>issued</u> the ticket to the poet.]

(10) Prime in the internal complement of a noun
The station reported the fact [that the policeman <u>issued</u> the ticket to the poet.]

(11) Prime in the internal complement of a verb
The station revealed [that the policeman <u>issued</u> the ticket to the poet.]

(12) Prime in relative clause
The station commended the policeman [who issued the ticket to the poet.]

In each of these sentences, the prime word *issue* occurs in roughly the same linear position, i.e. 6-8 words from the beginning of the sentence and five words from the end of the sentence. After hearing sentences such as (9)-(12), participants performed a decision task in which they determined whether the target word *issued* occurred in the sentence they just heard. If the word primed equally as well regardless of it larger structural context, there should be no systematic difference in response times based on the structural context of the prime.

Throughout both the lexical and structural priming experiments, the linear position of the prime, the number of fillers between the prime and the target, and, in some cases, the amount of linear time were controlled. Thus, if there are any differences in priming behavior, the structural context in which the prime occurred is the most likely cause.

Before turning to the studies in Chapters 3 and 4, I wish to highlight a few key features of language processing discussed above, focusing on their relevance to RICE. Recall that the RICE hypothesis, as repeated below, contends that context mediates recency effects.

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed

In order to test this hypothesis, I presented a model of language processing with a few key components beginning with relevant features of memory and linguistic knowledge.

> **Key Point 1:**
> Language processing is a series of coordinated goals. Satisfying these goals requires the use of long-term and working memory. The interaction of LTM and WM occurs

> in network of interacting buffers with the retrieval of declarative memories (chunks), the firing of procedural memories (rules), and application of unification operations to unify chunks.

Language processing is a memory-based, goal-oriented behavior that uses chunks, rules, and unification operations to satisfy a particular goal, for example producing or comprehending a sentence. A primary goal, such as 'process sentence,' generates secondary goals, or subgoals, such as 'process DP.' Each subgoal is satisfied through (i) the firing of production rules and (ii) the unification of the chunks retrieved and popped by the rules.

> **Key point 2:**
> The retrieval of both chunks and rules is sensitive to their activation/strength and their relation/utility given to the context.

Both a chunk's activation level and a rule's strength are sensitive to their retrieval history and the amount of time between the most recent use and the current context. Additionally, both chunks and rules are sensitive to the specific needs of the current context, e.g. if the processor is working on a 'process DP' subgoal or a 'process VP' subgoal. Other chunks in the context can increase or decrease the activation of another chunk. Similarly, the production rules associated with a given processing context or goal can increase or decrease the likelihood of another rule's retrieval. The likelihood of a particular chunk's retrieval depends in part on how high its activation weight is relative to other possible chunks. The likelihood of a particular rule's retrieval depends in part on how strong it is and how likely it is to satisfy a goal while incurring minimal cost.

> **Key point 3:**
> The retrieval of a form is sensitive to decay. This decay is determined by the amount of the time since its most recent activation (chunks) or firing (rules).

Forms receive a boost in activation after their retrieval. This boost decays over time. As this

boost decays, priming behavior decays. Decay for a retrieved chunk begins as soon as it has been popped from the retrieval buffer (chunks). Decay for a fired rule begins as soon as it has completed all the actions in the THEN part of it is rule. Decay is a constant function and should affect all forms equally.

> **Key point 4:**
> Unifications involved in the satisfaction of a common goal form a chain. The nature of these chains affects the speed and accuracy of recall for specific items in the chain.

The product of a unification cycle can subsequently unify with an open value in a chunk being held in the problem state buffer. When this occurs, the unification cycles form a chain. All the unification cycles that occur uninterrupted (i.e. the popped chunk unifies with the next chunk in the problem state) are part of the same unification chain. During subsequent retrieval, this chain of unification cycles is retrieved and inspected. The more elements within the chain, the slower and less reliable the search for a particular element within the chain.

Chapter 3 explores the following claim about how recency and structural context interact with regard to lexical priming:

> **Lexical Priming Claim**
> Lexical priming is sensitive to the structural configuration in which the lexical item occurs not just its linear position.

RICE assumes that the retrieval of a recently-processed chunk is not sensitive only to recency. The features of the unification chain in which the prime occurs also affect priming behavior. Chunks that occurred in longer chains demonstrate less priming than those that occur in shorter chains. This is explained in greater detail in Chapter 3. Standard accounts of lexical priming effects (e.g. Birch & Garnsey 1995; Fleischman & Gabrieli 1998; Lucas 2002; Para & Rosa

2002; McKoon, Ratcliff, Ward 1994; McNamara 2005; Ratcliff & McKoon 1992) claim that structural context affects lexical priming only when the prime occur in specific structural contexts, such as focus position of focus structures. Differences, such as whether the prime occurred in an argument clause (e.g. the internal complement of a noun) or an adjunct clause (e.g. a relative clause) should not affect priming behavior when time is held constant. The standard account claims that, as long as the structural contexts are all discourse neutral and the primes occur in the same linear position, there should be no differences in priming behavior among them.[1]

The second claim pertains to structural priming, i.e. the tendency to reuse recently encountered structural forms (e.g. Bock 1986b, Bock & Kroch 1989; Bock & Griffin 2000; Branigan, Pickering, McLean, & Steward 2006; Cleland & Pickering 2003; Ferreira 1996; Ferreira & Bock 2006; Frazier, Taft, Roeper, Clifton, & Ehrlich 1984; Levelt & Kelter 1982; Pickering & Branigan 1998):

> **Structural Priming Claim**
> Structural priming is sensitive to the structural configuration in which the prime occurs not just its linear position.

The ability of a structural pattern to prime depends on the availability of the prime's memory trace during subsequent processing. The availability of this prime depends, in part, on time. Primes that occurred more recently are more likely to demonstrate priming than those that did not occur recently. This tendency makes structural priming similar to lexical priming, and just as RICE claims that lexical priming is sensitive to more than just time, so too does it claims that

[1] Note that this effect pertains only to lexical priming and not semantic or referential priming as in Hofmeister (2008).

structural priming is sensitive to more than just time. Specifically, RICE contends that structural primes associated with structural contexts with shorter unification chains are more accessible and more likely to demonstrate priming than those associated with structural contexts with longer unification chains. I return to this point in greater detail in Chapter 4. This claim differs from the standard accounts of structural priming, which argue that the structural context in which a prime occurs does not affect structural priming (Branigan, Pickering, McLean, & Stewart 2006). The standard account argues that simply having processed a structural prime increases the likelihood of reusing the prime form. Thus, varying the sentence structure itself and varying the position of the structural prime or the structural context in which a prime occurs does not matter. There should be the same pattern of priming regardless of whether a prime occurs in one context (e.g. inside a matrix clause) or another (e.g. inside the internal complements of a verb).

The standard account of priming along with the RICE account are explained in greater detail in subsequent chapters. Specifically, Chapter 3 addresses the claim about recency, structural context, and lexical priming; and Chapter 4 addresses the claim about recency, structural context, and structural priming.

Appendix 2A: Declarative chunks

Name	Full Chunk	Abbreviated Chunk
S-chunk	isa : S num : sg spec : DP comp : VP tense : past/pres finite : finite	isa : S spec : DP comp : VP
S-gap-chunk	isa : S-gap num : sg spec : __ comp : VP tense : past/pres finite : finite gap: =NP	isa : S-gap spec : __ comp : VP gap : =NP
DP-*the*-chunk	isa : DP num : sg case : nom/acc/dat orth : *the* comp : NP	isa : DP orth: *the* comp : NP
NP-*king*-chunk	isa : NP case : nom num : sg orth : *king*	isa : NP orth: *king*
NP-*duke*-chunk	isa : NP case : nom/acc num : sg orth : *duke*	isa : NP orth: *duke*
NP-*duchess*-chunk	isa : NP case : dat num : sg orth : *duchess*	isa : NP orth: *duchess*
NP-*rubies*-chunk	isa : NP case : acc num : pl orth : *rubies*	isa : NP orth : *rubies*
VP-*like*-chunk	isa : VP num : sg-sg tense : pres orth : *like* comp : DP	isa : VP orth: *like* comp : DP
VP-*declare*-chunk	isa : VP num : sg-sg tense : past orth : *declare* comp : DP/CP	isa : VP orth: *declare* comp : DP/CP

VP-gap-*declare*-chunk	isa : VP-gap num : sg-sg tense : past orth : *declare* comp : __ gap : =S	isa : VP-gap orth: *declare* comp : __ gap : =S
VP-promise-chunk	isa : VP num : sg-sg tense : past orth : *promise* comp : DP : DP/PP	isa : VP orth: *promise* comp : DP : DP/PP
CP-chunk	isa : CP num : sg spec: Comp comp : S	isa : CP spec: Comp comp : S
Comp-*that*-chunk	isa : Comp case: acc orth : *that*	isa : Comp orth : *that*
RelC-chunk	isa : RelC num : sg spec: RelP comp : S-gap mod : NP	isa : RelC spec: RelP comp : S-gap mod : NP
RelP-*who*-chunk	isa : RelP num: sg case: nom/acc orth : *who*	isa : RelC orth : *who*
AdvC-chunk	isa : AdvC orth: Adv comp : S mod : S	isa : AdvC orth: Adv comp : S mod : S
Adv-*as*-chunk	isa : Adv orth : *as*	isa : Adv orth : *as*

Appendix 2B: Table of production rules

Name	Production Rule Syntax	English Description
Retrieve S (at the initial state)	=goal> [process S] =retrieval> isa : nil ==> +retrieval> isa : S	IF the goal chunk is is currently empty, but the control state is to process a sentence AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type sentence
Retrieve S (following a selector, e.g. CompC)	=goal> isa : CP head : =comp comp : =S =retrieval> isa : nil ==> +retrieval> isa : S	IF the goal chunk is of the type complement clause and it contains an open value for a complementizer and it contains an open value for an S AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type sentence
Retrieve DP	=goal> isa : S spec : =DP comp : =VP =retrieval> isa : nil ==> +retrieval> isa : DP	IF the goal chunk is of the type sentence and it contains an open value for a DP and it contains an open value for a VP AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type determiner phrase
Retrieve NP	=goal> isa : DP orth : *the* comp : =NP =retrieval> isa : nil ==> +retrieval> isa : NP	IF the goal chunk is of the type determiner phrase and contains *the* as its head and it contains an open value for a NP AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type noun phrase

Retrieve VP	=goal> isa : S spec : =DP comp : =VP =retrieval> isa : nil ==> +retrieval> isa : VP	IF the goal chunk is of the type sentence and it contains a DP as its specifier and it contains a VP as its complement AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type verb phrase
Retrieve CP (for internal complement of a verb)	=goal> isa : VP head : V comp : = CP =retrieval> isa : nil ==> +retrieval> isa : CP	IF the goal chunk is of the type verb phrase and contains verb as its head and it contains an open value for an CP AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type complement clause
Retrieve CP (for internal complement of a noun)	=goal> isa : NP head : N comp : =CP =retrieval> isa : nil ==> +retrieval> isa : CP	IF the goal chunk is of the type noun phrase and contains noun as its head and it contains an open value for an CP AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type complement clause
Retrieve Complementizer	=goal> isa : CP head : =Comp comp : =S =retrieval> isa : nil ==> +retrieval> isa : Comp	IF the goal chunk is of the type complement clause and it contains an open value for a complementizer and it contains an open value for an S AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type complementizer

Retrieve RelC	=goal> [process relative clause] =retrieval> isa : nil ==> +retrieval> isa : RelC	IF the goal chunk is is currently empty, but the control state is to process a relative clause AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type relative clause
Retrieve Relative Pronoun	=goal> isa : RelC spec : =RelP comp : =S =retrieval> isa : nil ==> +retrieval> isa : RelP	IF the goal chunk is of the type relative clause and it contains an open value for an RelP and it contains an open value for an S AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type relative pronoun
Retrieve AdvC	=goal> [process AdvC] =retrieval> isa : nil ==> +retrieval> isa : AdvC	IF the goal chunk is is currently empty, but the control state is to process an adverbial clause AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type adverbial clause
Retrieve Adverb	=goal> isa : AdvC spec : =Adv comp : =S =retrieval> isa : nil ==> +retrieval> isa : Adv	IF the goal chunk is of the type adverbial clause and it contains an open value for an Adv and it contains an open value for an S AND IF the retrieval buffer is currently empty THEN amend the retrieval buffer to retrieve a chunk that is of type adverbial conjunction

Pop S	=goal> [process S] =retrieval> isa : S spec: DP comp : VP ==> **!pop!** isa : S	IF the goal chunk is to process a sentence AND IF the retrieval buffer contains a sentence and the specifier is filled and the complement is filled THEN pop the content of the retrieval buffer
Pop S (following a selector, e.g. CP)	=goal> isa : CP head : =comp comp : =S =retrieval> isa : S spec: DP comp : VP ==> **!pop!** isa : S	IF the goal chunk is of the type complement clause and it contains an open value for a complementizer and it contains an open value for an S AND IF the retrieval buffer contains a sentence and the specifier is filled and the complement is filled THEN pop the content of the retrieval buffer
Pop DP	=goal> isa : S spec : =DP comp : =VP =retrieval> isa : DP orth : *the* comp : NP ==> **!pop!** isa : DP	IF the goal chunk is of the type sentence and it contains an open value for a DP and it contains an open value for a VP AND IF the retrieval buffer contains a determiner phrase and the specifier is filled and the complement is filled THEN pop the content of the retrieval buffer
Pop NP	=goal> isa : DP orth : *the* comp : =NP =retrieval> isa : NP head : N comp :nil/filled ==> **!pop!** isa : NP	IF the goal chunk is of the type determiner phrase and contains *the* as its head and it contains an open value for a NP AND IF the retrieval buffer contains a determiner phrase and its head is filled and either does not take a complement or its/ complement is filled THEN pop the content of the retrieval buffer

Pop VP	=goal> isa : S spec : =DP comp : =VP =retrieval> isa : VP head : V comp : nil/filled ==> **!pop!** isa : VP	IF the goal chunk is of the type sentence and it contains a DP as its specifier and it contains a VP as its complement AND IF the retrieval buffer contains a verb phrase and its head is filled and either does not take a complement or its/ complement is filled THEN pop the content of the retrieval buffer
Pop CP (for internal complement of a verb)	=goal> isa : VP head : V comp : = CP =retrieval> isa : CP spec : Comp comp : S ==> **!pop!** isa : CP	IF the goal chunk is of the type verb phrase and contains verb as its head and it contains an open value for an CP AND IF the retrieval buffer contains a complement clause and the specifier is filled and the complement is filled THEN pop the content of the retrieval buffer
Pop CP (for internal complement of a noun)	=goal> isa : NP head : N comp : = CP =retrieval> isa : CP spec : Comp comp : S ==> **!pop!** isa : CP	IF the goal chunk is of the type noun phrase and contains noun as its head and it contains an open value for an CP AND IF the retrieval buffer contains a complement clause and the specifier is filled and the complement is filled THEN pop the content of the retrieval buffer

Pop Complementizer	=goal>
	  isa : CP
	  spec : =Comp
	  comp : =S
	
	=retrieval>
	  isa : Comp
	  orth: *that*
	
	==>
	  !pop!
	  isa : Comp
		IF the goal chunk is
		  of the type complement clause
		  and it contains an open value for a Comp
		  and it contains an open value for an S
		
		AND IF the retrieval buffer
		  contains a complementizer
		  and its head is filled
		
		THEN
		  pop the content of the retrieval buffer

Pop RelC	=goal>
	  [process RelC]
	
	=retrieval>
	  isa : RelC
	  spec : RelP
	  comp : S
	
	==>
	  !pop!
	  isa : RelC
		IF the goal chunk is
		  is to process a relative clause
		
		AND IF the retrieval buffer
		  contains a relative clause
		  and its specifier is filled
		  and its complement is filled
		
		THEN
		  pop the content of the retrieval buffer

Pop Relative Pronoun	=goal>
	  isa : RelC
	  spec : =RelP
	  comp : =S
	
	=retrieval>
	  isa : RelP
	  orth: *who*
	
	==>
	  !pop!
	  isa : RelP
		IF the goal chunk is
		  of the type relative clause
		  and it contains an open value for an RelP
		  and it contains an open value for an S
		
		AND IF the retrieval buffer
		  contains a relative pronoun
		  and its head is filled (e.g. *who*)
		
		THEN
		  pop the content of the retrieval buffer

Pop AdvC	=goal>
	  [process AdvC]
	
	=retrieval>
	  isa : AdvC
	  spec : Adv
	  comp : S
	==>
	  !pop!
	  isa : AdvC
		IF the goal chunk is
		  is to process an adverbial clause
		
		AND IF the retrieval buffer
		  contains an adverbial clause
		  and its specifier is filled
		  and its complement is filled
		
		THEN
		  pop the content of the retrieval buffer

Pop Adv	=goal>
	  isa : AdvC
	  spec : =Adv
	  comp : =S
	
	=retrieval>
	  isa : Adv
	  orth: *as*
	
	==>
	  !pop!
	  isa : Adv
		IF the goal chunk is
		  of the type adverbial clause
		  and it contains an open value for an Adv
		  and it contains an open value for an S
		
		AND IF the retrieval buffer
		  contains and adverbial conjunction
		  and its head is filled (e.g. *as*)
		
		THEN
		  pop the content of the retrieval buffer

Push S	=goal>
	  [process S]
	
	=retrieval>
	  isa : S
	  spec: =DP
	  comp : =VP
	
	==>
	  !push!
	  isa : S
		IF the goal chunk is
		  to process a sentence
		
		AND IF the retrieval buffer
		  contains a sentence
		  and the specifier is unspecified
		  and the complement is unspecified
		
		THEN
		  push the content of the retrieval buffer into the problem state buffer

Push S
(following a selector, e.g. CP)
	=goal>
	  isa : CP
	  head : =comp
	  comp : =S
	
	=retrieval>
	  isa : S
	  spec: =DP
	  comp : =VP
	
	==>
	  !push!
	  isa : S
		IF the goal chunk is
		  of the type complement clause
		  and it contains an open value for a complementizer
		  and it contains an open value for an S
		
		AND IF the retrieval buffer
		  contains a sentence
		  and the specifier is unspecified
		  and the complement is unspecified
		
		THEN
		  push the content of the retrieval buffer into the problem state buffer

Push DP	=goal>
	  isa : S
	  spec : =DP
	  comp : =VP
	
	=retrieval>
	  isa : DP
	  orth : *the*
	  comp : =NP
	
	==>
	  !push!
	  isa : DP
		IF the goal chunk is
		  of the type sentence
		  and it contains an open value for a DP
		  and it contains an open value for a VP
		
		AND IF the retrieval buffer
		  contains a determiner phrase
		  and the specifier is filled
		  and the complement is unspecified
		
		THEN
		  push the content of the retrieval buffer into the problem state buffer

Push NP	=goal> isa : DP orth : *the* comp : =NP =retrieval> isa : NP head : noun comp : nil/=CP ==> **!push!** isa : NP	IF the goal chunk is of the type determiner phrase and contains *the* as its head and it contains an open value for a NP AND IF the retrieval buffer contains a determiner phrase and its head is filled and its complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer
Push VP	=goal> isa : S spec : =DP comp : =VP =retrieval> isa : VP head : V comp : =CP ==> **!push!** isa : VP	IF the goal chunk is of the type sentence and it contains a DP as its specifier and it contains a VP as its complement AND IF the retrieval buffer contains a verb phrase and its head is filled and its complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer
Push CP (for internal complement of a verb)	=goal> isa : VP head : V comp : = CP =retrieval> isa : CP spec : =Comp comp : =S ==> **!push!** isa : CP	IF the goal chunk is of the type verb phrase and contains verb as its head and it contains an open value for an CP AND IF the retrieval buffer contains a complement clause and the specifier is unspecified and the complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer
Push CP (for internal complement of a noun)	=goal> isa : NP head : N comp : = CP =retrieval> isa : CP spec : Comp comp : =S ==> **!push!** isa : CP	IF the goal chunk is of the type noun phrase and contains noun as its head and it contains an open value for an CP AND IF the retrieval buffer contains a complement clause and the specifier is filled and the complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer

Push RelC	=goal> [process RelC] =retrieval> isa : RelC spec : =RelP comp : =S ==> **!push!** isa : RelC	IF the goal chunk is is to process a relative clause AND IF the retrieval buffer contains a relative clause and its specifier is unspecified and its complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer
Push AdvC	=goal> [process AdvC] =retrieval> isa : AdvC spec : =Adv comp : =S ==> **!push!** isa : AdvC	IF the goal chunk is is to process an adverbial clause AND IF the retrieval buffer contains an adverbial clause and its specifier is unspecified and its complement is unspecified THEN push the content of the retrieval buffer into the problem state buffer

3 CHAPTER

Lexical Priming

> The three most important factors in buying a home are: location, location, location! ~ *Unknown*

During language processing, speakers and listeners search through memory to find the right word at the right time. The ability to locate the correct word depends, in part, on how closely it matches the needs of the current discourse. For example, in discussions about pets, the word *cat* is found and retrieved more quickly than the word *sage*, but in discussions about herbs, the *sage* is more likely than *cat*. However, more than just the current discourse context affects retrievability.

A form's retrievability is also affected by its ACTIVATION WEIGHT, which is a numerical value that reflects the history of use of the form (see Chapter 2, section 3). Forms with higher levels of activation, or greater activation weights, are more likely to be retrieved than those with lower levels or less weight. The reason for this tendency is that more active forms are easier for the processor to locate and, hence, retrieve and use for processing. One crucial factor in determining a form's activation weight is the recency of its use. Forms that have recently been encountered have higher activation weights than those that have not been recently encountered. For instance, at this moment, the herb *sage* is probably more active than *thyme* due to *sage*'s recent mentioning. When a form has been recently encountered (processed), its activation weight

is higher, and it is more likely to affect subsequent linguistic behavior. This general effect, which I call the RECENCY EFFECT, has been found in numerous tasks in which recently-encountered forms influence subsequent performance (e.g. Bock 1986a,b; Bjork & Whitten 1974; Deese & Kaufman 1957; Murdock 1962; Davelaar, Goshen-Gottstein, Haarmann, Ashkenazi, & Usher 2005; Howard & Kahana 1999; McNamara 2005; Pickering & Branigan 1998). The RICE hypothesis accepts this general claim and adds one stipulation, namely that recency effects are mediated by how the structural context in which the prime occurs was processed.

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed.

In this chapter, I test this hypothesis using lexical priming, specifically using a form of similarity-based lexical priming called IDENTITY or REPETITION priming (henceforth *identity priming*). Identity priming refers to the processing facilitation an item receives because the same lexical item was recently encountered, meaning that the prime and target are the same. This form of priming differs from other forms of lexically-based priming, such as semantic priming, as is described in greater detail in section 1.

According to the standard account of priming and recency effects, the reuse of recently encountered words should not be affected by the processing of the larger structural context in which the prime occurred. Most broadly, this account predicts that there should not be some systematic difference among lexical primes based solely on their larger structural context. A generous interpretation of this account would potentially allow for some elements of "context" (e.g. pragmatic, discourse, or semantic context) to facilitate priming, but the strictest

interpretation would rule out any facilitation that corresponds to systematic structural differences. However, there is research suggesting that this claim in its strongest interpretation is not correct. Antecedents that occur in the focus position of a focus construction (e.g. *it*-clefts and *wh*-clefts such as (1) and (2) below) facilitate response times at their anaphoric phrases (Birch & Garnsey 1995; Clifton, Kennison, & Albrecht 1997; McKoon, Ratcliff, Ward, & Sproat 1993; Nicol & Swinney 1989; Spivey, Tanenhaus, Eberhard, & Sedivy 2002; Sturt 2003; Swinney 1979; van Berkum, Brown, & Hagoort 1999). For examples (1) and (2) below, the antecedents in the focus position (bolded) lead to quicker responses at the target item than those in the deemphasized position (italicized) (Almor 1999).

(1) Focus item in the scope of an *It*-cleft
It was **the robin** that ate *the apple*.

Continuation: The bird/The fruit…

(2) Focus item in the scope of a *Wh*-clef
What *the robin* ate was **the apple**.

Continuation: The bird/The fruit…

What this research suggests is that items that occur in focused, 'emphasized' positions, i.e. forms occurring in structurally and pragmatically foregrounded positions, affect subsequent linguistic behavior more than those in 'deemphasized' positions, i.e. forms occurring in structurally subordinated, pragmatically backgrounded positions in cleft sentences. These findings appear to conflict with the standard account of priming, which claims structural context is irrelevant.

However, the focus effect is inconsistent. Research suggests that the focus effect is not as stable as assumed and that it may be sensitive to particular demands of task (Almor & Eimas

2008; Birch, Albrecht, & Myers 2000; see also Chapter 2, section 1). Items in focus positions ((3) below) do not facilitate priming any more than those in neutral contexts (i.e. contexts that are not part of focus constructions such as *it*-clefts or *wh*-clefts) ((4) below).

(3) Antecedent in focus position
It was **the mayor** who refused to answer a reporter's question.

(4) Antecedent in neutral position
The mayor refused to answer a reporter's question.

Focus position in and of itself may not affect the retrievability of specific words. Rather, it may be something about being in deemphasized positions or the task demands of some of the previous focus studies driving the effect.

Given that the source and the actual implications of the focus effect are unclear, the standard account that structural context doesn't matter may still hold. There is no clear evidence that supports the claim that structural context affects the retrievability of forms. The standard account contends that when time is held constant (e.g. the number of milliseconds or syllables between the prime and target are constant), then the primes should show the same priming behavior regardless of where in the larger structural context they occur.

The experimental results discussed below provide evidence against this claim. There is a difference between the retrieval of prime forms that occur in the internal complements of nouns (henceforth *noun complement clauses*, e.g. the underlined portion of (5)) and the retrieval of forms that occur in the internal complements of verbs (henceforth *verb complement clauses*, e.g. (6)), in relative clauses (e.g. (7)) or in matrix clauses (e.g. (8)).

(5) Noun complement clause
Amanda declared the fact that the lawyer lived in Washington.

(6) Verb complement clause
Amanda declared that the lawyer lived in Washington.

(7) Relative clause
Amanda liked the lawyer who lived in Washington.

(8) Matrix clause
Amanda worked in Virginia, and the lawyer lived in Washington.

Ultimately, I argue that this difference in priming stems from differences in the way forms are unified into larger structural units. UNIFICATION refers to the merger of two linguistic forms to generate a new, equally as complex or more complex form (Chapter 2, section 3.4). Each unification of one linguistic form with another form counts as a UNIFICATION CYCLE. Unification cycles act as a form of bookkeeping, a way of counting and tracking the steps used to process a linguistic unit such as a clause or sentence. I contend that the number of unifications and how they are joined into larger units affect the retrievability of a prime at the target.

Before delving into the interaction between the pattern of unifications and lexical priming, in section 1, I specify the type of lexical priming I am considering. In section 2, I discuss the different predictions of the standard account of priming (SAP) and the RICE-inspired account of priming (PRICE) as they pertain to lexical priming. Section 3 presents a lexical priming study that tests these predictions, and section 4 discusses the results. Sections 5 and 6 present a discussion of the findings and the conclusions respectively.

1. Defining lexical priming

LEXICAL PRIMING refers to the facilitation a word receives because it (or a related word) was recently processed. The term 'lexical priming' has been used to describe facilitation effects such as lexical repetition (e.g. quicker processing of and more frequent reuse of the same word) and semantic priming (e.g. quicker processing of semantically-related words) (e.g. Baayen, Dijkstra, & Schreuder 1997; Clahsen & Featherston 1999; Friederici, Steinhauer, & Frisch 1999; Glosser & Friedman 1991; Hutchinson 2003; Lucas 2000; Ferrand & New 2003; Perea & Rosa 2000, 2002; Rips, Shoben, & Smith 1973; see McNamara 2005 for a review). I take the former definition: lexical priming refers strictly to priming for forms of the same lemma. For example, lexical priming includes facilitation for *cats* after the word *cat* but not for semantically related words such as *kitten, pet,* or *dog.* I consider semantic priming to be a separate form of priming despite its similarity to other lexically-based forms of priming.

1.1 Forms of lexical priming

Generally speaking, there are three levels of lexical processing that lead to lexical priming in the broad sense (Allen & Badecker 2002; Dell & O'Seaghdha 1991; Hoey 2005; Levelt, Roelofs, & Meyers 1999; Levelt, Schriefers, Vorberg, Meyer, Pechmann, & Havinga 1991; McNamara 2005):

i. the 'semantic' or conceptual level (the lemma *walk* is associated with the concept of MOVEMENT, LEGS, etc.);

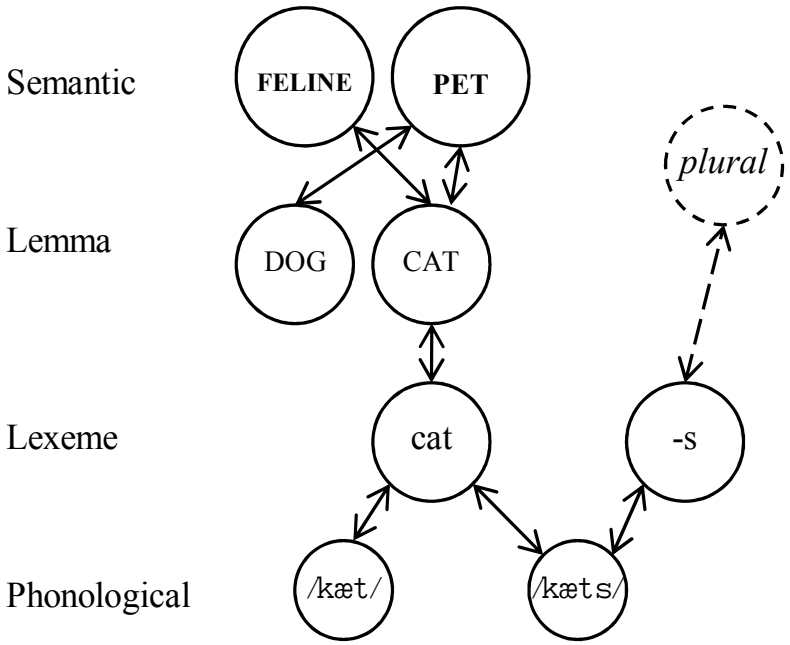
ii. the lemma level (e.g. the abstract lexical properties and morphosyntactic features of a word); and

iii. the lexeme level (e.g. the specific, structural form of the word).

Allen and Badecker (2002) argue that lexical priming arises when two forms access the same lemma entry. Thus *walked* primes *walks* and *walking*. However, some primes appear to inhibit responses to targets that share the same lemma entry. For instance, irregular verbs (*give*→*gave*) do not show the same amount of priming as regular verbs (*walk*→*walked*). Allen and Badecker contend that forms that have a great deal of surface similarity (e.g. they have a great deal of orthographic overlap as in *give* and *gave*) serve to inhibit one another in priming tasks, despite their shared lemma. Because of this, they argue that the lemma and lexeme level are distinct and affect behavior separately.[1] In Chapter 2, section 3.1, I presented Figure 2.1, which depicts the lexicon as having nodes at three levels: the semantic, lemma, and phonological levels. Below, I expand these levels to clarify the differences between the lemma and lexeme level, building off of Allen and Badecker's (2002) model. Assume that we have a representation in memory similar to that in Figure 3.1 below.

[1] This two-level distinction based on structural or surface similarity (e.g. the difference between the lemma and lexeme levels) is found at other levels of information processing beyond just linguistic processing. Forbus, Gentner, and Law (1995) argue that retrieval from long-term memory is a two-step process. At the early stages, surface similarity narrows the set of possible choices. Then structural similarity delimits the set even more. As such, the two types of information are represented separately and affect behavior separately.

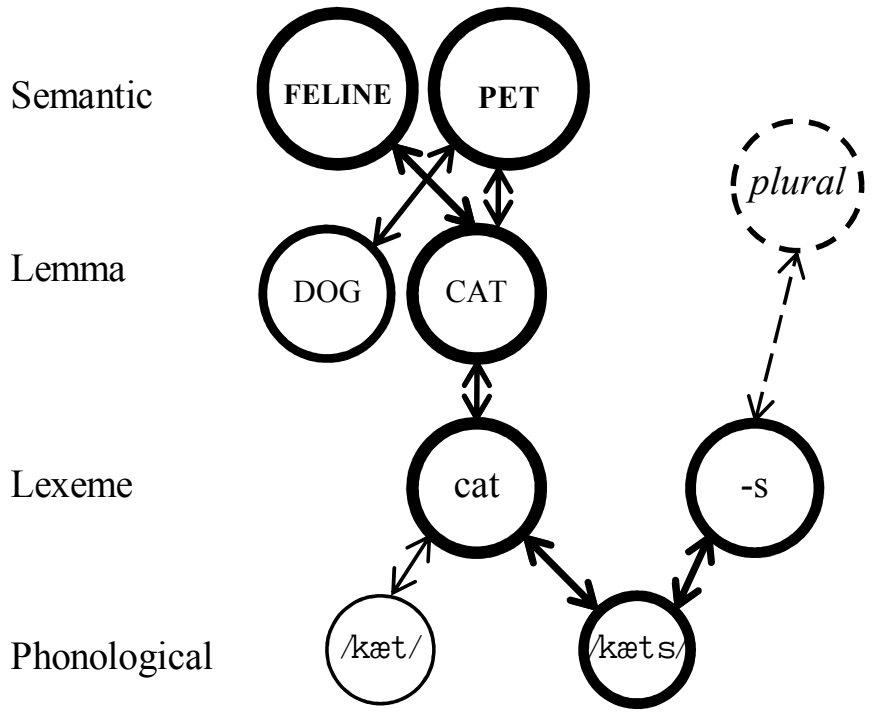
Figure 3.1: Four levels of *cat* according to Allen and Badecker's (2002) model



Here we see that the lemma CAT (all caps) is connected to semantic information (bold and all caps) and to different lexemes (normal font), which are connected to their phonetic realizations. I refer to these levels as the lemma, semantic, lexeme, and phonological levels respectively. The lexeme level contains forms based in lexical representations whereas those at the lemma are considered modality-neutral and more abstract (Allen & Badecker 1999, 2002).

As discussed in Chapter 2, once a form is encountered (e.g. the processor hears /kæts/), activation spreads throughout the network, as shown in Figure 3.2. This spreading activation leads also to the activation of semantically-related forms (e.g. DOG) as shown by the lesser bolded line and circle leading from **PET** to DOG.

Figure 3.2: Spreading activation from *cat*



One commonly-noted difference between SIMILARITY-BASED PRIMING, which refers to lemma- or lexeme-level priming, and SEMANTICALLY-BASED PRIMING, which refers to semantic-level priming, is the apparent difference in the persistence of their priming effects. Many contend that the effects of similarity-based priming (such as identity priming) can last anywhere from minutes to months, whereas the effects of semantically-based priming may fade as soon as the sentence has been processed (e.g. Bentin & Feldman 1990; Bentin & Moscovitch 1988; Bowers 2000; Jäger & Rosenbach 2008a,b; McClelland & Rumelhart 1986; Morton 1969; McNamara 2005; Ratcliff & McKoon 1994; Scarborough, Cortese, & Scarborough 1977; Sloman, Hayman, Ohta, Law, & Tulving 1988; Tenpenny 1995; but see Becker, Moscovitch, Behrmann, &

Joordens 1997; Joordens & Becker 1997 for arguments for long-term semantic priming). One reason for the possible differences may be the nature of the relationship between the prime and target exhibited by both types of priming. In semantic priming, the associative or conceptual link between the prime (e.g. an antecedent) and a target (e.g. an anaphor) is being primed. The use of one form (e.g. a specific lemma) activates a concept, and this activation is assumed to activate other forms that are also linked to the primed concept. For example, processing the phrase "the tabby" can prime the processing of "the cat." However, the link may be discourse-specific.

Anaphor resolution and the reactivation of antecedents demonstrate this type of discourse-specific association and sensitivity to current discourse-context needs (Cowles & Garnham 1995; Cowles, Walenski, & Kluender 2007; Gernsbacher 1989; Love & Swinney 1996; Nicol 1993; Nicol, Fodor, & Swinney 1994). For example Cowles et al. (2007) presented participants with short stories such as (9):

(9) Example from a pronoun-resolution anaphora lexical priming study (Cowles *et al.* 2007)

Setup:

a) Anne wanted to see the new movie with Sarah.
b) So, Anne called Sarah.

Target:

c) But later that night, **she** couldn't go to the movie after all.

Participants listened to the setup and target sentences and then read the name of one of the participants (e.g. *Anne* or *Sarah*) when it appeared on the computer screen in front of them. Cowles et al. took shorter speech onset latencies as evidence of priming. Thus, if *she* in (9c) reactivated (primed for) *Anne*, then there should be shorter latencies in the pronunciation of *Anne* relative to the pronunciation of *Sarah* during the trial. They found an effect of priming such that

depending on discourse features (e.g. topicality and prominence), *she* primed for *Anne* more than for *Sarah.* These sort of discourse-context links need to be flexible to allow for quick retrieval in the short term and also quick deactivation so that discourse referents can wax and wane and new antecedent-pronoun pairs can be established. However, in similarity priming, the connection between a form and its lemma or lexeme is being tested. The connection between a lemma (e.g. *walk*) and its instantiations (e.g. *walked, walking,* or *walks*) persists beyond the current text and as such may demonstrate long-term effects and less dependence on structural context.

Although the possible differences between the persistence of these two types of priming should be enough to motivate testing them separately, there is one additional reason to focus on similarity-based priming (henceforth *lexical priming*) rather than semantically-based priming (henceforth *semantic priming*). Research on the two forms of priming has often explored different phenomena. Of particular interest to us are the ways the two forms of priming have been used to explore the effects of structural context on the different forms of priming behavior.

Semantic priming has been used extensively to test context effects on the priming of semantically-related word pairs, anaphor resolution, and filler-gap dependencies (Almor 1999; Birch, Albrecht, & Myers 2000; Birch & Garnsey 1995; Clifton, Kennison, & Albrecht 1997; Cowles, Walenski, & Kluender 2007; Foraker & McElree 2007; Hofmeister 2008; Love & Swinney 1996; Morris & Folk 1998; Nicol 1993; Nicol, Fodor, & Swinney 1994; Nicol & Swinney 1989; Sturt 2003; Swinney 1979). These studies generally find that context heightens semantic priming. For example, being in the focus position of a cleft sentences leads to quicker performance in certain tasks (see Chapter 2, section 1 for a fuller discussion). However, research

on the effects of structural context on lexical priming lags behind, although there is research to suggest that structural context may not affect lexical discrimination (Birch *et al.* 2000; Connine, Blasko, & Wang 1994, see also Chapter 2, section 1) and, thus, may not affect lexical priming. Because semantic priming may be more transitory, as mentioned above, it may also depend more heavily on the current structural context for priming whereas the more stable, lexical priming may be independent of the structural context.

In this chapter, I explore how structural context affects lexical priming (i.e. form-based priming). To do so, I probe the activation of a word by using identity priming in which the prime and target are the same word in the same form (i.e. same tense). Specifically, participants hear a sentence containing a prime word (e.g. *bought*) and then determine whether the sentence they just heard contained the word they see on a computer screen (e.g. the same word *bought*). This type of identity priming task should activate the lexeme-level and the lemma-level (as well as the semantic level), giving the prime and target the greatest chance of facilitation (Luckatela, Savic, Urosevic, & Turvey 1997).

2. The activation-based model account for lexical priming effects

As discussed in Chapter 2, activation-based models of language processing assume that

(i) all linguistic forms can be represented as nodes in long-term memory,

(ii) that there are links among these nodes, and

(iii) that both the nodes and the links have activation weights which reflect the history of use for the node or link.

During language processing, the processor searches memory for a particular form (e.g. a word), and these activation weights come into play. Ultimately, there are three factors that determine which word is retrieved: relation to context, activation, and random noise. I mention this third factor only to note that it can affect retrieval, and I do not consider it further.

The first factor, RELATION, is determined by the weighted associations between the elements associated with the current goal and the connections between these elements and a particular chunk (see Chapter 2, section 3.3 for a fuller discussion of relevance and the activation of chunks). The more elements associated with a goal, the less weight each element receives. For example, previous research has found that the ability to verify facts about a referent (e.g. whether John plays soccer) depends on how many other facts you know about the same referent (John plays tennis, likes Mozart, studies linguistics, etc.) (e.g. Anderson 1974). The more facts one knows, the longer it takes to verify a particular fact. The explanation is that the number of facts associated with the referent compete with one another, thereby attenuating the activation of each individual fact. At the same time, the more elements in the context pointing toward the same chunk, the greater the weight for the chunk in question. For example, some words are likely to co-occur (e.g. "Mickey" and "Mouse") such that the occurrence of one may boost the activation of the other. Although this is a relevant factor for determining activation weights, Anderson & Schuun (2000) contend that it is the least important aspect at least as far as learning is concerned. Still, others have found that the co-occurrence of words (e.g. *baby-hospital* versus *baby-concrete*) affects retrieval and priming behavior (e.g. McKoon & Ratcliff's (1992) 'compound cue' account of semantic priming). Thus, we have two factors to consider: (i) how

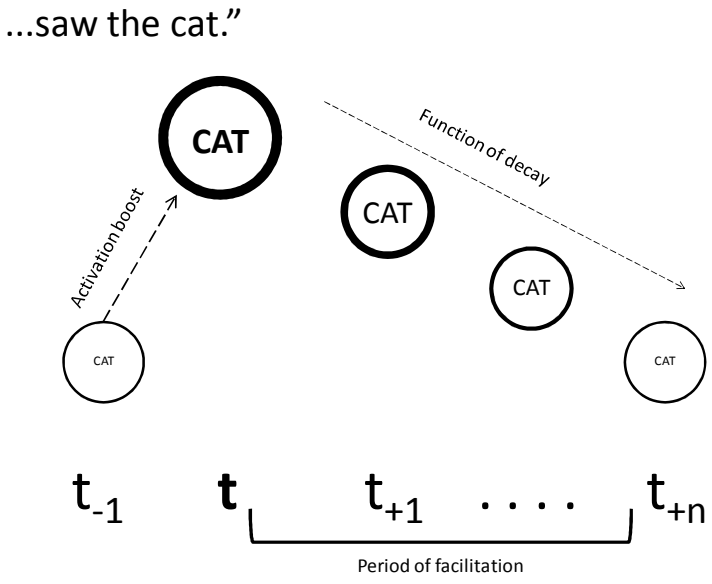
many chunks are associated with a particular context and (ii) the strength of the link between two chunks.

The first of these factors reflects the current state of the processor by determining how much weight each element should receive given the number of elements needed for the processing of the sentence. The more elements, the less activation available for each element. The second factor reflects the history of use of one chunk given the use of another chunk, thereby capturing an element of contextual priming associated with the co-occurrence of two chunks. The stronger the links between two elements, the greater the activation boost a chunk receives. These two factors estimate the amount of activation a chunk receives from its context.

Of particular interest to us is the amount of interference that can arise from competing chunks in the context. This interference can lead to potentially slower or less accurate performance (e.g. Lewis & Vasishth 2005, Van Dyke & Lewis 2003, and Van Dyke & McElree 2006). Interference can arise when many forms are vying for limited cognitive resources or when the links between forms are ambiguous (e.g. they point to many possible other forms). Interference can be one of two types: proactive or retroactive interference. Proactive interference refers to difficulty integrating incoming information due to old information (e.g. trying to remember where you parked today when you normally park in a different location), and retroactive interference refers to difficulty re-accessing old information due to new information (e.g. trying to remember an old phone number after having memorized a new one). Either form of interference makes it more difficult for the processor to retrieve chunks from memory (Lewis, Vasishth, & Van Dyke 2006).

Another factor that affects retrieval is the word's ACTIVATION WEIGHT, which is determined by a combination of the word's base-level activation and any boost in activation it received from recent processing minus a function of decay (Chapter 2, section 3.3.2). This 'recent processing' can refer to the processing of the same word, in which case the weight adjustment is a direct consequence of the form's use, or from the processing of a semantically-related word, in which case the adjustment is a consequence of spreading activation. Regardless, after a word's activation weight is adjusted and the processor proceeds to the next phase (e.g. it begins to retrieve the next word), the activation boost for the word begins to decay at a constant, set rate. For example, say someone heard the sentence "The child saw the cat." After the processing of the word *cat,* the form *cat*'s representation in memory receives a boost, as shown in Figure 3.3 below.

Figure 3.3: Activation and decay of *cat*



Time *t* denotes the point at which the representation receives its activation boost. Once the processor has moved on to process the next word (t_{+1}), the activation boost for *cat* begins to decay at a constant rate, denoted by the decreasing size of the nodes and the slope above. While this decay process is taking place ($t_{+1...}t_{+n}$), the activation weight for the word is still higher than it would have been had it not been recently processed, giving it a slight advantage over competing alternative forms. During the period in which the activation boost is waning, processing of *cat* is facilitated. This interaction of retrieval, activation boost, and decay leads to the recency effects in priming as discussed in Chapter 2.[1] At the next point in processing when a choice must be made between the retrieval of the recently-used word and an acceptable alternate, the three factors of relevance, activation, and noise come into play again. However, this time the recently-retrieved word may have a distinct advantage over other, possible forms due to its recent boost in activation. This 'heightened activation' is the source of the recency effect.

2.1 Comparing the accounts

The general assumption about the effects of recent processing applies both to the standard account of lexical priming and the RICE account of lexical priming. Where the two accounts differ is in how they factor in the processing of the larger structural context in which the prime word occurs. The standard account predicts that the retrievability of a prime form is not affected by the processing of the structural context in which the word occurs. By the standard account, only recency and the amount of decay matters (Bentin & Moscovitch 1988; Ratcliff, Hockley, &

[1] Keep in mind that, even though the activation boost decays over time, there is a cumulative or residual effect of the activation boost, meaning that there is a gradual accrual of weight that ultimately raises the base-level activation weight for the word.

McKoon 1985; Scarborough et al. 1977). The onset of decay depends, in large part, on a form's linear position, so primes that occur in the same sentence position (e.g. sentence final) should be equally as "recent" regardless of whether they were in matrix position or embedded in a noun or verb complement clause. The Standard Account of Priming (SAP) can be best summarized in the following way:

> **Standard Account of Priming (SAP)**
> Having recently encountered a linguistic form increases the likelihood of that form's subsequent reuse.

This hypothesis predicts an effect of time only. Structural differences between the contexts in which the prime appears are irrelevant. For example, in the sentences below, the prime *cat* occurs in the same position for all the sentence types. In these sentences, the number of words (i.e. zero) and the amount of time (as denoted by the ellipsis) is the same across all contexts, meaning that all the instances of *cat* occur the same amount of time away from the target. Furthermore, in these examples, *cat* occurs approximately the same number of words away from the beginning of the sentence.

Table 3.1: Structural context and priming in the standard account

Sentence type	Prime	Delay	Target
Matrix clause	I saw the dog, and the child saw the cat.	. . . . .	CAT
Relative clause	We both know the child who saw the cat.	. . . . .	CAT
Verb complement clause	We both know that the child saw the cat.	. . . . .	CAT
Noun complement clause	We both know the fact that the child saw the cat.	. . . . .	CAT

According to the standard account, the primes in each sentence type begin their decay at the same time regardless of their structural position, and the amount of decay is equal across types, as shown above by the bracket 'delay' section. The SAP contends that encountering a word increases the probability of its reuse as long as the target occurs before the prime form's activation boost has fallen beneath some threshold. There is no mediating factor of context. How the structural context of the prime was processed does not affect priming in general. In contrast, the RICE hypothesis claims that the structural context of the prime matters.

The RICE hypothesis stems from the assumption that different structural contexts lead to different patterns of processing. These different patterns of processing arise from patterns of subgoals generated during processing. For example, the processor may generate the subgoal of processing a relative clause or processing a matrix clause. Although these two clauses may ultimately be part of the same sentence, they are distinct goals with distinct subgoal structures. During processing, the processor generates separate unification chains that reflect these subgoal structures (see Chapter 2, section 3.4 for a discussion of unification operations and the formation of unification chains). These unification chains reflect the processing of different structural contexts, and ultimately the features of the chains affect the accessibility of particular forms that occur in the chains. Depending on features of the unification chain (e.g. length), the prime is more or less easily retrieved.

According to activation-based models, such as the one I presented in Chapter 2, the retrieval of a linguistic form is affected by its total activation and its relation to the context. A form's activation level is a function of its base level, which reflects its entire history of use, and

any boost from recent processing minus an element of decay. The context can buttress a form's activation when other elements linked to a form occur in the context. At the same time, the context can undermine the activation of a form if there are too many elements associated with the current goal. This is where the features of unification chains come into play.

The model of language processing presented in Chapter 2 suggests that the structural context of a prime affects subsequent behavior (e.g. the speed and accuracy of a word's retrieval) by mediating the amount of priming a form receives. Some structural contexts have fewer elements (e.g. fewer words that need to be unified) than others. The chunks used in a sentence are associated with unification chains. Two sentences may have the same total number of chunks but differ in how these chunks are associated to one another. Specifically, chunks are grouped into larger units, and these units reflect unification chains. The unification chains act as retrieval structures, and when retrieved, all of the chunks associated with the chain are retrieved. For example, say the processing of a sentence results in tow unification chains, one associated with four chunks and one associated with six chunks. During subsequent processing, the processor retrieves the chain with the four chunks to verify whether a particular word occurred in it. The processor then needs to determine the relations among the four chunks. However, if the processing of the sentence only resulted in one chain and the chain associated with ten total chunks, then during subsequent retrieval, the processor would need to determine the relations among all ten of the chunks. The more words within the chain, the greater the possible processing interference (I return to this in section 5).

As the number of elements associated with a unification chain increases, the amount of cognitive resources any particular element receives decreases. At the point of a subsequent retrieval, the processor must retrieve the entire unification chain that the prime is associated with. When working memory must retrieve larger unification chains to search for a particular form, processing slows down and/or becomes less accurate. Processing differences can arise during the initial processing of a sentence (e.g. more elements in the context makes it harder for the processor to process each successive element) or at subsequent retrieval of the sentence (e.g. more elements in the context makes it harder for the processor to sort among them). Thus, processing difficulties due to structural context can arise during either initial processing or subsequent retrieval of a sentence and its forms. The current study does not attempt to distinguish between these two possibilities. However, I contend that if there are problems at either point in the processing, priming should decrease.

The current study is a step toward asking whether structural context affects priming at all. In the present chapter, I test the basic hypothesis that it does and the more specific claim stated below:

> **Priming According to RICE (PRICE)**
> The processing of both a prime form and its structural context affects how the form is represented, and differences in these representations affect subsequent priming behavior.

2.2 Predictions of the SAP and PRICE

Both the SAP and PRICE accounts contend that recency matters. Having recently processed a linguistic form facilitates reuse of that same form. If a speaker processes a specific word, she is primed to respond more quickly to that word in a subsequent task than if she hadn't

recently processed it. Both SAP and PRICE maintain that this facilitation should persist as long as the activation weight keeps the primed form more active than competing forms. In other words, there should be priming until decay has caused the activation weight for the prime form to fall beneath some threshold. Where the two accounts differ is in whether they predict that the structural context of a prime can also affect the retrieval of the prime. The SAP states that structural context does not affect lexical priming. PRICE states that it does. RICE proposes that structural context mediates recency effects. PRICE claims that the features of the unification chains that the prime is associated with affect priming behavior: the more elements associated with the chain, the less priming for the forms associated with the chain.

3. Experiment: Lexical priming from different structural contexts

This experiment was designed to determine whether the reactivation of words is affected by the structural context in which the words recently occurred. To determine the ease of reactivation, I used response times in an identity priming task. The assumption is that the faster the response, the quicker the retrieval. Recent processing of a form (e.g. a word) should facilitate subsequent retrievals of the form at least until its activation boost wanes and the form's activation weight drops. If the SAP is correct, only the amount of decay affects priming, and this decay is not affected by the structural context of the prime. Thus, all structural contexts lead to equal amounts of priming. If PRICE is correct, the features of the structural context that the prime is associated with (e.g. the unification chain's length and the number of chunks associated with it) affect priming. Specifically, primes that are associated with structural contexts that contain more

chunks show less priming than those that occur in other structural contexts with fewer chunks, even when the time between the prime and target is held constant.

The experiment presented here used a cross-modal priming design in which participants listened to sentences and then verified whether a word on the computer screen did or did not occur in the sentence. Cross-modal priming has been used widely in research that explores the activation of words or concepts. In these experiments, participants hear a sentence read aloud and then perform a lexical decision, naming, or probe recognition task (e.g. Allen & Badecker 2002; Boland, Tanenhaus, Garnsey, & Carlson 1995; Callahan, Shapiro, & Love 2008; Clahsen & Featherston 1999; Connine, Blasko, & Wang 1994; de Goede 2007; Love & Swinney 1996; Nakano, Felser, & Clahsen 2002; Nicol, Fodor, & Swinney 1994; Shapiro, Oster, Garcia, Massey, & Thompson 1999; Wester, de Goede, Bastiaanse, Shapiro, & Swinney 2004). By using cross-modal priming, I tried to avoid (i) purely visual or orthographical priming, which could arise from the presentation of strictly written material, and (ii) effects that may arise due to similarity-based priming's sensitive to modality (Allen & Badecker 2002).

3.1 Experimental items

For the experimental items,[1] the prime and probe word was always a dative verb in the past tense (such as the prime word bolded in (10) below). The prime was followed by two definite noun phrases (underlined below).

(10) The manager left the request, and the secretary **bought** the supplies for the owner.

[1] Appendix 3A contains all the experimental items used in this study.

Each of the noun phrases was three syllables long, and one occurred in a prepositional phrase. Thus, the probe verb occurred seven syllables before the end of the sentence.[1]

A total of eight target verbs were used: *buy*, *offer*, *sell*, *show*, *promise*, *hand*, *issue*, and *pass*. All of the verbs were presented in the past tense in both the prime sentences and the target task. There were four scenarios for each of the verbs, meaning that each verb occurred with four different sets of agents and objects. An example of two different scenarios for the verb *bought* is shown in (11) and (12).

(11) Matrix prime: Scenario 1
The manager left the request, and the secretary **bought** the supplies for the owner.

(12) Matrix prime: Scenario 2
The reporter smiled, and the agent **bought** the diamonds for the singer.

For each scenario, four sentence types were constructed as shown in Table 3.3. The structural context of the prime (i.e. matrix clause, noun complement clause, verb complement clause, and relative clause) are underlined beginning with the subordinator (e.g. the complementizer *that*) if there is one. The prime word is bolded.

Table 3.2: Example of four versions of one scenario

Context	Sentence
Matrix clause	The manager left the request, and the secretary **bought** the supplies for the owner.
Noun complement clause	The manager reported the fact that the secretary **bought** the supplies for the owner.
Verb complement clause	The manager revealed that the secretary **bought** the supplies for the owner.
Relative clause	The manager liked the secretary who **bought** the supplies for the owner.

[1] According to Marinis (2003), using at least seven syllables helps to ensure that the prime is no longer in active short-term/working memory.

Each sentence within a scenario had the same number of words following the prime, but they varied slightly in the number of words leading up to the prime. Matrix clauses had an average of 7 words before the prime, noun complements clauses had 8, verb complement clauses had 6, and relative clauses had 6. I return to the possible implications of these differences in section 5.

Each participant saw only one version of each of the scenario for a total of 32 experimental sentences per participant. Using a Latin square design, I divided the sentences such that each block contained one sentence per verb and equal numbers of each structural context per block.

3.2 Filler items

There were 128 filler sentences.[1] Each of these sentences was followed by a word verification task.[2] Eighty of the verification tasks were intended to elicited a 'no' response and 48 a 'yes' response, leading to equal numbers of intended 'no' and 'yes' responses over the course of the entire experiment when the experimental items were included. Of the filler items meant to elicit 'no' responses, half probed for nouns and half for verbs. Of the fillers items meant to elicit 'yes' responses, 32 probed for nouns and 16 probed for verbs. Sixteen of the 'yes'-nouns occurred in the first half of the sentence, and 16 occurred in the second half of the sentence. All of the 'yes'-verb fillers probed for the first verb of the sentence. Some of the verbs and nouns that occurred as probes were repeated as targets throughout the experiment. This was done to mask the repetition of the target dative verbs.

[1] Appendix 3B contains all the filler items used in this study.
[2] Note that all sentences, experiment and filler, were immediately followed by a word-verification task. The statement-verification task followed the word-verification task.

Fifty-four of the filler items were followed by a statement-verification task. This task was meant to mask the manipulation and help ensure that participants were attending to the meaning of the sentence and not simply memorizing the words. An example of one of the filler items with a statement-verification task is in (13) below.

(13) Filler item and statement-verification pair
The policeman liked the uniform, and the fireman loved the new red truck.
The policeman hated the uniform. *YES* *NO*

Approximately 42% of all the filler sentences were followed by both a word verification task and a statement verification task. In the statement verification task, participants heard the prime sentence and responded to the probe word as they normally would, and then they read a sentence and determined whether it was true ('yes') or false ('no') given the sentence they just heard. The proportion of 'yes'/'no' responses for the statement verification task was equal over the course of the experiment. These statement verification sentences probed for information occurring either in the first half or second half of the sentence equally.

The filler and experimental sentences, along with four training sentences were read by a female native-speaker of North American English. The recordings were spliced such that each sentence was preceded and followed by 200 msec of silence. For the experimental items, the same recording of the dative verb phrase (e.g. "bought the supplies for the owner") was used for each sentence version of each scenario. This was done to ensure that the amount of time between the onset of the dative verb for a particular scenario and the probe task was consistent across each sentence version for that scenario. The spliced recordings were checked for naturalness by

native speakers who were naïve of the manipulation.[1]

3.3 Method

The experimental and filler items were split into four blocks with three breaks between them. The items within a block were randomized; however, the presentation of blocks was not. This was done to ensure that the general facilitation that arises from repeated exposure to a word did not inadvertently affect participants' response time data. Thus, each participant saw Scenario 1 for *bought* in the first block and Scenario 2 for *bought* in the second block, and so on. Because only the comparison among the different sentence types of each scenario was relevant for the analysis, it was not necessary to compare different scenarios for the same verb, making counterbalancing of block presentation unnecessary.

Prior to beginning the experiment, participants read through instructions and used practice items to familiarize themselves with the task.[2] During the experiments, participants saw a cross-bar fixation point in the middle of the screen while listening to the sentence. Two-hundred msec after the off-set of the item, the cross-bars disappeared, and a word appeared in its place. Participants decided whether they had heard the word in the sentence or not. Following some of these probe tasks, participants saw a statement and determined whether it was true given the preceding sentence (statement-verification task), using the same yes/no keys. Figure 3.4 shows the presentation of two items, one filler and one experimental item.[3]

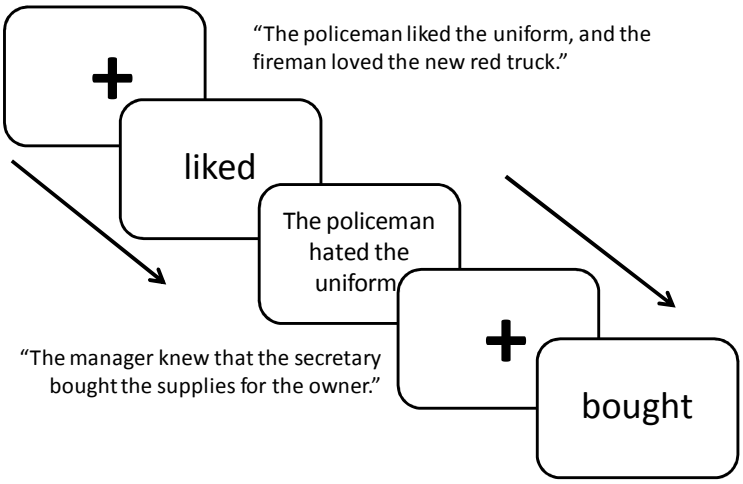
---

[1] Volunteers piloted the experiment, listening for inconsistencies and oddities such as volume changes, pops, or splices. Each volunteer heard the entire set of filler sentences but only one version of each of the experimental scenarios. Only two experimental items and three filler sentences were flagged for inconsistencies. These sentences were re-spliced or re-recorded depending on the problem.

[2] Instead of counterbalancing the presentation, the block ordering was entered into the regression as a fixed effect.

[3] Appendix 3C contains the instructions used to train the participants on this task.

Figure 3.4: Presentation of two items



3.4 Participants

Forty-four native-speakers of North American English from the Northwestern University community participated for pay or for partial fulfillment of course credit. Eight participants were excluded due to a high number of incorrect responses to the statement-verification and word-verification tasks and due to significantly deviant response behavior, i.e. they answered *TRUE* to all the stimuli, suggesting that they were not attending to the experiment. Data from the remaining 36 participants were used for the analysis. A total of 9 participants saw each version of each scenario.

3.5 Data preparation

For analyses of the response time data, only the times for correct responses to the target items were used for the response time analysis. A mean for each participant was calculated and used to trim each participant's data. Response times that were two standard deviations away from a participant's overall mean were removed (both those that were two deviations faster than

the average and two deviations slower than the average). These combined measures led to the exclusion of 6% of the total number of responses.

3.6 Review of SAP and PRICE predictions

The SAP account predicts that all primes should demonstrate the same effects on subsequent behavior. Thus, response times for the same word in the same scenario should not differ simply because the words occur in different structural contexts (e.g. a relative clause rather than a matrix clause). Because all the primes were controlled for linear position, they should each have similar response times during the identification task. Table 3.2, repeated below, demonstrates how linear position was controlled within a set of different sentence types for a version of a *bought*-scenario.

Table 3.2: Example of four versions of one scenario

Context	Sentence
Matrix clause	The manager left the request, and the secretary **bought** the supplies for the owner.
Noun complement clause	The manager reported the fact that the secretary **bought** the supplies for the owner.
Verb complement clause	The manager revealed that the secretary **bought** the supplies for the owner.
Relative Clause	The manager liked the secretary who **bought** the supplies for the owner.

The PRICE account claims that we should see more nuanced priming behavior than is expected from SAP. PRICE claims that priming effects are mediated by structural context. Specifically, the ability of the processor to reactivate a form at the target task depends on the features of the unification chain that the prime is associated with. Unification chains are formed during the initial processing of the prime sentence. The length of a given chain and the number

of chains that result from the processing of a given sentence depend on the sentence's syntactic structure. PRICE claims that primes associated with longer chains should demonstrate less priming than those associated with shorter chains. When time is held constant and only structural context is varied, PRICE claims that there should be differences in response times to primed forms. In contrast, SAP claims that there should be no differences among the primes, even when structural context is varied.

4. Results

Accuracy: Accuracy scores were calculated prior to the trimming[1] measure mentioned in section 3.5 above. Thus, all the data were used to determine the overall accuracy. In general, participants were highly accurate. Participants responded correctly to the primes in virtually all cases. Response accuracy for primes in main clauses, relative clauses, and verb complement clauses was about 99%. Response accuracy for primes in noun complement clauses was 97%. Due to the high level of accuracy and possible ceiling effects, these data were not explored in greater depth.

Response time: Only the data remaining after the trimming mentioned in section 3.5 were used for the analyses. The response time data were run through a linear mixed model logistic regression fit by REML (Baayen, Davidson, & Bates 2008) in which main clauses served as the baseline and in which block presentation was a fixed effect, and in which the random intercepts

[1] Recall that for the response time analysis only correct responses were used and that data over or under two standard deviations from a participant's average were excluded.

for both participants and verbs were included.[1] The data were treatment coded so that each level of the structural context was compared to the baseline (matrix clause structural context). Table 3.3 contains the results from this regression.[2]

Table 3.3: Results from linear mixed model regression

	Estimate	Std Error	t-value	p-value
Intercept	833.93	32.13	26.11	0.001***
Noun complement clause	34.27	14.56	2.36	0.02*
Verb complement clause	5.26	14.42	0.37	0.72
Relative clause	0.84	14.42	0.06	0.91
Block	-42.08	4.56	-9.38	0.001***

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

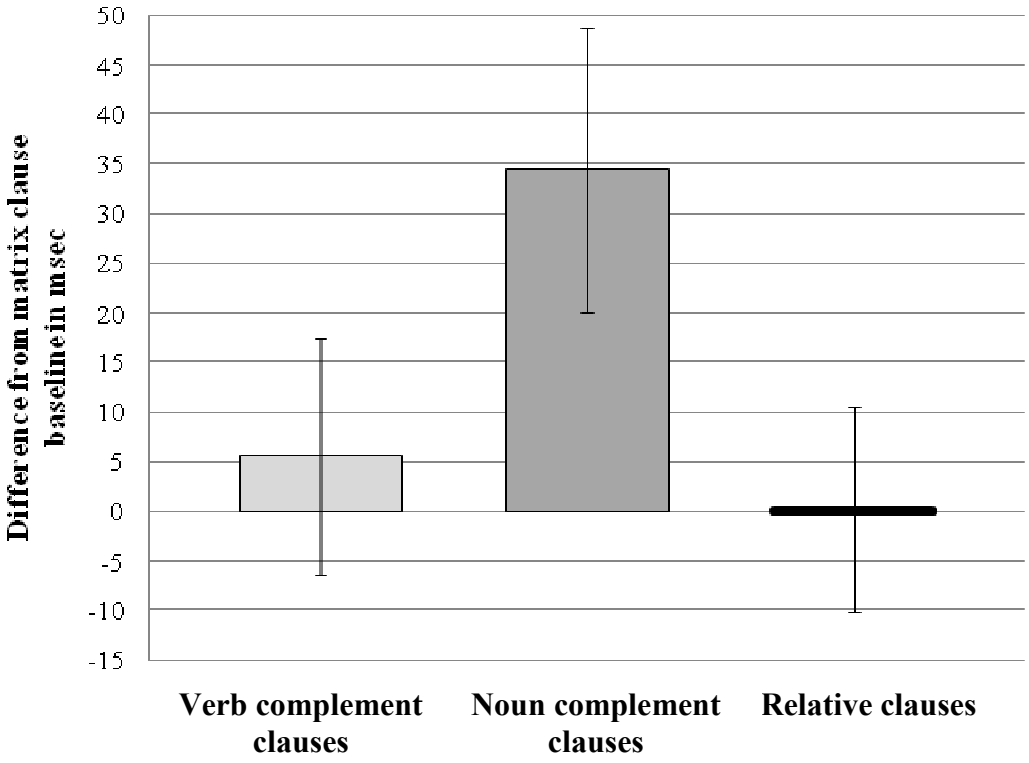
The regression found a significant difference between the baseline and noun complement clauses (N(36) t = 2.36, p < 0.02) but no difference between the baseline and verb complement clauses (N(36) t = 0.37, p = 0.72) or between the baseline and relative clauses (N(36) t = 0.06, p = 0.91). Participants were slower in responding to targets when the prime originally occurred in noun complement clauses (average: 761 msec; stdev:198 msec) as compared to when the prime occurred in matrix clauses (average: 726 msec; stdev:156 msec), relative clauses (average: 726 msec; stdev:173 msec), or verb complement clauses (average: 732 msec; stdev:168 msec). Post hoc t-test analyses further revealed that there was no significant difference between verb complement clauses and relative clauses (N(35) t = 0.41, p = 0.68) but that both differed from noun complement clauses (verb complement clauses N(35) t = 2.04, p < 0.05; relative clauses N(35) t = 2.22, p < 0.05).

[1] Along with allowing me to compare the results to a base line, the use of a mixed model regression allows me to control for both random and fixed effects. This helps to control for any noise due to random variation among participants or the prime and for the intended, fixed variation among the different blocks.

[2] MCMC was used to estimate p-values for each of the factors.

Figure 3.5 depicts these effects by showing the difference scores for the baseline (matrix clause primes) and the three other structural context types.

Figure 3.5: Difference scores between baseline and other clause types



As this figure suggests, primes in the scope of noun complement clauses were significantly different than any of the other primes relative to the baseline.

Before moving into the discussion, there is one possible source of the difference between noun complement clauses and other clause types that I wish to address and dismiss. There was a possible confound in the stimuli that could have led to proactive interference, situations in which previous processing interferes with current processing (Anderson & Neely 1996). Proactive interference can be illustrated by considering the ways in which older memories can inhibit the

application of newer memories. For example, one may have difficulty recalling where she parked her car at work today due to previous parking events in the same lot on prior days. Similarly, one may have difficulty recalling a more recently processed word due to the processing of earlier words. In the experiment described above, sentences with noun complement clauses had on average more words prior to the prime word than the other sentence types. This may have led to more processing difficulty at the point of the prime word for sentences with noun complement clause than for the other sentence types. The average number of words prior to a prime for relative clause sentences and verb complement clause sentences was 6, for matrix clause sentences 7, and for noun complement clause sentences 8. However, this difference is unlikely to be the source of the response time differences reported above. If proactive interference was, indeed, the source of the effect, we should have found the baseline to have been at least numerically slower—if not significantly slower—than relative clause sentences and verb complement clause sentences. The baseline sentences, i.e. those with two matrix clauses, also tended to have more words prior to the prime. If response time in this experiment were simply a matter of proactive interference due to the number of words preceding a prime, then the baseline primes should have led to response times falling somewhere between the noun complement clause sentences and the relative clause and verb complement clause sentences. However, they did not.

5. Discussion

The results from this experiment indicate that not all lexical primes are equal. Those that occur in

certain structural contexts demonstrate less priming than the same primes in different structural contexts. According to the model presented in Chapter 2, during the processing of sentences in this experiment, lexical items ('declarative chunks') are activated and held them in working memory (WM) until they are popped (released) from the buffer system and unified with another chunk. Recall that for all the experimental sentences, the prime word was held constant,[1] and its distance from the target, as measured by syllable length and stimulus onset asynchrony (SOA)[2] was held constant. The only factor that varied was the structural context in which the prime occurred. The results reported here conflict with the SAP account, which predicts that priming for specific lexical items should not be sensitive to the structural context in which the prime occurs. The SAP contends that only time matters, specifically, only the amount of time between the priming event and the trial matters. If time is held constant, features such as being in a noun complement clause or relative clause should be irrelevant. The above results do not support this prediction. Rather, they support PRICE.

In what follows, I first revisit PRICE along with the model of language processing presented in Chapter 2. After this review, I walk through examples of how language processing proceeds in the various structural contexts explored in the current study. This demonstration focuses on how chunks are unified and the outcome of these unifications. In the final section of the discussion, I reflect on how unification cycles affect priming behavior.

---

[1] To be specific, there were eight total prime words, and each of these prime words was compared to the same prime word within a given block across subjects. Thus, the prime *bought* in a relative clause in Block 1 was compared to the prime *bought* in a verb complement clause in the same block.

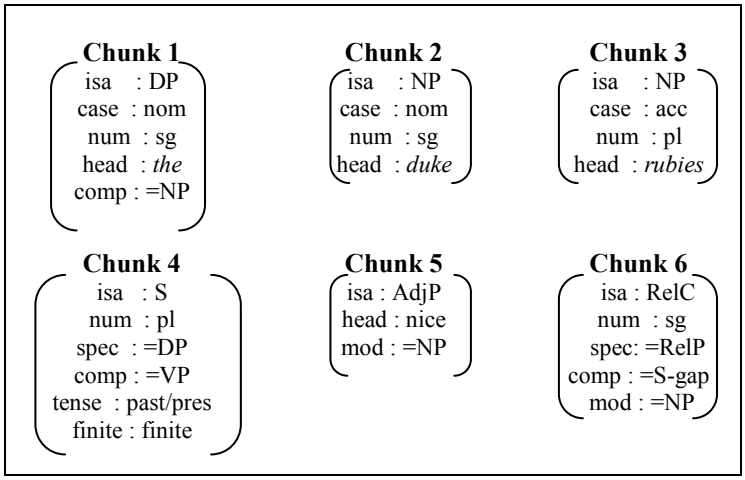
[2] Stimulus Onset Asynchrony (SOA) refers to the amount of time (e.g. msec) between the end of one event and the onset of another event. In the current experiment, the SOA refers to the end of the sentence and the onset of the probe word verification task.

5.1 PRICE and language processing

According to PRICE, the structural context in which a prime occurs affects its subsequent effect on linguistic behavior. RICE states that the reason for structural context differences stems from how various structural contexts are processed and subsequently represented in memory. According to the model of language processing presented in Chapter 2, memory holds traces of each priming sentence, but the nature of these traces (e.g. their size) is sensitive to how the sentences were processed. Some traces allow for greater access to the linguistic units that comprised the priming sentence than others. Features such as the size of a memory for a priming sentence (i.e. memory for a specific sentence) limit subsequent access for linguistic forms within the prime sentence.

In the model of language memory presented in Chapter 2, sentence processing is treated as a series of coordinated problem-solving subgoals used to satisfy a main goal (e.g. 'process sentence'). As the processor works to achieve this goal, subgoals are added to the problem state depending on the nature of the chunks in the retrieval buffer or lack thereof (Chapter 2, section 3.3.1). To determine whether a chunk (and, hence, a new subgoal) enters into the problem state buffer, the processor determines whether there are any unresolved (open) values in the chunk's feature-value pairs (Chapter 2, section 3.3.1).

Recall that chunks can have open values, such as the open NP value (=NP) in the DP chunk below and the open =DP and =VP in Chunk 4:



When the processor retrieves the DP chunk, it places it in the retrieval buffer. On the next cycle, when the processor checks the status of the problem state buffer and the retrieval buffer, it notices the open =NP value in the DP-chunk, so it pushes the DP-chunk into the problem state buffer, making the resolution of the open value a new subgoal (Chapter 2, section 3.3.1). During language processing, the processor generates a series of subgoals in a stack-like fashion. Each new subgoal is placed on top of the stack, and no preceding subgoal can be resolved until the most-recently generated one has been resolved. Each of these subgoals is a consequence of features of retrieved chunks. For example, say the processor needs to process the sentence "the duke loves the rubies." In order to begin processing this sentence, the processor must retrieve an S-chunk (e.g. Chunk 4), which has two open values (=DP and =VP). Because it has open values, the S-chunk is placed into the problem state buffer, generating two subgoals: 'process DP' and 'process VP.' Neither of these two subgoals is resolved until each of the phrases has been completely processed.

In the demonstration below, I show only the processing of the subject DP of a sentence

such as "The duke promised the duchess the rubies." This is not meant to suggest that processing must be incremental, with the subject always being processed before the predicate. Although in comprehension the subject is processed first due to the linear nature of the speech stream, the current model is agnostic about which is processed first: the subject DP or predicate VP. The model of language processing I am assuming allows for the verb to be processed first, as in verb-centered processing models.[1] Furthermore, my model does not address when other processes, such as phonological encoding, take place. I restrict the scope of the current discussion strictly to syntactic processing, with a current focus on comprehension because of the design of the experiment described above.

The processor begins sentence comprehension with the goal 'process sentence.' This goal leads to the retrieval of an S-chunk. The S-chunk has two open values: =DP and =VP. The processor begins with the =DP ('process DP') subgoal.

	Subgoal stack
⇩	'process DP' 'process S'

The processor retrieves a DP-chunk (e.g. Chunk 1). However, this chunk has an open value: =NP. The DP-chunk is placed in the problem state, generating a new subgoal: 'process NP.' The subgoal 'process S' cannot be resolved until the other two subgoals have been resolved.

	Subgoal stack
⇩	'process NP' 'process DP' 'process S'

[1] For discussion of incremental processing and the centrality of the verb, see Bock & Levelt 1994; Ferreira 2000; Ferreira & Swets 2002; Ferreira & Slevc 2007; Ford 1982; Garrett 1982; Keller 2009; Tanenhaus, Spivey-Knowlton, Eberhard, & Sedivy 1995.

The processor then retrieves the NP-*duke*-chunk (Chunk 2) and places it in the retrieval buffer. This chunk has no open values, so it does not go into the problem state buffer. Rather, it is popped. This popping resolves the subgoal 'process NP,' thereby allowing the processor to move on to resolving the next subgoal: 'process DP.'

	Subgoal stack	**Resolved subgoals**
⇩	'process DP' 'process S'	*'process NP'*

Once the values of the popped NP are unified with the open values of the DP-chunks =NP (see Chapter 2, section 3.3), the open values of the DP-chunk's =NP are resolved, leading to the completion of the 'process DP' subgoal:

	Subgoal stack	**Resolved subgoals**
⇩	'process S'	*'process DP'* *'process NP'*

Because the 'process DP' subgoal is resolved, the processor can return to the next subgoal: 'process S.'

This demonstration depicts a line of subgoal processing in which the product of one unification becomes input for the next subgoal. For example, the unification cycle that unified the NP-chunk's values and the open =NP value in the DP-chunk led to the popping of a more fully specified DP-chunk. This DP-chunk could then unify with the open =DP value in the S-chunk, which was the next subgoal in the subgoal stack. Because each unification cycle generated a form that could be unified with an open value in the next subgoal, together the

unification cycles form a unification chain. However, there are times when the product of a unification cycle cannot unify with an open value in the next subgoal.

For example, consider the processing of the phrase ‘the nice duke’ (see also Chapter 2, section 3.4.3). We begin with the subgoal ‘process NP.’ This subgoal is always generated whenever a DP-chunk with an open =NP value is pushed into the problem state buffer.

	Subgoal stack
⇩	‘process NP’ ‘process DP’ ‘process S’

Next, the processor encounters an AdjP chunk: Chunk 5 (*nice*). Because the AdjP-chunk has an open value (=NP) in its ‘mod’ feature, the chunk is pushed into the problem state buffer, generating another ‘process NP’ subgoal.

	Subgoal stack
⇩	‘process NP’ ‘process NP’ ‘process DP’ ‘process S’

Now, the processor processes the NP-*duke*-chunk. The chunk has no open values, so it is popped.

	Subgoal stack	**Resolved subgoals**
⇩	‘process NP’ ‘process DP’ ‘process S’	*‘process NP’*

Because it matches the open value (=NP) in the AdjP-chunk’s ‘mod’ feature, the popped NP unifies with the open =NP value of the AdjP-chunk. The AdjP-chunk is popped and becomes available for subsequent unification.

	Subgoal stack	Resolved subgoals
⇩	'process NP' 'process DP' 'process S'	*'process NP'*

However, the popped AdjP-*nice*-chunk does not satisfy the current subgoal 'process NP.' Thus, the AdjP-chunk goes directly to long-term memory (LTM). Recall that just because the syntactic processing is complete, other levels of processing (e.g. semantic) may still be active (Allen & Badecker 1999, 2000; Dell 1986; Roelofs 1992, 1993 *inter alia*, see also Chapter section 3.1). In other words, the syntactic aspects of processing the phrase 'nice duke' are complete. The AdjP has been formed. However, the phonological encoding of the phrase may still be underway and the semantic referent of the phrase may still be active. As such, different levels of processing for the phrase 'nice duke' may still be in progress.

The important difference to note between the processing of the phrase 'the duke' and the phrase 'the nice duke' is that in the first, each popped ('processed') chunk satisfies an open value in the top-most chunk in the problem state buffer, whereas in the second example ('the nice duke'), some chunks (i.e. the AdjP-chunk) encountered do not. Recall that these open values act as subgoals that create a stack-like structure. As the processor works through the stack, it satisfies the open values in the chunks. In the second example ('the nice duke'), there is not an open value for the AdjP-chunk and, hence, no subgoal that it could satisfy. Thus, the popped AdjP-chunk cannot modify the subgoal structure, unlike the popped NP-chunk and popped DP-chunk which can.

In Chapter 2, section 3.4.3, I argued that whether a popped chunk modifies the subgoal structure (i.e. satisfies an open value) has implications for the way memory traces are generated

and subsequently recalled. Each time a popped chunk is successfully unified with an open value of another chunk, there is a unification cycle. Each cycle can constitute a link in a chain of unifications ('unification chain'). If the product of a unification cycle feeds directly into the next subgoal, the unification chain grows longer. The chain does not grow if the product does not satisfy a subgoal.

One consequence of this process of linking unification cycles into chains is that the processing of arguments and adjuncts differs. Arguments always lead to the satisfaction of the next subgoal in the subgoal stack, e.g. processing an NP satisfies the subgoal 'process NP' associated with the processing of a DP-chunk. However, adjuncts never lead to the satisfaction of a subgoal on the subgoal stack. Adjuncts chunks (e.g. the AdjP-chunk Chunk 5 and the RelC-chunk Chunk 6) place restrictions on the types of chunks they modify via their 'mod' feature. However, by definition, no chunk selects for an adjunct chunk such as the AdjP-chunk. Consequently, adjuncts cannot be part of the same unification chain as the chunks preceding it. Rather, an adjunct chunk, such as the one above, is associated with a separate unification chain that results from unifying an NP-chunk and the 'mod' value of an AdjP-chunk. Thus, the phrase 'the duke' is associated with only one unification chain, whereas the phrase 'the nice duke' is associated with two (see also Chapter 2, section 3.4.3):

Unification chain for "the duke"
unify pop-NP with =NP in DP *unify pop-DP with =DP in S*

Unification chains for "the nice duke"	
unify pop-NP with =NP in AdjP	*unify pop-NP with =NP in DP* *unify pop-DP with =DP in S*

These unification chains act as a form of bookkeeping. The chunks and rules associated with each chain are ultimately represented together in memory. As discussed in section 2.1 above, the features of a sentence's chain(s) affects subsequent behavior. For example, the length of a given chain affects the ability of the processor to locate specific chunks associated with the chain. During subsequent processing, a prime's unification chain, which reflects the structural context in which the prime was processed, affects how accessible the prime is. This accessibility, in turn, affects the likelihood that the prime can affect subsequent performance.

I now turn to specific examples of how the different sentence types are processed given the model of processing presented in Chapter 2. In doing so, I demonstrate how the sentences differ in terms of the number and length of the unification chains that they are associated with. These differences correlate with performance differences, i.e. primes that are associated with longer unification chains demonstrate less priming than those associated with shorter chains, even when time is held constant.

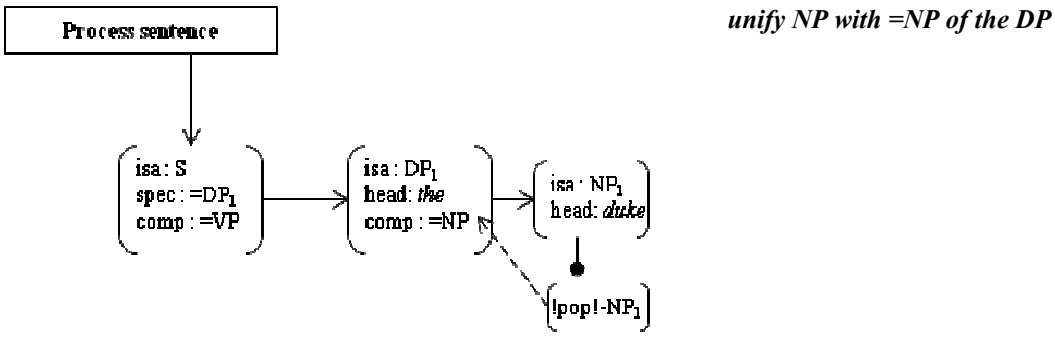
5.2 Processing chunks in different structural contexts

We begin with a sentence in which the prime word (i.e the bolded *promise*) occurs in a matrix clause.

(14) Prime in matrix clause
The duke **promised** the duchess the rubies.

For the following demonstration, I present the history of chunk retrievals, unifications, and formation of unification chains from the perspective of sentence comprehension. This means that when the processor 'processes' a word, the processor has comprehended the word and has automatically retrieved the associated chunk and placed it in the retrieval buffer. However, I

attempt to be as general as possible, such that the history of events could apply to either sentence comprehension or production. In this and all subsequent demonstrations, the processing goals are shown in a box, whereas all the retrieved chunks appear in brackets. When I discuss the processing of adjuncts, I introduce additional notation. A solid arrow (→) denotes the application of a production rule that retrieves a chunk. A terminal button (↓) denotes the application of a production rule that pops chunks. A dashed arrow (↖) represents unification. For example, the diagram below shows the series of steps used to retrieve an S-chunk, then a DP-chunk (→), then an NP-chunk (→), followed by the popping (↓) of the NP, and its unification with the open =NP value of the DP-chunk (↖).



The right-hand column keeps track of all the unification cycles that have occurred during the processing event. The numbering of the nouns and verbs is strictly for the purpose of describing the examples. For instance, the NP that is unified with the open =NP value of the DP-chunk is labeled as "NP_1." Following each demonstration is a table that displays all of the chunks retrieved generated during the sentence's processing. The tables take a form such as the one given below:

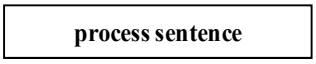
Retrieved chunks	Unification cycles

S-chunk **DP-*the*-chunk** **NP-*duke*-chunk**	*unify* NP_1 *with* $=NP_1$ *of* DP_1 *unify* DP_1 *with* $=DP_1$ *of S* *unify* NP_2 *with* $=NP_2$ *of* DP_2

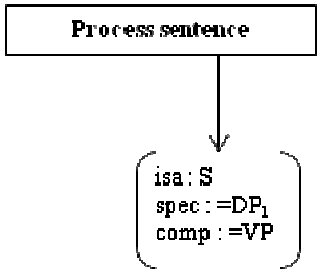
I first illustrate the processing of (14):

(14) Prime in matrix clause
The duke promised the duchess the rubies.

The processor begins with a goal, i.e. 'process sentence':

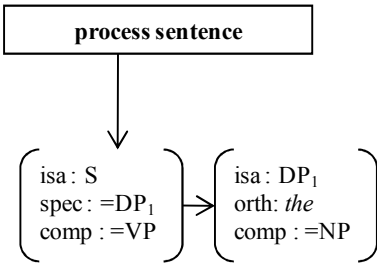


Once this goal is set in the control state buffer, the system retrieves the basic sentence frame using a production rule ('retrieve S-chunk,' see Appendix 2A for a complete list) which places an S-chunk into the retrieval buffer:

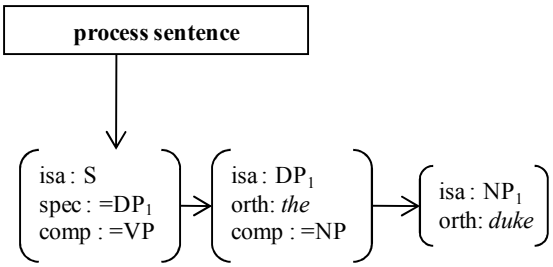


The S-chunk has two open values: =DP and =VP. These open values serve as placeholders for values to be provided by other chunks. Because there are open values, the processor determines that the chunk is incomplete and that its values must be satisfied before the goal is satisfied. Thus, the processor selects a rule that pushes the S-chunk into the problem state buffer (see Chapter 2, section 3.3.1). At this point, there are two subgoals: 'process DP' and 'process VP.' The processor could choose to begin work on either of them first, but for our purposes, I stipulate that the processor begins with the 'process DP' subgoal.

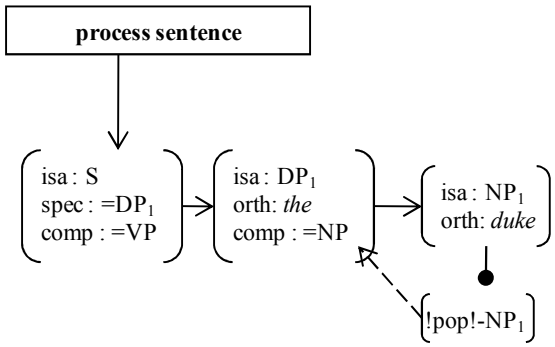
Upon the processing of the word *the*, a new chunk is retrieved (i.e. DP-*the*-chunk), which contains its own open value (i.e. =NP). Because of this open value, the processor chooses a 'push' production rule that forces the DP-chunk into the problem state buffer, thereby leading to a new subgoal ('process NP').



Now there are two chunks associated with the goal of comprehending the sentence, both of which contain open values and, thus, are part of the problem state. The most recently added subgoal is the one associated with the open value of the DP, namely the 'process NP' subgoal. This subgoal must be resolved before the processor can resolve the subgoals associated with the S-chunk. After processing *duke,* the processor retrieves an NP-chunk from memory (NP-*duke*-chunk) and places it in the retrieval buffer.

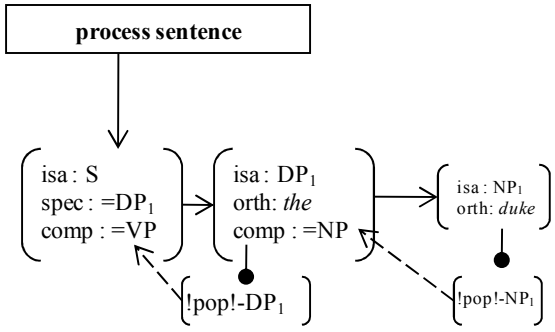


Because this chunk has no open values, it is popped from the retrieval buffer, thereby becoming available for unification. Unification proceeds and the popped NP's values unify with the =NP in the DP-chunk:



Because the popped NP-chunk unified with the open =NP value of the DP-chunk, there is now one unification cycle listed above.

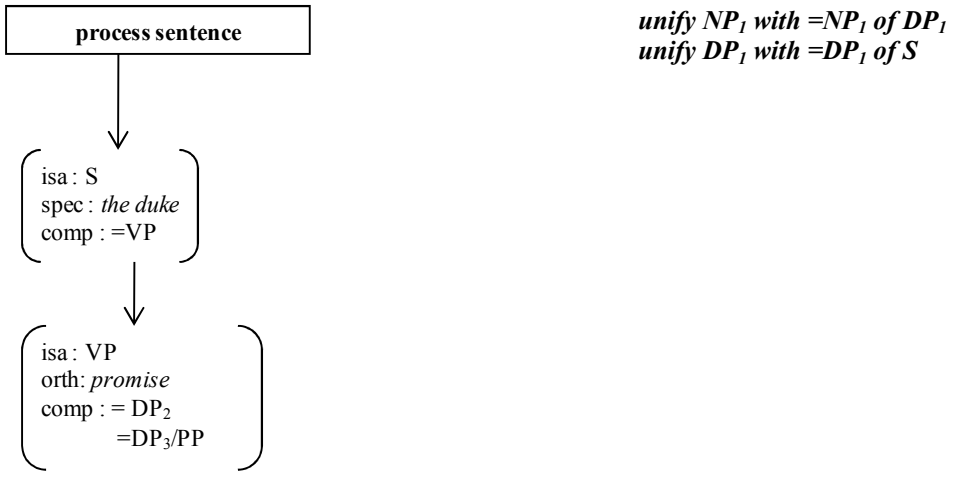
Now that the values of the DP-chunk are filled, it too is popped. The DP-chunk's values unify with the open =DP value in the S-chunk. This unification also counts as a unification cycle.



Because the product of the two unification cycles (e.g. NP and =NP, and DP and =DP) directly lead to the resolution of the next subgoal in the subgoal stack 'process DP' of the S-chunk, the two unification cycles are linked together in the same unification chain. One thing to note in the depictions above is that once a chunk is popped, the appearance of the chunk changes, i.e. the font size decreases. This is meant to indicate the onset of decay. Once a form is popped (e.g. NP_1), its individual activation begins to wane. For ease of presentation, I do not show chunks

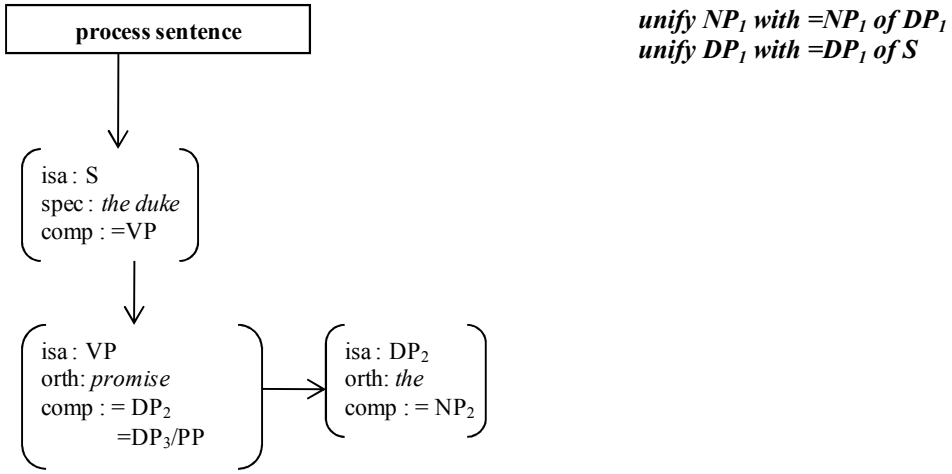
that have been popped and unified from this point forward. After a chunk has been popped, it is no longer in use and, hence, its activation boost begins to wane.

The S-chunk still has one open value (=VP), so the S-chunk remains in the problem state. Upon comprehending *promise*, the processor retrieves the VP-*promise*-chunk and places it in the retrieval buffer. The slash between the two forms in the second 'comp' position (i.e. DP/PP) indicates that the post-verbal complement of *promise* can take two forms: "promise the duchess the rubies" (NP,NP), or "promise the rubies to the duchess" (NP,PP). The alternation between these two patterns is called DATIVE ALTERNATION, which refers to the variable ordering of arguments following dative verbs such as, *promise, show,* and *give* (e.g. Bresnan 2007; Bresnan, Cueni, Nikitina, & Baayen 2007; Bresnan & Nikitina 2009; Doyle & Levy 2008; Green 1974; Oehrle 1976). In the dative object order (NP,NP), the recipient/benefactor ("the duchess") precedes the patient ("the rubies"). In the prepositional dative order (NP,PP), the patient precedes the recipient/benefactor.

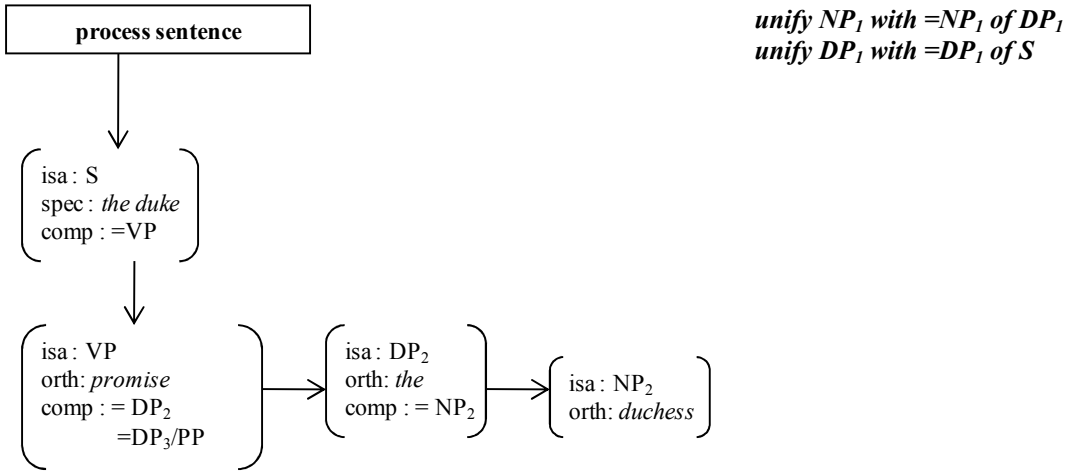


The VP-chunk has multiple open values, so it too is placed into the problem state buffer. Each of

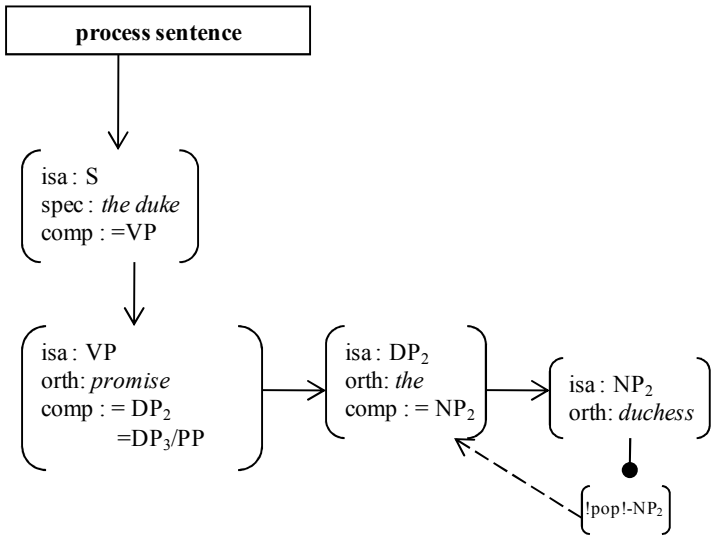
its open values entails the creation of a new subgoal. Upon hearing *the,* the processor again retrieves the DP-*the*-chunk. This chunk also has an open value, so it too is placed in the problem state buffer until its values are filled.



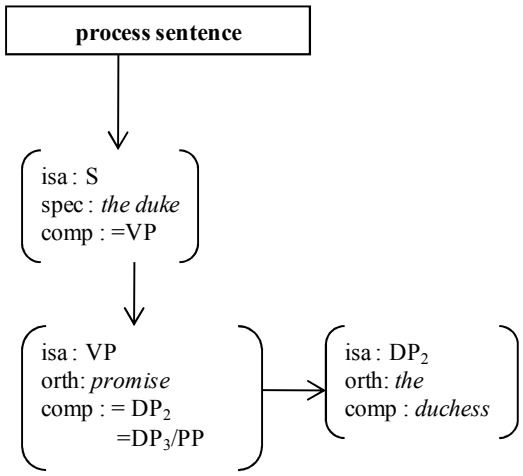
Then the word *duchess* is comprehended, leading to the retrieval of the NP-*duchess*-chunk.



There are no open values in this chunk, so it is popped and unified with the DP-*the*-chunk. Because the NP-*duchess*-chunk substitutes its values for the open =NP value in the DP-*the*-chunk, it counts as a unification cycle.

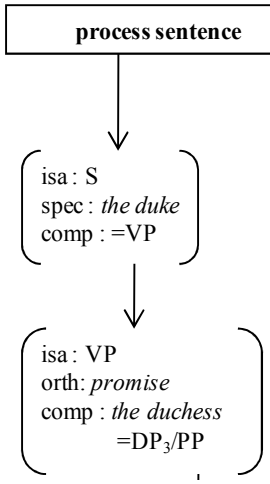


This unification resolves the open =NP value in the DP, meaning that the DP is now complete and the subgoal 'process NP' is complete. The DP is popped from the retrieval buffer, becoming available for unification with the chunk associated with the next subgoal, i.e. the VP-chunk. The DP-chunk and the open =DP value of the VP-chunk are unified, counting as another unification cycle. The output of this unification cycle satisfies an open value in another chunk (i.e. the VP-chunk). Thus, it affects the subgoal structure and becomes a link in the unification chain.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP

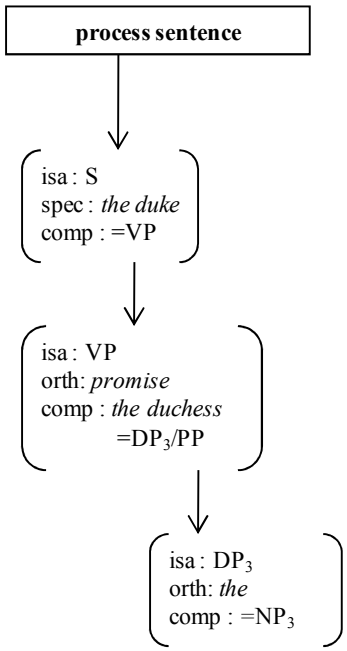
The first argument of the VP-*promise*-chunk is now filled.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP

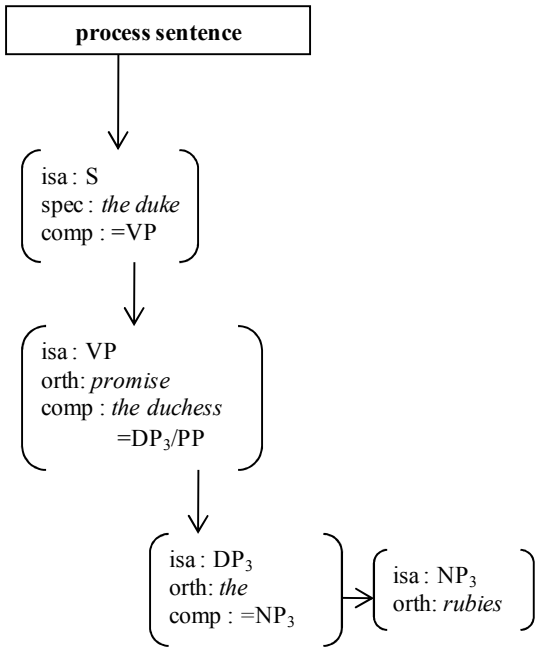
However, there is still another open value, so the VP-*promise*-chunk remains in the problem state buffer.

After processing *the,* the processor retrieves the DP-*the*-chunk and places it in the retrieval buffer.

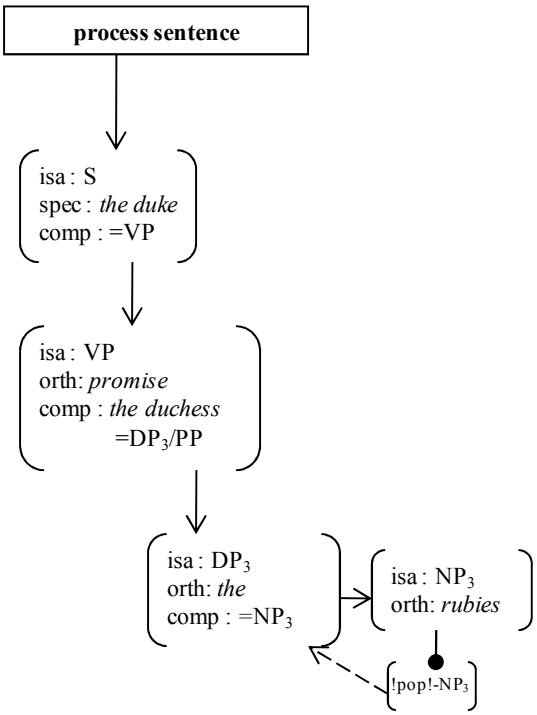


unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP

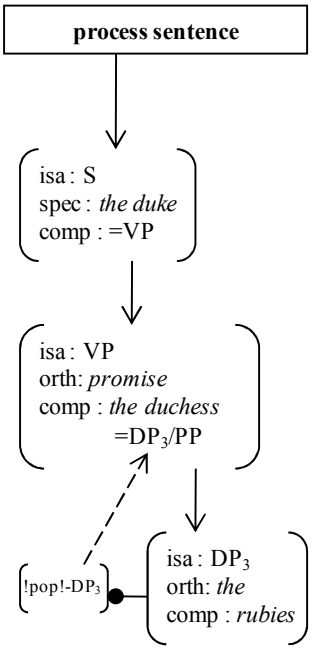
The same process of retrieving, popping, and unifying that we saw for the previous DPs continues with the processing of this DP:



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP



unify NP_1 with =NP_1 of DP_1
unify DP_1 with =DP_1 of S
unify NP_2 with =NP_2 of DP_2
unify DP_2 with =DP_2 of VP
unify NP_3 with =NP_3 of DP_3

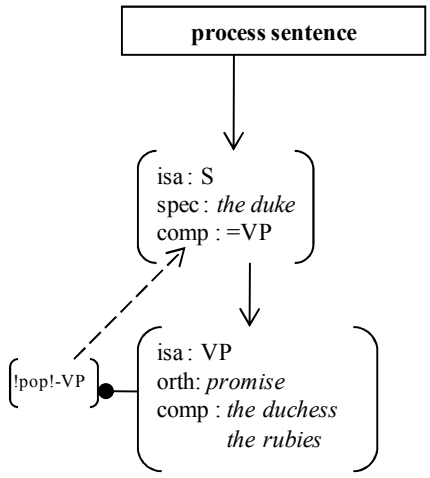


unify NP_1 with =NP_1 of DP_1
unify DP_1 with =DP_1 of S
unify NP_2 with =NP_2 of DP_2
unify DP_2 with =DP_2 of VP
unify NP_3 with =NP_3 of DP_3
unify DP_3 with =DP_3 of VP

The product of each of these unification cycles can unify with open values of the chunk

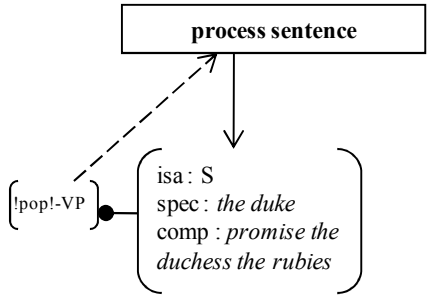
associated with the next subgoal. As such, they occur in the same unification chain.

Now that all the open values for the VP-*promise*-chunk are filled, the VP chunk is popped. It unifies with the =VP in the S-chunk. This unification cycle adds another link to the unification chain that formed during the processing of the VP predicate for the S-chunk.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of VP
unify VP with =VP of S

Now that the S-chunk's values are all satisfied, it is popped. There are no additional subgoals in the problem state. The main goal is now complete, and the sentence "The duke promised the duchess the rubies" enters into memory as one unit.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of VP
unify VP with =VP of S
unify S with control state

In this example, the prime word *promise* occurs in the matrix clause as the main verb. It is associated with the unification chain generated by the processing of "the duke promised the duchess the rubies." Figure 3.6 presents a summary of all the processing steps (chunk retrievals,

poppings, and unifications) with the prime word circled. This figure displays each of the components involved in the processing of the matrix clause discussed above ("The duke promised the duchess the rubies").[1]

Figure 3.6: Retrieval of chunks and rules for processing a matrix clause

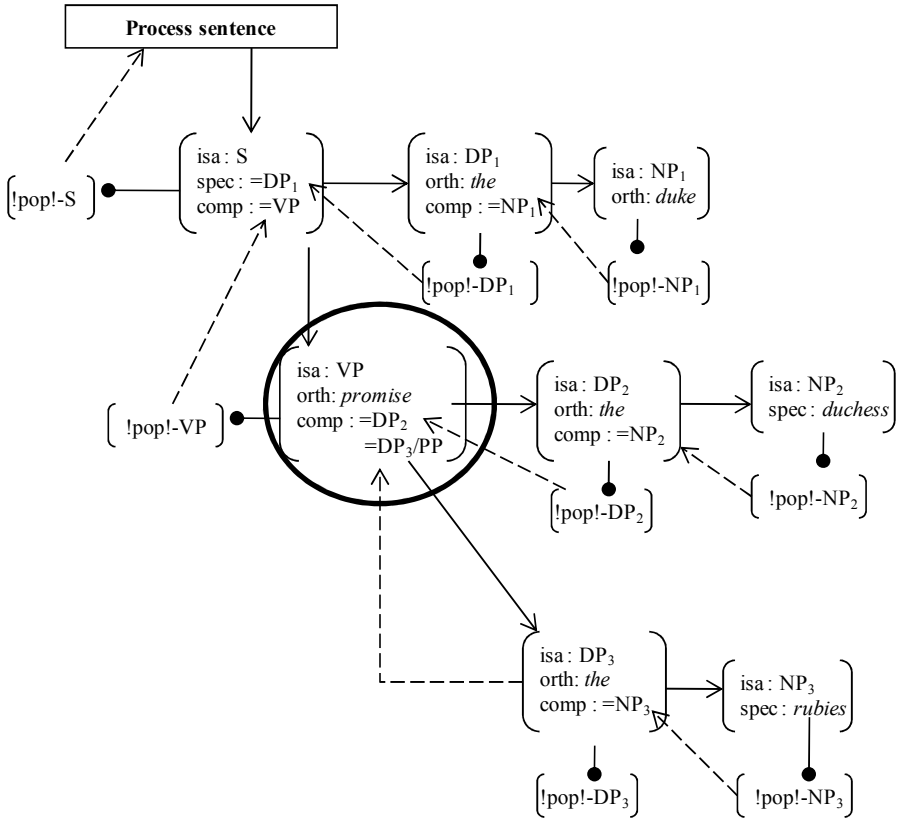


Table 3.4 below contains a full list of all the unification cycles that occurred in the processing of this sentence. The unifications are listed in the order in which they occurred. The chunks associated with the processing of the matrix clause "The duke promised the duchess the rubies" are on the left.

[1] Figures for each of the sentence types, similar to this one, are included in Appendix 3D.

Table 3.4: Outline of the unification chain with cycles and associated chunks

Retrieved chunks	Unification cycles
S-chunk	*unify* NP_1 *with* $=NP_1$ *of* DP_1
DP-*the*-chunk	*unify* DP_1 *with* $=DP_1$ *of S*
NP-*duke*-chunk	*unify* NP_2 *with* $=NP_2$ *of* DP_2
VP-*promise*-chunk	*unify* DP_2 *with* $=DP_2$ *of VP*
DP-*the*-chunk	*unify* NP_3 *with* $=NP_3$ *of* DP_3
NP-*duchess*-chunk	*unify* DP_3 *with* $=DP_3$ *of VP*
DP-*the*-chunk	*unify VP with =VP of S*
NP-*rubies*-chunk	*unify S with control state*

For now, the relevant column is the left-hand column, which contains the chunks associated with the unification chain in which the prime word, *promise,* occurs. There are eight chunks in the set of chunks that were used during the processing of the matrix clause "The duke promised the duchess the rubies." This number is relevant because the number of chunks associated with a given context affects the activation weight of the primed chunk.

In Chapter 2, section 3.3.2, I argued that chunk retrieval was sensitive to a chunk's activation. The higher a chunk's activation, the more like it is to be retrieved. This activation can be estimated using the total activation weight formula repeated below:

$$A_i = B_i + \sum_j w_j S_{ji} \qquad \text{Total activation weight}$$

In this equation, we see that activation is a combination of the base activation weight *Bi*, which is calculated using the formula below, and other factors, which I return to later:

$$Bi = \ln \sum_{j=1}^{n} tj^{-d}$$ (Base) Activation weight

The important aspects of this formula for our current purposes are *j*, which reflects the amount of time *t* since its most recent retrieval *j*, and *d*, which is the constant rate of memory decay. Combined, these factors reflect the recency effect. Recently processed forms have experienced less decay than those that were processed less recently, and as such should have slightly higher activations.

The base activation weight feeds into the total activation weight equation above. The total activation weight reflects the base activation weight and other factors that can affect the activation weight of a given chunk. In Chapter 2, section 3.3.2, I contend that the number of elements *j* associated with a goal and their weighted strength *Wj* can affect the activation weight of a chunk. Specifically, I argued that the more chunks associated with a goal, the less activation there is for each particular chunk. The reason that the number of chunks is relevant is that *Wj* is not free but is determined by the formula *G*/*j*, where *j* is the number of goal features (e.g. chunks) and *G* the amount of goal activation. The amount of cognitive resources is constant and must be shared among the chunks for a given goal. As the number of chunks increases, the amount of goal activation each chunk receives decreases. In the processing of the sentence "The duke promised the duchess the rubies," the processor retrieves and unifies 8 chunks, so the goal activation is shared equally among these 8 chunks (0.13 each).

In the experiment described in the previous section, when the processor tries to determine whether a specific word occurred in a previous sentence, the processor retrieves the unification

chains generated during the processing of a sentence.[1] Prime chunks that are associated with chains with fewer chunks than those associated with chains with more chunks have a greater portion of the cognitive resources associated with the goal. This leads to the prediction that chunks that are associated with shorter unification chains should affect priming more than those that are associated with longer chains when time and the effects of recency (tj^{-d}) are held constant.

Using the number of chunks in the matrix sentence as a baseline, we now turn to the processing of the other sentence types examined in the experiment discussed in the previous section to determine whether differences in the unification chains and their associated chunks correlate with the response time data. I contend that arguments are processed with their selectors. The processing of the selector (e.g. a DP-chunk, 'the') entails the processing of its argument (e.g. an NP-chunk, 'duke') to generate a grammatical phrase or clause (e.g. "the duke"). Because of this, the argument and its selector are inextricably linked, and this linking is captured by the unification chain that joins them. However, adjuncts by definition are not selected by any chunk. Thus, they are not linked to other chunks other than those necessary for the processing of the adjunct itself. For example, all the chunks necessary for the processing of the relative clause "who like pumpkins" plus the chunk it modifies "girl" are associated with one unification cycle "girl who likes pumpkins").

I propose that the fact that argument clauses (e.g. the complement clause "that the girl likes pumpkins") necessarily are associated with longer unification chains than adjunct clauses

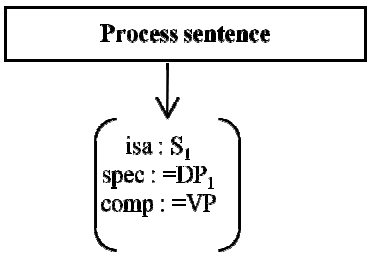
[1] Recall that in the experiment, participants were asked to determine whether a currently displayed word (e.g. *bought*) occurred in the sentence they previously heard (e.g. "the secretary bought the supplies for the owner"), so they need to reactivate their memory for the previous sentence.

(e.g. the relative clause and the form it modifies "girl who likes pumpkins") because argument clauses are selected by their heads (e.g. "Gordon knows that the girl likes pumpkins") and are, therefore, associated with the same unification chain. I further claim that differences in the length of these chains ultimately affects priming from inside argument and adjunct clauses.

To test this claim, I now illustrate how sentences containing noun complement clauses are processed in the model presented in Chapter 2. As discussed in the previous section, in this sentence type, the prime word occurred in the same linear position as it did in matrix clauses such as "the duke promised the duchess the rubies." Specifically, the prime word always occurred the same number of syllables and seconds away from the target task. Consider the following sentence, which contains the prime (bolded) within a noun complement clause (bracketed):

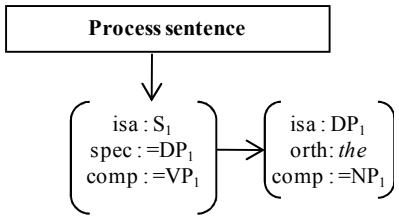
(15) Prime in noun complement clause
The report declared the fact [that the duke **promised** the duchess the rubies].

As with the matrix clause above, we start with a main goal and primary S-chunk with open values.

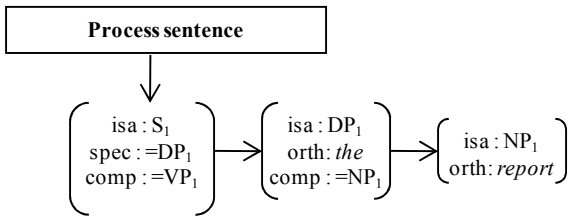


Upon the processing of *the*, the relevant chunk is retrieved (i.e. DP-*the*-chunk) and the process of

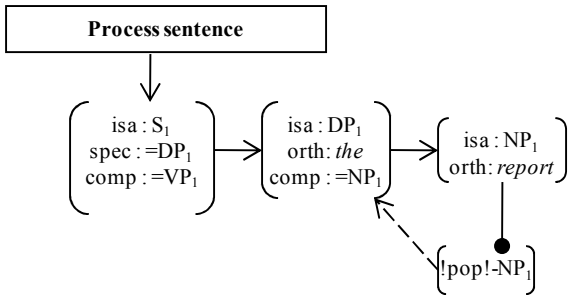
building a subject proceeds.



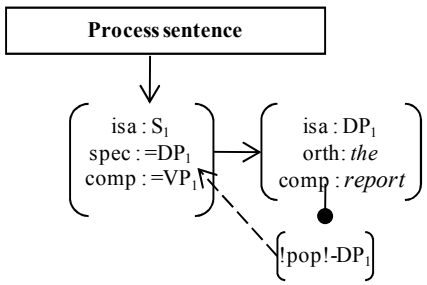
After the processor encounters the word *report,* a NP-*report*-chunk is placed in the retrieval buffer.



This NP has no open values, so the process of popping and unifying elements of the subject DP proceeds in the same manner as in the previous demonstration:

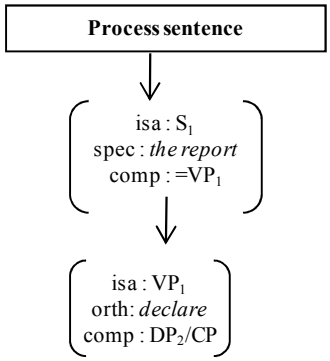


unify NP$_1$ with =NP$_1$ of DP$_1$



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

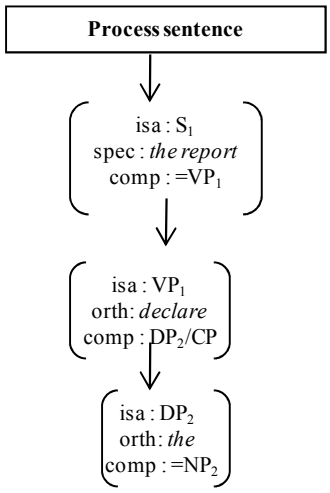
Two unification cycles have occurred, one unifying the NP-chunk and open =NP value of the DP-chunk, one unifying the DP-chunk and the open =DP value of the S-chunk. The product of each unification cycle resolved open values in the chunk associated with the problem state, so the cycles are part of the same unification chain. The open =DP value of the S-chunk is now filled, and the processor is ready for the next element. Once the processor encounters the verb *declared*, it retrieves the VP-*declare*-chunk and places it in the retrieval buffer.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

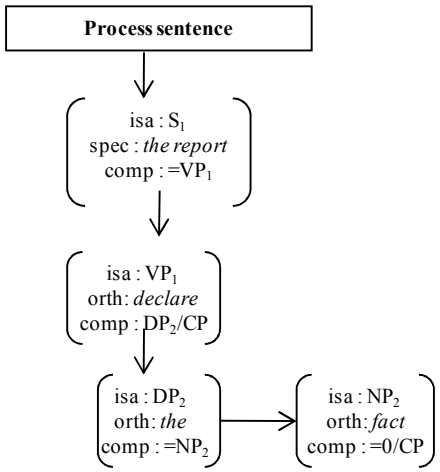
The *declare* chunk requires either a DP or CP argument as denoted by the =DP/CP,[1] so it cannot be popped. It is added to the problem state with a subgoal set for processing its open =DP/CP value. Upon hearing *the,* the processor again retrieves the DP-*the*-chunk and begins work to fill its open values.

[1] See Grimshaw (1979) for a detailed review of complement-taking predicates such as the ones used in the current work.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

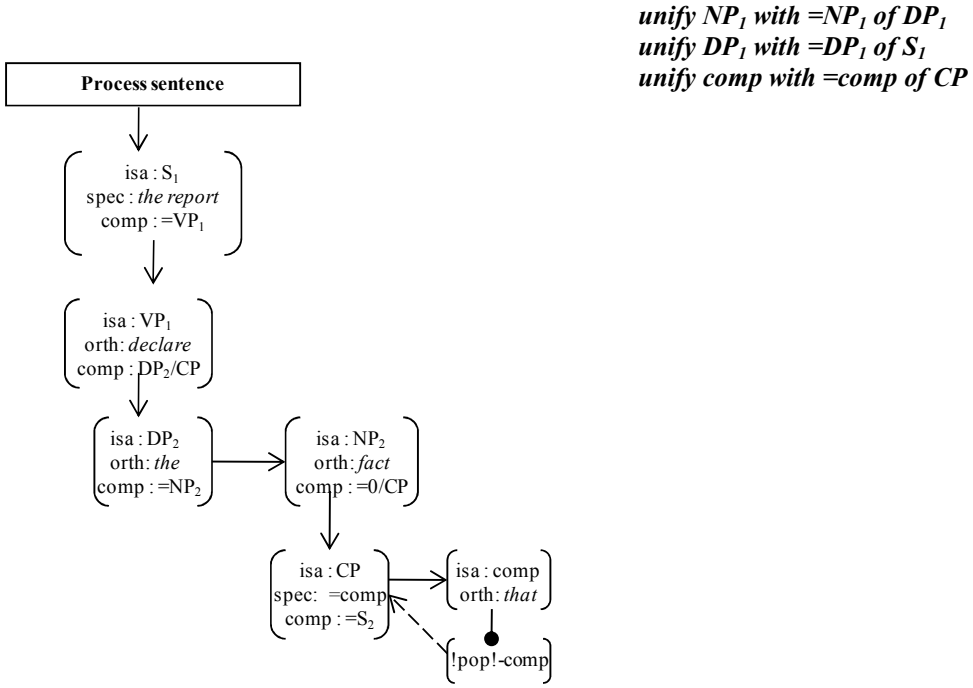
The word *fact* is then processed, leading to the retrieval of the NP-*fact*-chunk. This chunk can optionally take a complement clause as an argument. I represent the optionality of the CP argument by using the value =0/CP. This notation is meant to reflect that *fact* optionally takes a CP argument.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

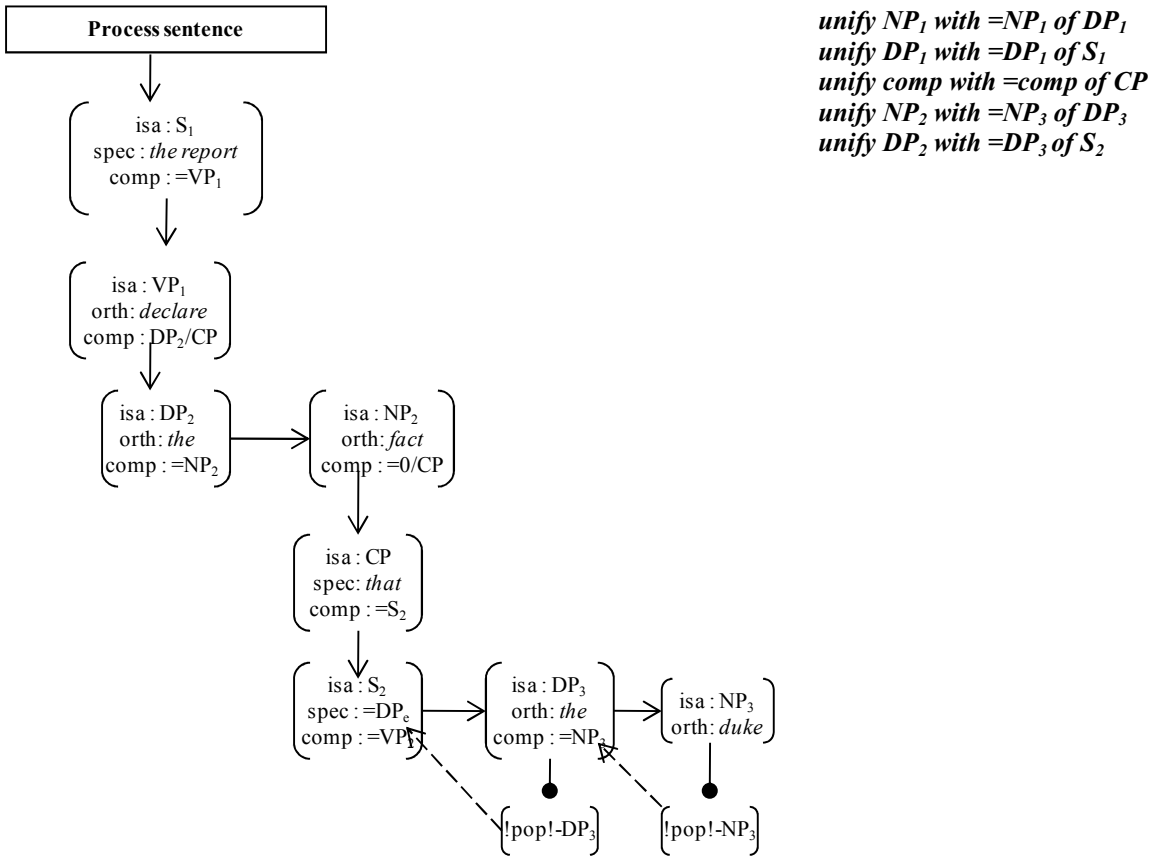
Then the processor encounters the complementizer *that*. The retrieval of the complementizer leads to the creation of a new subgoal: 'process CP.' The processor continues with the retrieval

of the CP-chunk and the comp-*that*-chunk.[1]

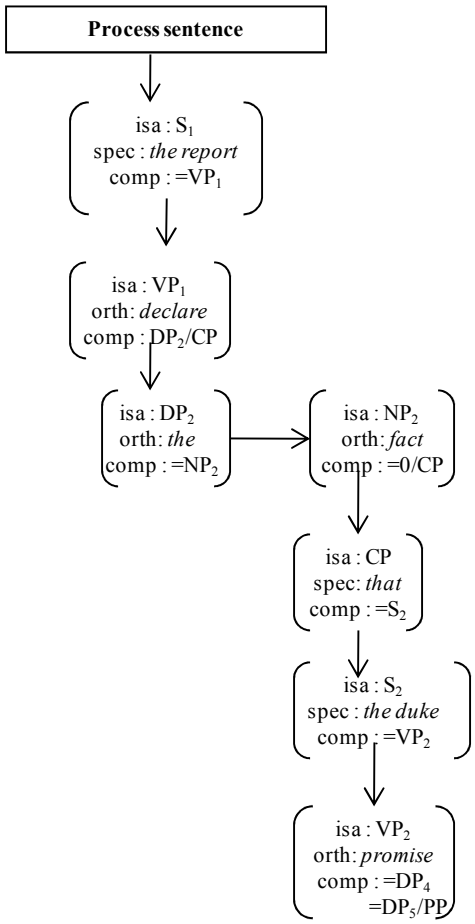


The comp-*that*-chunk unifies with the open =comp value in the CP-chunk, but the CP-chunk still has an open value, namely an =S. The S-chunk is retrieved and placed in the retrieval buffer. It has two open values (=DP and =VP) and is, hence, sent to the problem state buffer until both of these values are resolved. With the comprehension of *the* and the retrieval of the DP-*the*-chunk, the processing of the DP subject begins. The chain of retrievals, popping, and unifying for the subject DP are shown below.

[1] For ease of presentation, I show the retrieval of these two chunks, the popping of the comp-*that*-chunk, and the unification of these chunks in one step.

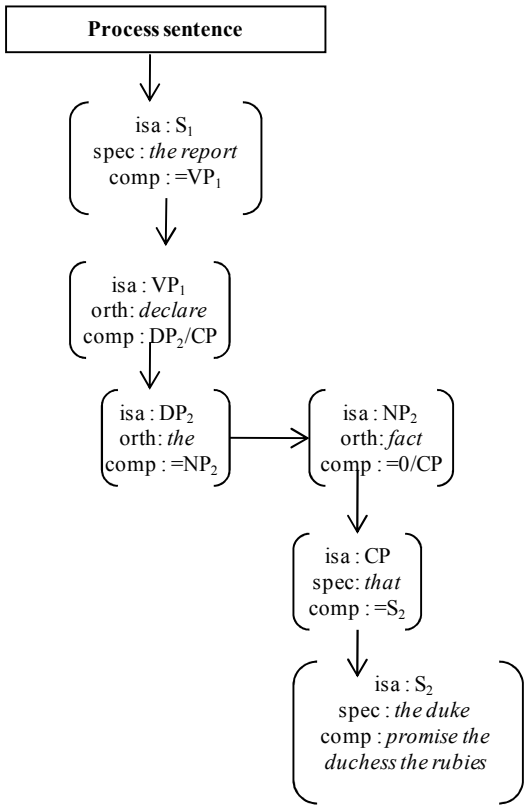


The DP-*the duke*-chunk is unified with the open =DP in the S-chunk, satisfying one of the S-chunk's open values, leaving only the =VP unresolved. Once the *promised* is processed, the VP-*promise*-chunk is retrieved and placed in the retrieval buffer.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$
unify comp with =comp of CP
unify NP$_2$ with =NP$_3$ of DP$_3$
unify DP$_2$ with =DP$_3$ of S$_2$

This chunk has two open values for its arguments, so it is placed into the problem state buffer until these values are resolved. The processing of the VP-*promise*-chunk's two arguments proceeds exactly as it did in the matrix demonstration above, so I do not show the process here. We pick up the processing again with the completed VP-*promise*-chunk, after the unification of VP_2 with the = VP value of the of S_2-chunk. The unification cycles that arose during this processing are shown in the right-hand column.

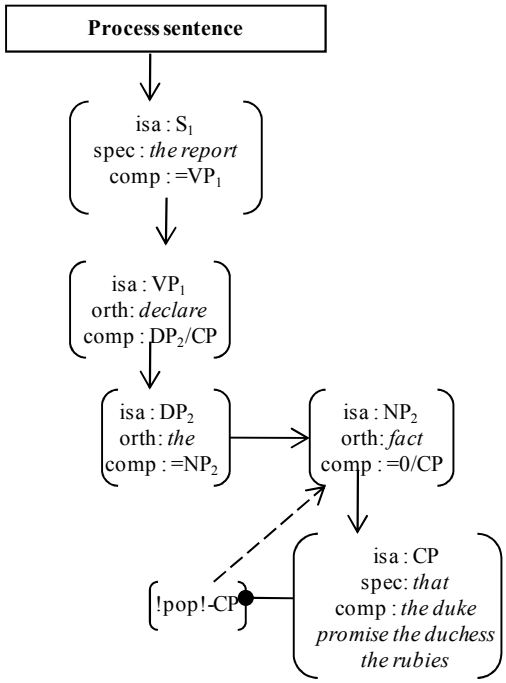


unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$
unify comp with =comp of CP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of S$_2$
unify NP$_4$ with =NP$_4$ of DP$_4$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify NP$_5$ with =NP$_5$ of DP$_3$
unify DP$_5$ with =DP$_5$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$

Note that thus far the product of each unification cycle has unified with an open value in the chunk associated with the next subgoal in the problem state. Thus, each unification cycle has been part of the same unification chain.

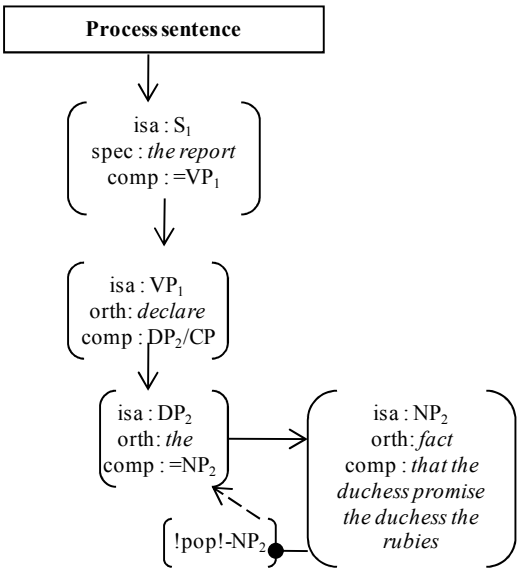
Now that the S-chunk has been satisfied, it is popped. However, in the current demonstration—unlike the matrix example above, the S-chunk does not satisfy the final goal in the control state (i.e. 'process sentence'). There are still many stacked sugoals in the problem state that need to be resolved. The popped S-chunk unifies with the open =S value in the CP-chunk, and this unification cycle is added to the ever-growing unification chain. The values for the CP-chunk are all satisfied, so the CP-chunk is popped. Its values satisfy the open =CP value

in the NP-*fact*-chunk, and they are unified, adding another unification cycle to the chain.



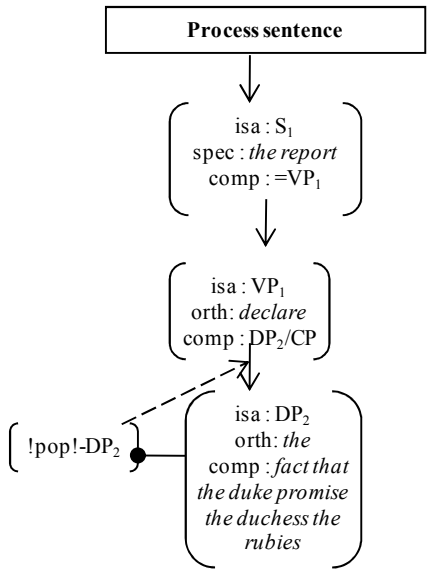
unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$
unify comp with =comp of CP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of S$_2$
unify NP$_4$ with =NP$_4$ of DP$_4$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify NP$_5$ with =NP$_5$ of DP$_3$
unify DP$_5$ with =DP$_5$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of CP
unify CP with =CP of NP$_2$

We continue to unify chunks, working our way back toward the main goal 'process sentence.' The NP$_2$-chunk's values are satisfied, so it is popped. It unifies with the open =NP$_2$ value in the DP$_2$-chunk in the problem state buffer, adding another cycle to the chain.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$
unify comp with =comp of CP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of S$_2$
unify NP$_4$ with =NP$_4$ of DP$_4$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify NP$_5$ with =NP$_5$ of DP$_3$
unify DP$_5$ with =DP$_5$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of CP
unify CP with =CP of NP$_2$
unify NP$_2$ with =NP$_2$ of DP$_2$

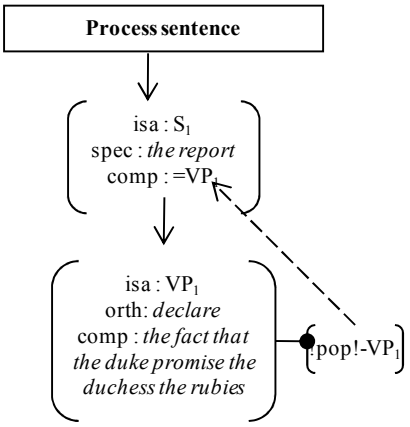
The DP$_2$-chunk's open values are resolved, it pops, unifies with the open =DP$_2$ value of the VP$_1$-chunk, and another link is added to chain.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$
unify comp with =comp of CP
unify NP$_3$ with =NP$_3$ of DP$_3$
unify DP$_3$ with =DP$_3$ of S$_2$
unify NP$_4$ with =NP$_4$ of DP$_4$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify NP$_5$ with =NP$_5$ of DP$_3$
unify DP$_5$ with =DP$_5$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of CP
unify CP with =CP of NP$_2$
unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP$_1$

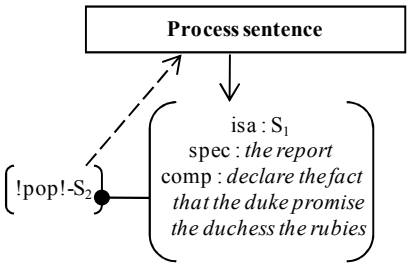
The processing continues with the popping of the VP$_1$-chunk and its unification with the open

=VP_1 value of the S_1-chunk, adding a link to the unification chain.



unify NP_1 with =NP_1 of DP_1
unify DP_1 with =DP_1 of S_1
unify comp with =comp of CP
unify NP_3 with =NP_3 of DP_3
unify DP_3 with =DP_3 of S_2
unify NP_4 with =NP_4 of DP_4
unify DP_4 with =DP_4 of VP_2
unify NP_5 with =NP_5 of DP_3
unify DP_5 with =DP_5 of VP_2
unify VP_2 with =VP_2 of S_2
unify S_2 with =S_2 of CP
unify CP with =CP of NP_2
unify NP_2 with =NP_2 of DP_2
unify DP_2 with =DP_2 of VP_1
unify VP_1 with =VP_1 of S_1

Finally, both of the open values for the S_1-chunk are satisfied, and the S_1-chunk is popped. It unifies with the main goal in the control state: 'process sentence." We have reached the end of the unification chain that the prime is associated with.

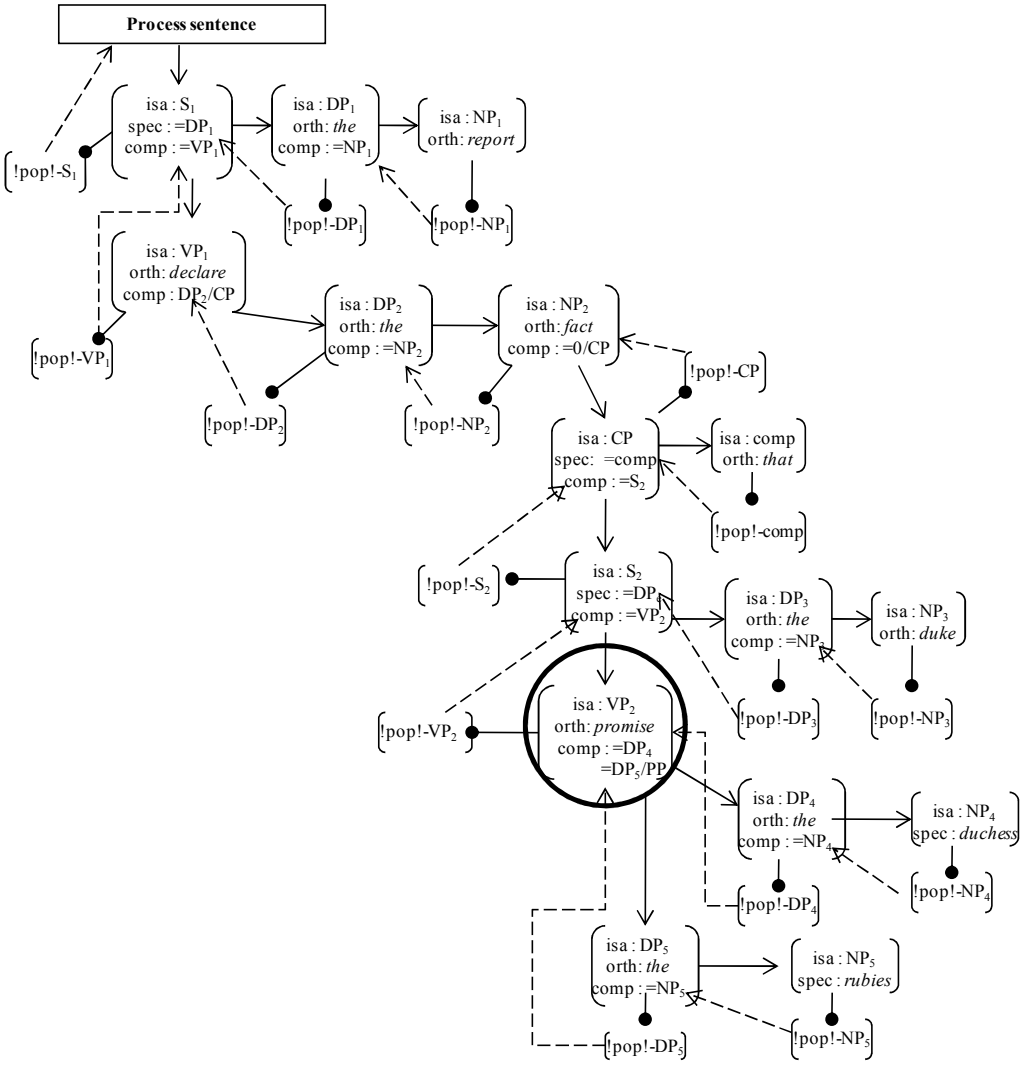


unify NP_1 with =NP_1 of DP_1
unify DP_1 with =DP_1 of S_1
unify comp with =comp of CP
unify NP_3 with =NP_3 of DP_3
unify DP_3 with =DP_3 of S_2
unify NP_4 with =NP_4 of DP_4
unify DP_4 with =DP_4 of VP_2
unify NP_5 with =NP_5 of DP_3
unify DP_5 with =DP_5 of VP_2
unify VP_2 with =VP_2 of S_2
unify S_2 with =S_2 of CP
unify CP with =CP of NP_2
unify NP_2 with =NP_2 of DP_2
unify DP_2 with =DP_2 of VP_1
unify VP_1 with =VP_1 of S_1
unify S_1 with the control state

Figure 3.7 depicts all the retrieved chunks, poppings, and unifications associated with the processing of the sentence "The report declared the fact that the duke promised the duchess the

rubies." In this diagram, we see that the prime word, *promise* (circled), occurred in a larger network of chunks than the same prime in the matrix sentence (i.e. "The duke promised the duchess the rubies.").

Figure 3.7: Retrieval of chunks and rules for processing noun complement clauses



In this example (as in the previous one), the product of each unification cycle fed directly into the problem state. In other words, when two chunks were unified, the unified form was of a type

that satisfied one of the open values of the next available chunk in the problem state buffer.

Recall that the prime in the matrix clause and the prime in the noun complement clause occurred in the same linear position, i.e., they were the same number of syllables and milliseconds away from the target. Hence, recency was the same across the two structural contexts. However, the length of the unification chains that they were associated with and the number of elements within these chains varied drastically. To make this easier to visualize, consider Table 3.5, which contains the unification chain and its cycles (right-hand side) and the chunks associated with this chain (i.e. the unification chain that arose during the processing of "The report declared the fact that the duke promised the duchess the rubies").

Table 3.5: Unification chain and associated chunks for prime in noun complement clause

Retrieved chunks	Unification cycles
S-chunk	*unify* NP_1 *with* $=NP_1$ *of* DP_1
DP-*the*-chunk	*unify* DP_1 *with* $=DP_1$ *of* S_1
NP-*report*-chunk	*unify comp with =comp of CP*
VP-*declare*-chunk	*unify* NP_3 *with* $=NP_3$ *of* DP_3
DP-*the*-chunk	*unify* DP_3 *with* $=DP_3$ *of* S_2
NP-*fact*-chunk	*unify* NP_4 *with* $=NP_4$ *of* DP_4
CP-chunk	*unify* DP_4 *with* $=DP_4$ *of* VP_2
Comp-*that*-chunk	*unify* NP_5 *with* $=NP_5$ *of* DP_3
S-chunk	*unify* DP_5 *with* $=DP_5$ *of* VP_2
DP-*the*-chunk	*unify* VP_2 *with* $=VP_2$ *of* S_2
NP-*duke*-chunk	*unify* S_2 *with* $=S_2$ *of CP*
VP-*promise*-chunk	*unify CP with =CP of* NP_2
DP-*the*-chunk	*unify* NP_2 *with* $=NP_2$ *of* DP_2
NP-*duchess*-chunk	*unify* DP_2 *with* $=DP_2$ *of* VP_1
DP-*the*-chunk	*unify* VP_1 *with* $=VP_1$ *of* S_1
NP-*rubies*-chunk	*unify* S_1 *with the control state*

Here we see that there are 16 chunks associated with the unification chain in which the prime (VP-*promise*-chunk) occurs. Compare this with the number of chunks occurring in the unification chain for the matrix example, as shown in Table 3.6.

Table 3.6: Comparison of matrix and noun complement clause chunks

	Matrix clause	Sentence with noun complement clause
	S-chunk	S-chunk
	DP-*the*-chunk	DP-*the*-chunk
	NP-*duke*-chunk	NP-*report*-chunk
	VP-*promise*-chunk	VP-*declare*-chunk
	DP-*the*-chunk	DP-*the*-chunk
	NP-*duchess*-chunk	NP-*fact*-chunk
	DP-*the*-chunk	CP-chunk
	NP-*rubies*-chunk	Comp-*that*-chunk
		S-chunk
		DP-*the*-chunk
		NP-*duke*-chunk
		VP-*promise*-chunk
		DP-*the*-chunk
		NP-*duchess*-chunk
		DP-*the*-chunk
		NP-*rubies*-chunk
# of chunks (G/j)	**8 chunks (0.13)**	**16 chunks (0.06)**

Here we see that the unification chain for the matrix clause is associated with 8 chunks, quite a bit less than the number of chunks used in the processing of the noun complement clause sentence (16 chunks). According to the model of processing presented in Chapter 2, the number of chunks associated with a particular context affects the activation of chunks. During retrieval,

the processor retrieves the unification chains. The more chunks in the chain (i.e. the higher value of *j*), the less activation each chunk receives (see the Total Activation Equation as explained in Chapter 2, section 3.3.2 and as discussed earlier in the current section). The reason for this is that there is a limited amount of cognitive resources for a goal *G.* This amount is constant and must be shared equally among the chunks in a context *G/j.* The amount of cognitive resources each chunk receives in turn affects the weight of each chunk *Wj.* Because the default value of *G* is 1, each chunk in the single clause sentence receives 0.13 of the cognitive resources for the goal, whereas each chunk in the noun complement clause sentence receives 0.06. Because each chunk in the noun complement clause sentence has less of the resources, the prime is less active than if it had received more of the resources. This lower activation makes the retrieval of the prime slower.

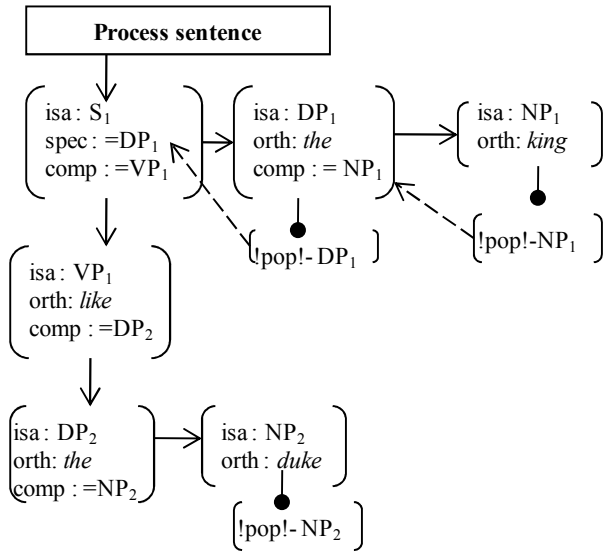
The model of language processing we are using predicts differences not only between single clause sentences and sentences with noun complement clauses but also between sentences with noun complement clauses and sentences with relative clauses. The reason these two sentences types should differ is that one contains an argument clause (i.e. the sentence with a noun complement clause) and one contains an adjunct clause (i.e. the sentence with a relative clause). The language processing model presented in Chapter 2 treats arguments and adjuncts as processed differently (Chapter 2, section 3.4.3). In particular, arguments occur in the same unification chains as their selectors, whereas adjuncts form distinct chains. PRICE predicts that this difference should have implications for priming, namely, primes that are associated with sentences that contain argument clauses (e.g. verb complement or noun complement clauses)

should demonstrate less priming than those that are associated with sentences with adjunct clauses (e.g. relative clauses) when time (recency) is held constant. Let us now explore this prediction by comparing priming from relative clauses to priming from noun complement clauses.

We begin with a sentence with a relative clause:

(16) Prime in relative clause sentence
The king liked the duke [who **promised** the duchess the rubies].

In this sentence, we start with the main goal in the control state buffer, 'process sentence.' This goal leads to the retrieval of an S-chunk, which in turn is sent to the problem state due to its open =DP and =VP values. The processing of these two values become subgoals of the problem state. The formation of the main goal's subject DP and predicate VP proceed in a manner similar to the formations that occurred in the demonstrations above. The primary change is that different chunks (e.g. NP-*king*-chunk and VP-*like*-chunk) were retrieved leading to slightly different patterns (e.g. the building of a sentence with a transitive verb rather than a dative verb). Rather than step through each component of this process, I have provided the pattern of retrievals, poppings, and unifications for all chunks up to the relative clause.



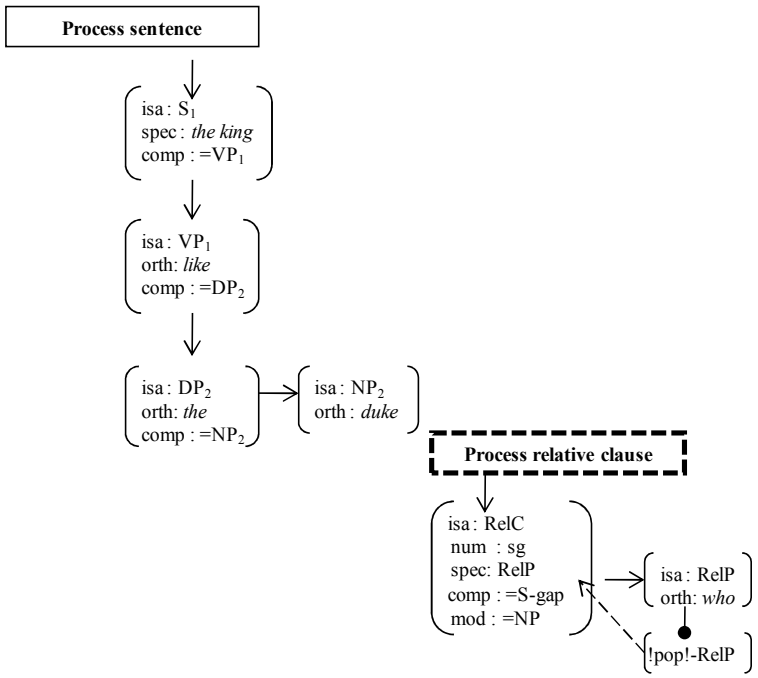
unify NP1 with =NP1 of DP1
unify DP1 with =DP1 of S1

Note that the diagram above stops with the popping of NP_2. Rather than moving to resolve the next subgoal already on the subgoal stack (i.e. 'process DP,' see Chapter 2, section 3.4 for demonstrations of the updating and resolution of subgoals via unification), the processor instead begins to process a RelC by generating the subgoals necessary for its processing. To denote this shift, I adopt new notation, namely the use of a dotted box around the heading 'process RelC.' This notation indicates that ultimately this unification chain is separate from the unification chain associated with the processing of the rest of the sentence.

Note also that there is not an arrow (→) leading to the RelC-chunk from the NP-chunk. The reason for this is that the arrow denotes a retrieval based on features of the chunk currently in the problem state buffer. For example, if there is a DP-chunk with an open =NP value in the problem state buffer, the processor may retrieve an NP-chunk to satisfy the DP-chunk's 'process NP' subgoal. I indicate the relationship between a subgoal in the problem state and a retrieval using the solid arrow. When the retrieved item does not have this type of relationship, I do not

use the arrow. For example, if the problem state has the subgoal 'process NP', but an AdjP-chunk is retrieved, its retrieval is not due or linked to a subgoal in the problem state buffer. Because the retrieval of the RelC-chunk in the current example differs from the example's other retrievals in this respect, I do not use the normal retrieval notation. Furthermore, for ease of tracking, I distinguish between the rules necessary for the relative clause's processing and those necessary for the matrix clause's processing by not bolding the rules associated with the RelC-chunk (e.g. '*unify RelP with =RelP$_1$ of RelC*') in the right-hand column in the diagram below.

The processing of *who* leads to the retrieval of a RelC-chunk (i.e. a relative clause chunk) and a RelP-*who*-chunk. Both of these retrievals and the unification of the RelP-chunk and the RelC-chunk are shown below.

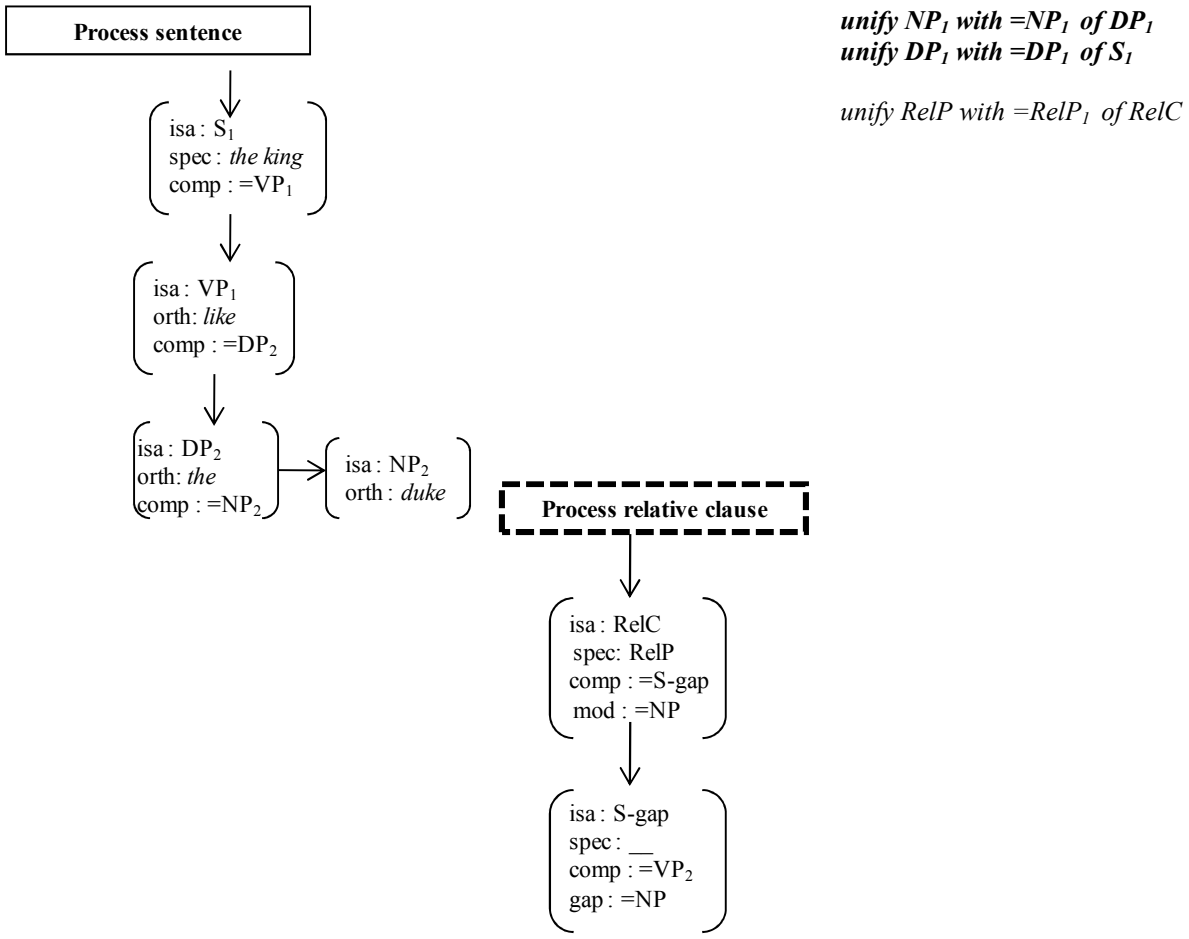


unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

unify RelP with =RelP$_1$ of RelC

Note that the RelC has not only a ‘spec’ and ‘comp’ feature but also a ‘mod’ feature. This ‘mod’ feature denotes the type of element the RelC modifies, i.e. a noun (=NP). The resolution of this feature-value pair, along with the resolution of the other open values (=RelP and =S-gap) must occur before the RelC can be popped. The S-gap-chunk is similar to the S-chunk used in previous examples. S-gap-chunks are used during the processing of clauses that are missing an explicit argument. For example, during the processing of a normal S-chunk, the processor predicts the processing of both a subject (DP) and a predicate (VP) as in “the queen drank the tea.” However, when the processor processes an S-gap-chunk, it predicts that one of the arguments in the clause is gapped. For instance, in subject-relative clauses (e.g., “who __ drank the tea”) or in object relative clauses (e.g., “what the queen drank __”), there is a missing argument. As such, an S-gap-chunk act as a cue for the retrieval of the dislocated item (Lewis & Vasishth 2005). This type of cue occurs not only for S-gap-chunks but also other types of chunks, e.g. VP-gap-chunks. The final resolution of these gaps depends in part on the unification of the open value in the ‘mod’ feature with a retrieved chunk of the appropriate type (e.g. an NP-chunk for a RelC-chunk’s open =NP value in its ‘mod’ feature).

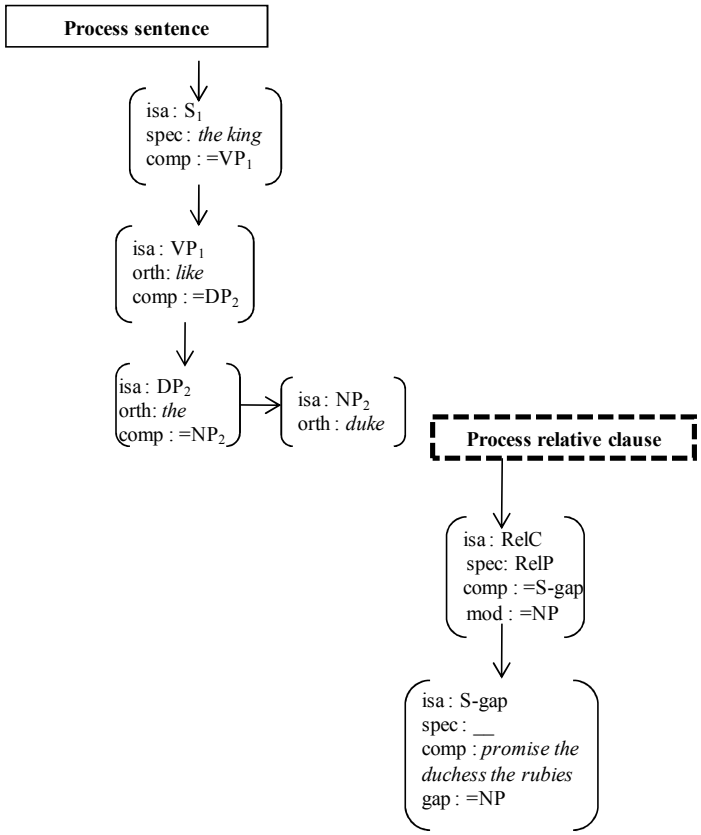
Above, I presented the RelC-chunk and the RelP-chunk as already having been retrieved and unified, satisfying one of the RelC’s subgoals. I pick up below with the subgoal of processing the S-chunk.



The retrieval of the S-gap-chunk sets into motion the chain of retrievals, poppings, necessary for satisfying its open values, similar to the way the S-chunks in the above demonstrations did, with a couple of differences. First, this chunk contains a 'gap' feature with the value '=NP'. This feature-value pair denotes presences of a gap ("extraction site") and the type of element that can satisfy the gap (Sag 2009). The second difference is that the 'spec' of the S-gap-chunk does not contain an open =DP value like the S-chunk. Rather, it value of the 'spec' is left unfilled, as denoted by the " __." This 'empty' position is ultimately saturated with the values associated

with the gap feature, i.e. the values that satisfy the open =NP value in ‘gap.’.

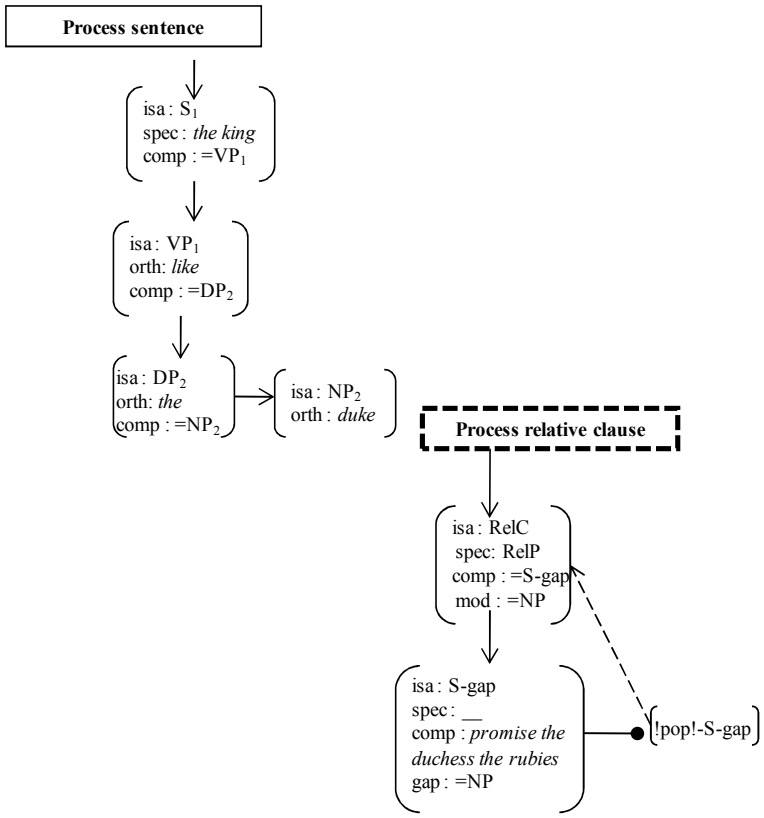
Because the subject of the relative clause has been extracted (as denoted by the ‘spec : __’ and ‘gap : =NP’), the processor turns to resolving the subgoal ‘process VP.’ The processing of this VP is the same as the processing of the dative VP in the matrix and noun complement clause examples above, so I skip these processing steps and pick up with the unification of the completed VP with the open =VP value of the S-chunk.



Unify NP_1 with $=NP_1$ of DP_1
unify DP_1 with $=DP_1$ of S_1

unify RelP with $=RelP_1$ of RelC
unify NP_3 with $=NP_3$ of DP_4
unify DP_3 with $=DP_3$ of VP_2
unify NP_4 with $=NP_4$ of DP_3
unify DP_4 with $=DP_4$ of VP_2
unify VP_2 with $=VP_2$ of S_2

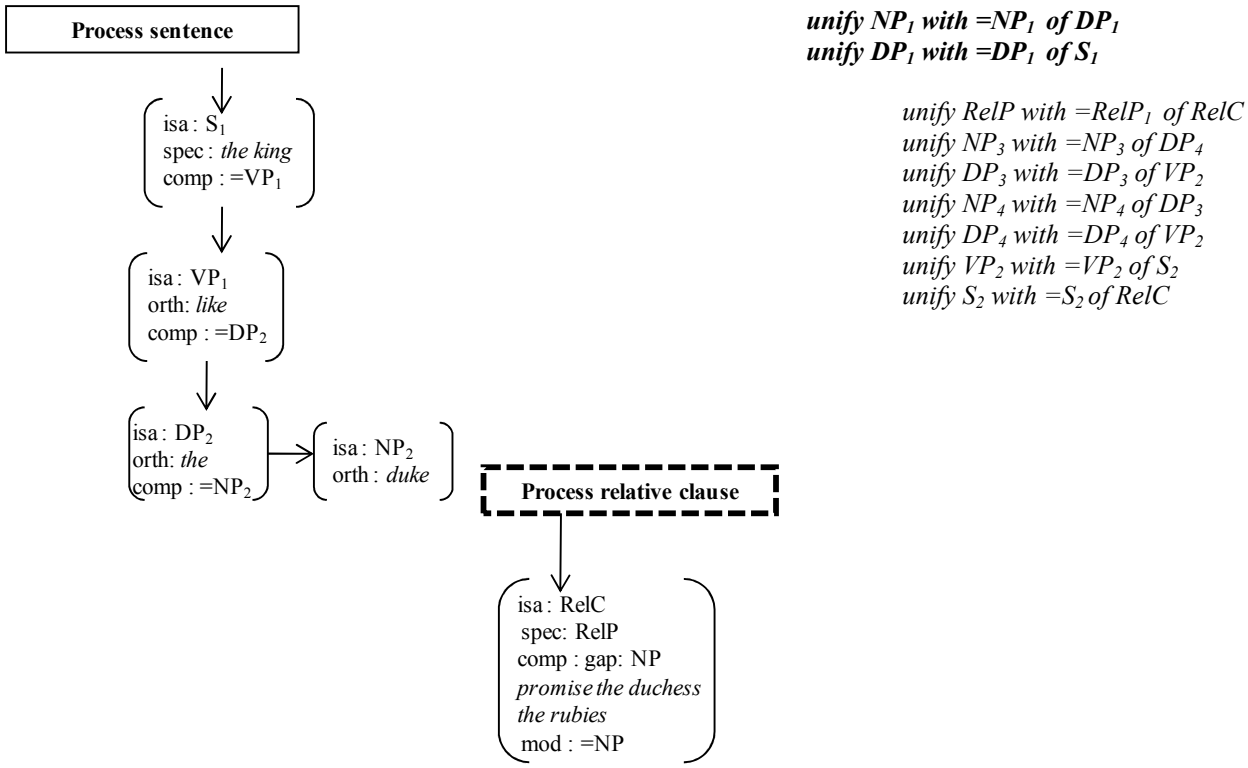
The gap-S-chunk then pops and becomes available for unification with the next subgoal. It unifies with the open =S value in the RelC-chunk, thereby satisfying the top-most subgoal.



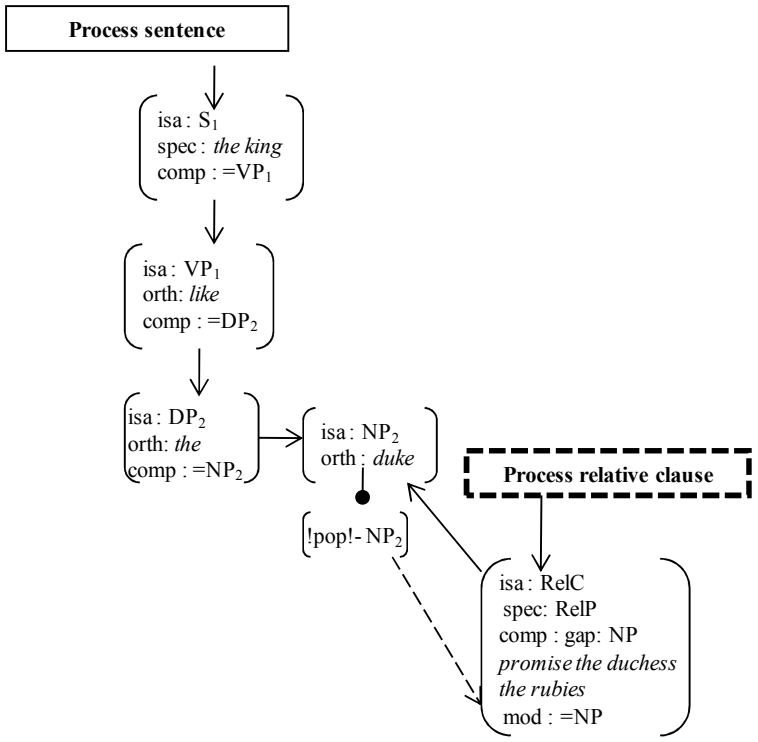
unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

unify RelP with =RelP$_1$ of RelC
unify NP$_3$ with =NP$_3$ of DP$_4$
unify DP$_3$ with =DP$_3$ of VP$_2$
unify NP$_4$ with =NP$_4$ of DP$_3$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of RelC

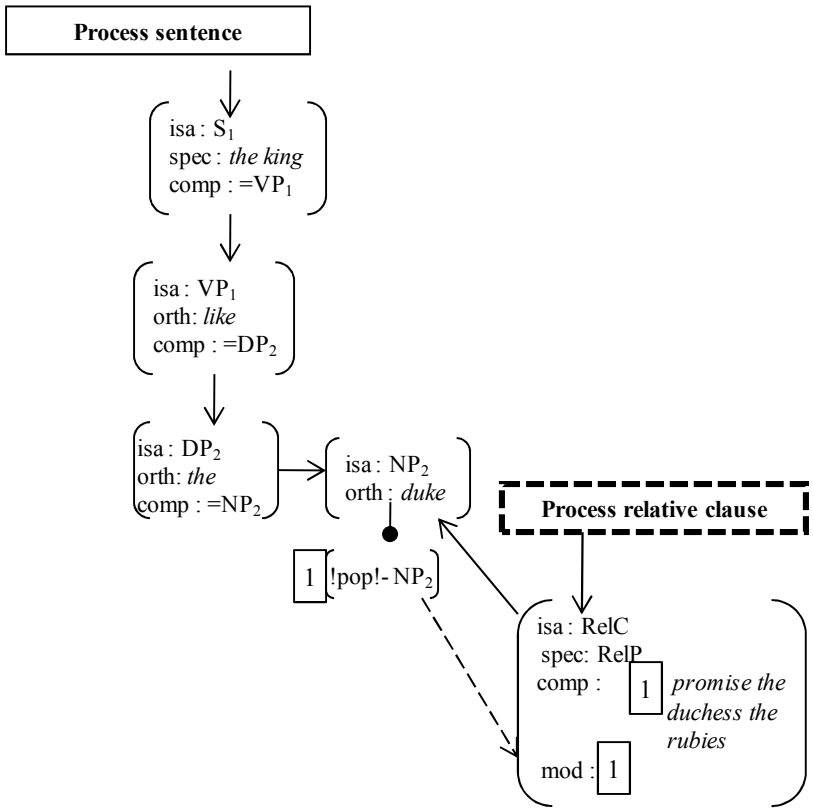
There is only one open value left in the RelC-chunk (i.e. 'mod : =NP), meaning that there is one more subgoal associated with the RelC chunk: 'process NP.' Because there is no currently active or popped NP that could unify with the RelC's =NP, the processor retrieves the most active and relevant NP-chunk from long-term memory, in this case the NP-*duke*-chunk. This retrieval returns the same chunk as was retrieved earlier. This is denoted below by the arrow pointing to the NP-chunk.



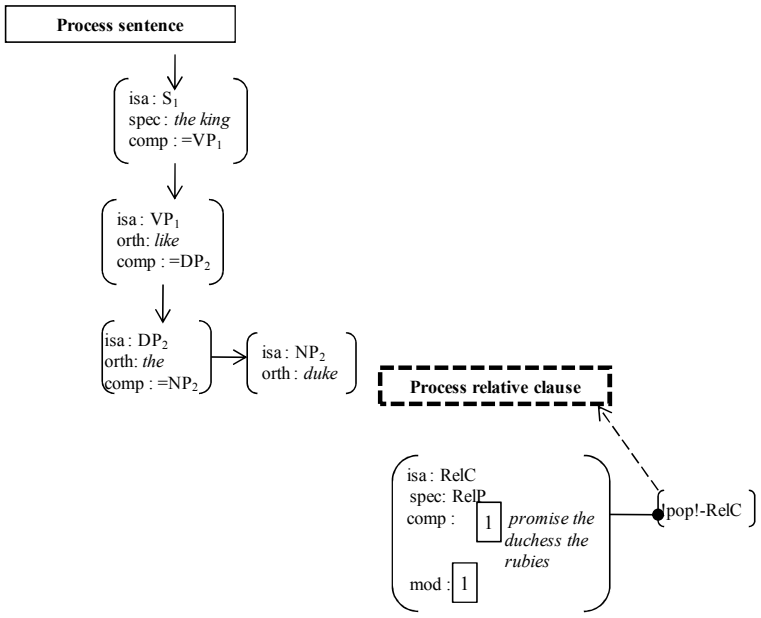
The NP-*duke*-chunk has no open values, so it is popped and unifies with the open =NP in the RelC-chunk.



Once this unification has occurred, the gap list is saturated by the values associated with the NP-*duke*-chunk. To denote this, I use the index 1 as shown in the diagram below. This indexing indicates that the element associated with 'gap' feature of the S-gap-chunk is the NP-chunk that unified with the open =NP value of the RelC-chunk.



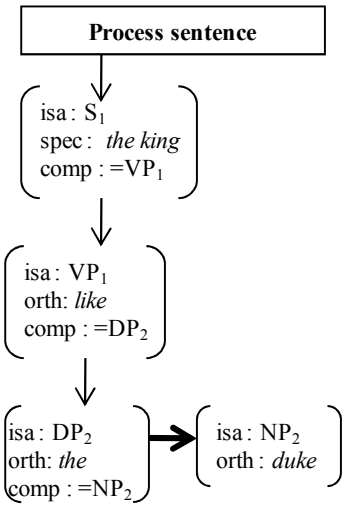
The NP that was retrieved to satisfy the RelC-chunk's subgoal 'process NP' has unified with the open =NP value in the RelC-chunk. As such, the processing of the specific NP-*duke*-chunk is complete, and its activation can begin to decay. Thus, I do not show it any longer but rather denote it via the indexing. All of the open values in the RelC-chunk are now filled. The subgoals associated with the processing of the RelC have all been resolved and the RelC-clause is popped.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

unify RelP with =RelP$_1$ of RelC
unify NP$_3$ with =NP$_3$ of DP$_4$
unify DP$_3$ with =DP$_3$ of VP$_2$
unify NP$_4$ with =NP$_4$ of DP$_3$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of RelC
unify S$_2$ with =S$_2$ of RelC
unify NP$_2$ with =NP of RelC

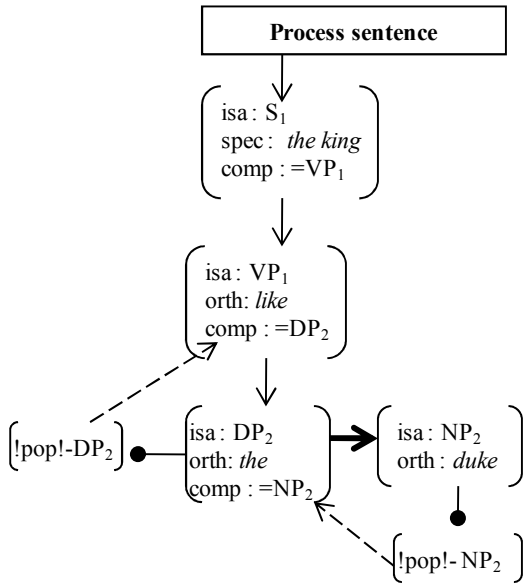
Now that the RelC has been popped, the processor checks the buffers to see if it can unify with any of the open values in the chunk in the problem state buffer (i.e. the DP-*the*-chunk). The DP-chunk does not have an open value that requires a RelC-chunk, so the RelC-chunk cannot unify with an element in the problem state and is sent to LTM. The processor moves to resolve the next subgoal on the subgoal stack, in this case the open =NP value of the DP-chunk. Now that this subgoal is reactivated, the processor needs to retrieve the appropriate NP-chunk, i.e. the NP-*duke*-chunk. The bolded arrow (→) denotes this retrieval.



unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

unify RelP with =RelP$_1$ of RelC
unify NP$_3$ with =NP$_3$ of DP$_4$
unify DP$_3$ with =DP$_3$ of VP$_2$
unify NP$_4$ with =NP$_4$ of DP$_3$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of RelC
unify S$_2$ with =S$_2$ of RelC
unify NP$_2$ with =NP of RelC

The NP-chunk is popped and unified with the open =NP value in the DP-chunk. The DP-chunk is then popped and unified with the open =DP value in the VP-chunk. Because both of these unification cycles produce forms that unify with the open values in the problem state, they form a chain. The chain they form is connected to the chain generated during the processing of the matrix subject.

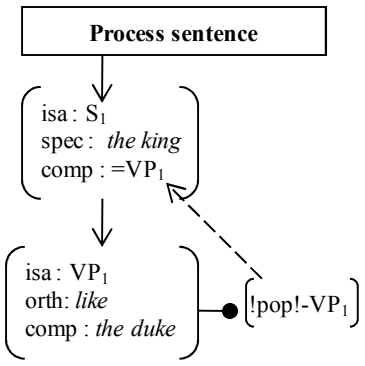


unify NP₁ with =NP₁ of DP₁
unify DP₁ with =DP₁ of S₁

unify RelP with =RelP₁ of RelC
unify NP₃ with =NP₃ of DP₄
unify DP₃ with =DP₃ of VP₂
unify NP₄ with =NP₄ of DP₃
unify DP₄ with =DP₄ of VP₂
unify VP₂ with =VP₂ of S₂
unify S₂ with =S₂ of RelC
unify S₂ with =S₂ of RelC
unify NP₂ with =NP of RelC

unify NP₂ with =NP₂ of DP₂
unify DP₂ with =DP₂ of VP₁

The VP-chunk pops and unifies with the open =VP value in the S-chunk. Because the unification of this VP-chunk and the subject DP ("the duke") both satisfy open values in the same chunk, their unification chains are part of a single chain.

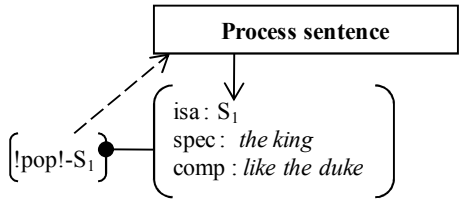


unify NP₁ with =NP₁ of DP₁
unify DP₁ with =DP₁ of S₁

unify RelP with =RelP₁ of RelC
unify NP₃ with =NP₃ of DP₄
unify DP₃ with =DP₃ of VP₂
unify NP₄ with =NP₄ of DP₃
unify DP₄ with =DP₄ of VP₂
unify VP₂ with =VP₂ of S₂
unify S₂ with =S₂ of RelC
unify S₂ with =S₂ of RelC
unify NP₂ with =NP of RelC

unify NP₂ with =NP₂ of DP₂
unify DP₂ with =DP₂ of VP₁
unify VP₁ with =VP₁ of S₁

The S-chunk's open values are now resolved. It is popped and unifies with the main goal in the control state buffer, and then the sentence proceeds to LTM.



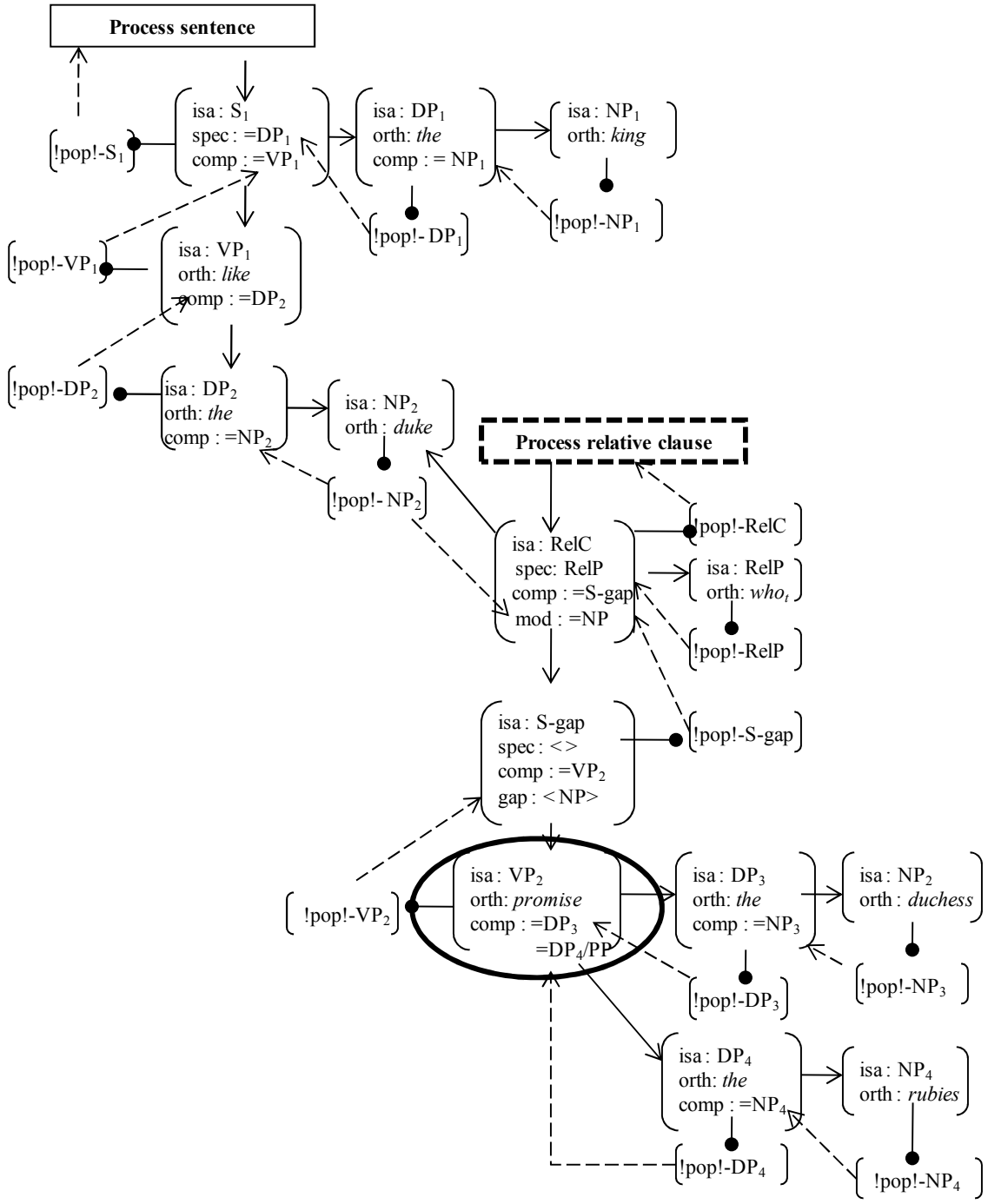
unify NP$_1$ with =NP$_1$ of DP$_1$
unify DP$_1$ with =DP$_1$ of S$_1$

unify RelP with =RelP$_1$ of RelC
unify NP$_3$ with =NP$_3$ of DP$_4$
unify DP$_3$ with =DP$_3$ of VP$_2$
unify NP$_4$ with =NP$_4$ of DP$_3$
unify DP$_4$ with =DP$_4$ of VP$_2$
unify VP$_2$ with =VP$_2$ of S$_2$
unify S$_2$ with =S$_2$ of RelC
unify S$_2$ with =S$_2$ of RelC
unify NP$_2$ with =NP of RelC

unify NP$_2$ with =NP$_2$ of DP$_2$
unify DP$_2$ with =DP$_2$ of VP$_1$
unify VP$_1$ with =VP$_1$ of S$_1$
unify S$_1$ with control state

Figure 3.8 below depicts all of the steps involved in processing the sentence "The king likes the lord who promised the duchess the rubies."

Figure 3.8: Retrieval of chunks and rules for processing a sentence with a relative clause



The sentence is associated with two unification chains. As such, the memory trace for the

sentence has two parts: one that represents the processing of the matrix clause, one that represents the processing of the relative clause. These two unification chains along with their associated chunks are depicted in Table 3.7 below.

Table 3.7: Unification chain and associated chunks for prime in relative clause

Retrieved chunks	Unification cycles
S-chunk	***Chain 1:***
DP-*the*-chunk	*unify NP_1 with $=NP_1$ of DP_1*
NP-*king*-chunk	*unify DP_1 with $=DP_1$ of S_1*
VP-*like*-chunk	*unify NP_2 with $=NP_2$ of DP_2*
DP-*the*-chunk	*unify DP_2 with $=DP_2$ of VP_1*
NP-*duke*-chunk	*unify VP_1 with $=VP_1$ of S_1*
	unify S_1 with control state
RelC-chunk	***Chain 2:***
RelP-*who*-chunk	*unify RelP with $=RelP_1$ of RelC*
S-gap-chunk	*unify NP_3 with $=NP_3$ of DP_4*
VP-*promise*-chunk	*unify DP_3 with $=DP_3$ of VP_2*
DP-*the*-chunk	*unify NP_4 with $=NP_4$ of DP_3*
NP-*duchess*-chunk	*unify DP_4 with $=DP_4$ of VP_2*
DP-*the*-chunk	*unify VP_2 with $=VP_2$ of S_2*
NP-*rubies*-chunk	*unify S_2 with $=S_2$ of RelC*
	unify S_2 with $=S_2$ of RelC
	unify NP_2 with =NP of RelC

The important thing to note in the table above is the fact that the chunks used during the

generation of each chain are grouped separately, meaning all the chunks used during the formation of the matrix clause (Chain 1) are associated with the same chain, whereas those used during the formation of the adjunct clause (Chain 2) are associated with the same chain. According to the model of language processing presented in Chapter 2, the processor can retrieve these chains independent of one another. When the processor retrieves a particular chain, it needs to search through only the chunks associated with that chain to determine whether a particular chunk was retrieved. Cognitive resources are distributed over only the chunks associated with a particular chain. This means that if the processor retrieved Chain 2 for the relative clause "duke who promised the duchess the rubies," i.e. the chain associated with the prime *promised*, each of the 9 chunks would receive 0.11 of the cognitive resources. This numeric value is similar to that which the chunks in the matrix clause received (0.13). Consider Table 3.8 below. Each column contains the chunks associated with unification chain in which the prime occurs.

Table 3.8: Comparison of matrix, noun complement, and relative clause unification chains

	Matrix clause	Sentence with noun complement clause	Relative clause
	S-chunk	S-chunk	RelC-chunk
	DP-*the*-chunk	DP-*the*-chunk	RelP-*who*-chunk
	NP-*duke*-chunk	NP-*report*-chunk	S-gap-chunk
	VP-*promise*-chunk	VP-*declare*-chunk	VP-*promise*-chunk
	DP-*the*-chunk	DP-*the*-chunk	DP-*the*-chunk
	NP-*duchess*-chunk	NP-*fact*-chunk	NP-*duchess*-chunk
	DP-*the*-chunk	CP-chunk	DP-*the*-chunk
	NP-*rubies*-chunk	Comp-*that*-chunk	NP-*rubies*-chunk
		S-chunk	NP-*duke*-chunk
		DP-*the*-chunk	
		NP-*duke*-chunk	
		VP-*promise*-chunk	
		DP-*the*-chunk	
		NP-*duchess*-chunk	
		DP-*the*-chunk	
		NP-*rubies*-chunk	
# of chunks (G/j)	**8 chunks (0.13)**	**16 chunks (0.06)**	**9 chunks (0.11)**

The model of language processing presented in Chapter 2 further claims that differences are likely to arise between arguments and adjuncts. Given the three structural contexts we have covered thus far (matrix clause, noun complement clause, and relative clause), the claims are that (i) primes in matrix clauses lead to the quickest response times, (ii) noun complement clauses the slowest, and (iii) relative clauses somewhere between the two. Thus far, the response time data are in keeping with these predictions.

However, one aspect of PRICE's claim did not hold true. The prediction that arguments

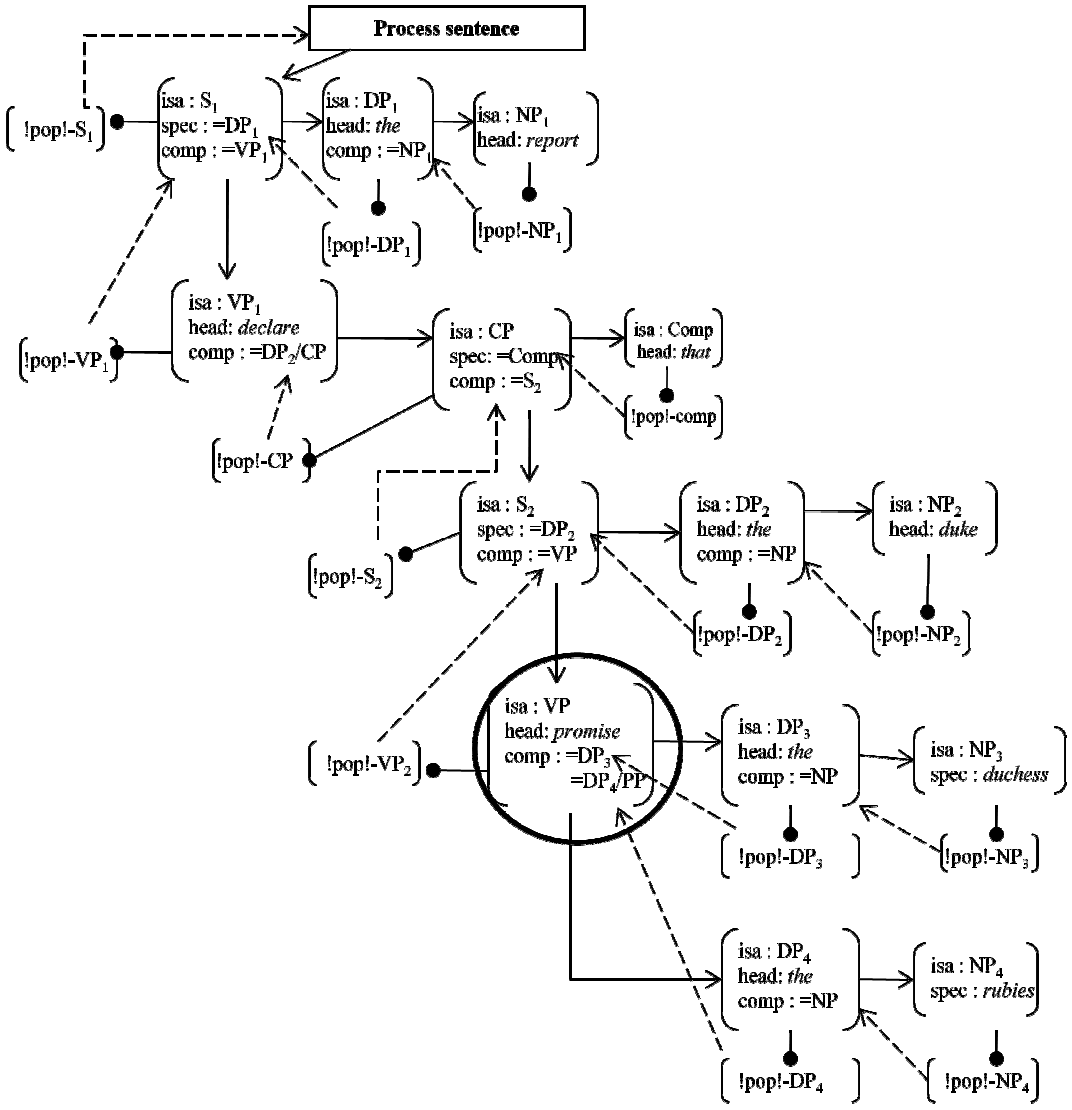
and adjuncts are inherently different did not hold for the verb complement clause primes. Response time for primes in verb complement clauses (average 732 msec) was not significantly different from response times for those in relative clauses (average 726 msec) or matrix clauses (average 726 msec). The response times for primes in verb complement clauses did, however, differ from response times for primes in noun complement clauses (average 761 msec). There are a couple of potential reasons for the response time difference between noun complement clause primes and verb complement clauses. Before I address the possibilities, consider the prime in the verb complement clause below.

(17) Prime in verb complement clause
The report declared that the duke **promised** the duchess the rubies.

According to the processing model presented in Chapter 2, the steps involved in processing (17) are virtually the same as those involved in process a similar sentence with a noun complement clause such as "The report declared the fact that the duke **promised** the duchess the rubies." The primary difference between the two is that the complement clause is an argument of a noun in one and a verb in the other. However, the processing of both types of complement clauses leads to the formation of a single chain for the entire sentence, rather than the multiple unification chains that are associated with the processing of sentence with a relative clause.

Consider the pattern of retrievals, poppings, and unifications associated with the processing of sentence (17) in Figure 3.9 below:

Figure 3.9: Retrieval of chunks and rules for processing a verb complement clause



Just as in the noun complement clause's processing, each unification cycle in the verb complement clause's processing links to the next cycle. That is, the product of one unification cycle can satisfy an open value in the problem state buffer, thereby adding another link to the unification chain. Each step involved in the processing of the sentence follows directly from the

previous, and each popping and unification leads directly back into the problem state.

Table 3.9 below contains all the chunks and unification cycles involved in the verb complement clause prime's (i.e. (17)) unification chain.

Table 3.9: Unification chain and associated chunks for prime in noun complement clause

Retrieved chunks	Unification cycles
S-chunk	*unify* NP_1 *with* $=NP_1$ *of* DP_1
DP-*the*-chunk	*unify* DP_1 *with* $=DP_1$ *of* S_1
NP-*report*-chunk	*unify comp with =comp of CP*
VP-*declare*-chunk	*unify* NP_3 *with* $=NP_3$ *of* DP_3
CP-chunk	*unify* DP_3 *with* $=DP_3$ *of* S_2
Comp-*that*-chunk	*unify* NP_4 *with* $=NP_4$ *of* DP_4
S-chunk	*unify* DP_4 *with* $=DP_4$ *of* VP_2
DP-*the*-chunk	*unify* NP_5 *with* $=NP_5$ *of* DP_3
NP-*duke*-chunk	*unify* DP_5 *with* $=DP_5$ *of* VP_2
VP-*promise*-chunk	*unify* VP_2 *with* $=VP_2$ *of* S_2
DP-*the*-chunk	*unify* S_2 *with* $=S_2$ *of CP*
NP-*duchess*-chunk	*unify CP with =CP of* VP_1
DP-*the*-chunk	*unify* VP_1 *with* $=VP_1$ *of* S_1
NP-*rubies*-chunk	*unify* S_1 *with the control state*

Here we see that the processing of the matrix clause and the complement clause results in a single unification chain. During subsequent retrieval of the chain, the processor needs to retrieve and search through this entire chain to verify whether the prime occurred. The notable difference between this verb complement clause example and the noun complement clause example is that the noun complement clause has two additional unification cycles and two additional chunks.

Table 3.10 compares the chunks retrieved during the processing of the prime and the unification chain that primes are associated with for all four clause types.

Table 3.10: Comparison of all structural context types

	Matrix clause	Sentence with noun complement clause	Relative clause	Sentence with verb complement clause
	S-chunk	S-chunk	RelC-chunk	S-chunk
	DP-*the*-chunk	DP-*the*-chunk	RelP-*who*-chunk	DP-*the*-chunk
	NP-*duke*-chunk	NP-*report*-chunk	S-gap-chunk	NP-*report*-chunk
	VP-*promise*-chunk	VP-*declare*-chunk	VP-*promise*-chunk	VP-*declare*-chunk
	DP-*the*-chunk	DP-*the*-chunk	DP-*the*-chunk	CP-chunk
	NP-*duchess*-chunk	NP-*fact*-chunk	NP-*duchess*-chunk	Comp-*that*-chunk
	DP-*the*-chunk	CP-chunk	DP-*the*-chunk	S-chunk
	NP-*rubies*-chunk	Comp-*that*-chunk	NP-*rubies*-chunk	DP-*the*-chunk
		S-chunk	NP-*duke*-chunk	NP-*duke*-chunk
		DP-*the*-chunk		VP-*promise*-chunk
		NP-*duke*-chunk		DP-*the*-chunk
		VP-*promise*-chunk		NP-*duchess*-chunk
		DP-*the*-chunk		DP-*the*-chunk
		NP-*duchess*-chunk		NP-*rubies*-chunk
		DP-*the*-chunk		
		NP-*rubies*-chunk		
# of chunks (G/j)	**8 chunks (0.13)**	**16 chunks (0.06)**	**9 chunks (0.11)**	**14 chunks (0.07)**

Here we see that the prime in the matrix clause example has the fewest associated chunks, and the prime in the noun complement clause example has the most. The other examples fall somewhere between, with relative clauses more similar to matrix clauses and verb complement clauses more similar to noun complement clauses.

The number of chunks helps determine how much activation each chunk receives: the

more chunks, the less activation per chunk. Assuming that the amount of cognitive resources is constant and equally divided among the chunks, we can approximate how much activation a chunk does or does not receive. Primes in the verb complement clauses and noun complement clauses receive similar amounts of activation (0.07 and 0.06 respectively). However, there is still a significant difference between the two complement types' response times.

There are at least two reasons why lexical priming from verb complement clauses and noun complement clauses may differ. One potential reason stems from the nature of complex noun phrases like the ones used in this experiment. Previous research has argued that complex noun phrases can create islands that limit the extraction of elements and make processing generally more difficult (e.g. Ross 1967, Haegeman1991, Lasnik 1999, Kromann 2004). For example, consider the sentences in (18)-(20), each of which is an example of one of the sentence types discussed early in this section. The extraction of one of the prime verb's arguments is allowed from a matrix clause position (18) and from a verb complement clause (19). However, extraction is not allowed from a noun complement clause (20).

(18) Extraction (matrix clause)
The duke promised the duchess the rubies.
What did the duke promise the duchess __?

(19) Extraction (verb complement clause)
The report declared that the duke promised the duchess the rubies.
What did the report declare that the duke promised the duchess __?

(20) Extraction (noun complement clause)
The report declared the fact that the duke promised the duchess the rubies.
What did the report declare the fact that the duke promised the duchess __?

The same features that make extraction difficult may also inhibit priming. This possibility is

discussed again in Chapter 5.

The second possibility is more closely associated with the PRICE claim by maintaining the distinction between arguments and adjuncts but adding one stipulation. The difference between noun and verb complement clauses arises because the effects of additional chunks are exponential and not linear. The amount of the cognitive resources *G* may be allotted equally among the chunks in the context. However, this does not entail that the effects of additional chunks is linear. One reason to think that each additional chunk compounds the response times is that other retrieval tasks also display exponentially increased reaction times. For example, the Fan Effect, as reported by Anderson (1974), leads to an exponential increase in reaction times. The basic finding of the fan effect is that the more facts that participants learn about a particular concept, the slower their response time to a particular fact about the concept is. Lewis and Anderson (1976) found that although there wasn't a large increase in response times following one additional fact, the addition of two extra facts added approximately 1000 msec to the response time and three additional facts 2000 msec. Although this suggests a linear function after a larger initial leap for each additional fact, it is not clear that the same linear-nature applies to the addition of chunks. Anderson and Reder (1999) argue that the latency for a chunk's retrieval is "an exponential function of the amount of activation reaching [the] chunk," and this amount of activation is affected by the number of other chunks (or bits of information) associated with the target chunk (p 186). Thus, it is possible that additional chunks affect response times exponentially.

This fan effect helps to explain what is occurring with the retrieval of the prime during

the verification task reported in section 4. When the number of chunks associated with a unification chain increases, the processor must take more time checking each chunk to determine whether the prime occurred or not due to lower activation. Previous research in language processing estimates the time for a rule firing is 50 msec, and the latency associated with chunk retrieval is a factor of 0.14 (Lewis & Vasishth 2005), but it is not clear how long it should take to retrieve a chunk, verify whether it matches a target word on a computer screen, and hit a button.

For the time being, let's say that the initial cost of retrieving a chunk in a chain is 5 msec, and the cost associated with checking each chunk is 0.25. I use the formula $T = B\ (1 + c)^n$ where *T* refers to the response time, *B* refers to the time necessary to retrieve a chunk, *c* refers to the addition cost associated with checking a chunk, and *n* refers to the number of chunks the processor must check. If we take the baseline response time that is needed to check 8 chunks as our starting point, we can begin to add the time associated with checking each additional chunk. Checking 8 chunks takes approximately 726 msec. Using the formula above, cost associated with checking the first additional chunk be

$5\ (1 + .25)^1$ = 6.25 msec

and for checking the second chunk

$5\ (1 + .25)^2$= 7.81

and so forth. Given this, we can estimate how much slower the processing of each of the structural contexts associated with the prime should be relative to the matrix clause baseline.

Relative clauses have 1 more chunk than matrix clauses in the stimuli used in the experiment (see section 3.1 for a description). Using the exponential function, the response times

should be 6.25 msec slower for primes in relative clause, leading to a predicted lag of approximately 732 msec. Verb complement clauses have 6 additional chunks, which add 19.1 msec, leading to an estimated response time of approximately 748 msec. Noun complement clauses have 8 additional chunks, adding 29.8 msec, placing the estimated response time at approximately 759 msec.

This pattern of growth is similar to what the results found. Primes in verb complement clauses were, on average, 6 msec slower (732 msec) than the baseline (726 msec), and primes in noun complement clauses were approximately 37 msec slower (761 msec). If the effect of checking through each chunk is exponential, then the additional two chunks may have been enough to make the primes in noun complement clauses significantly less accessible. I return to this point in Chapter 5. For the time being, one thing we know for certain is that primes occurring in different structural contexts lead to different amounts of priming. PRICE contends that these differences arise due to the way the primes were processed.

6. Conclusion

The results from the response time data in this study suggest that primes in noun complement clauses do not facilitate identification as much as those occurring in any of the other three structural contexts considered in the experiment reported in section 4. The processing model presented in Chapter 2 predicts that the retrieval of chunks depends on the amount of interference a chunk experiences from its context and its base activation level. A chunk's base activation weight is sensitive to how recently the chunk was processed, whereas the amount of

interference is sensitive to how many other chunks occurred in the context.

In the current experiment, time was held constant, so the base activation level for each prime should have been the same. The only thing that varied was the structural context in which the prime occurred. The prime's structural context is best identified by the unification chain it is associated with. These chains become units in memory that are retrieved in whole. Each unification chain is associated with the chunks used during the formation of the chain. The longer the chain, the more chunks. The more chunks, the greater the interference. This interference arises because the chunks must share limited cognitive resources. When there are more chunks, each chunk gets less of the resources, weakening the activation of any one chunk.

Appendix 3A: Experimental items for the lexical priming study

Below, NCC stands for Noun Complement Clauses, and VCC stands for Verb Complement Clauses.

Bought

1.	Matrix	The manager left the request, and the secretary bought the supplies for the owner.
	NCC	The manager reported the fact that the secretary bought the supplies for the owner.
	VCC	The manager revealed that the secretary bought the supplies for the owner.
	Relative	The manager liked the secretary who bought the supplies for the owner
2.	Matrix	The reporter smiled, and the agent bought the diamonds for the singer.
	NCC	The reporter stated the fact that the agent bought the diamonds for the singer.
	VCC	The reporter revealed that the agent bought the diamonds for the singer.
	Relative	The reporter kissed the agent who bought the diamonds for the singer.
3.	Matrix	The patient slept, and the doctor bought the cocktail for the surgeon.
	NCC	The patient stated the fact that the doctor bought the cocktail for the surgeon.
	VCC	The patient revealed that the doctor bought the cocktail for the surgeon.
	Relative	The patient met the doctor who bought the cocktail for the surgeon.
4.	Matrix	The man nodded his head, and the clerk bought the cigar for the salesman.
	NCC	The man believed the fact that the clerk bought the cigar for the salesman.
	VCC	The man revealed that the clerk bought the cigar for the salesman.
	Relative	The man saw the clerk who bought the cigar for the salesman.

Offered

1.	Matrix	The mother thanked the maid, and the father offered the sweater to the butler.
	NCC	The mother stated the fact that the father offered the sweater to the butler.
	VCC	The mother revealed that the father offered the sweater to the butler.
	Relative	The mother hugged the father who offered the sweater to the butler.
2.	Matrix	The nurse typed, and the intern offered the files to the dentist.
	NCC	The nurse reported the fact that the intern offered the files to the dentist.
	VCC	The nurse revealed that the intern offered the files to the dentist.
	Relative	The nurse saw the doctor who offered the files to the dentist.
3.	Matrix	The journalist smirked, and the agent offered the bonus to the actress.
	NCC	The journalist reported the fact that the agent offered the bonus to the actress.
	VCC	The journalist revealed that the agent offered the bonus to the actress.
	Relative	The journalist visited the agent who offered the bonus to the actress.
4.	Matrix	The editor napped, and the judge offered the award to the writers.
	NCC	The editor stated the fact that the judge offered the award to the writers.
	VCC	The editor revealed that the judge offered the award to the writers.
	Relative	The editor dated the judge who offered the award to the writers.

Passed

1.	Matrix	The customer sat down, and the host passed the menu to the waiter.

	NCC	The customer stated the fact that the host passed the menu to the waiter.
	VCC	The customer revealed that the host passed the menu to the waiter.
	Relative	The customer knew the host who handed the menu to the waiter.
2.	Matrix	The salesman smiled, and the executives passed the contract to the owner.
	NCC	The salesman believed the fact that executives passed the contract to the owner.
	VCC	The salesman revealed that the executives passed the contract to the owner.
	Relative	The salesman greeted the executives who passed the contract to the owner.
3.	Matrix	The sister pouted, and the brother passed the pencil to the cousin.
	NCC	The sister reported the fact that the brother passed the pencil to the cousin.
	VCC	The sister revealed that the brother passed the pencil to the cousin.
	Relative	The sister liked the brother who passed the pencil to the cousin.
4.	Matrix	The women gathered, and the florist passed the roses to the matron.
	NCC	The women believed the fact that the florist passed the roses to the matron.
	VCC	The women revealed that the florist passed the roses to the matron.
	Relative	The women admired the florist who passed the roses to the matron.

Issued

1.	Matrix	The station received the call, and the policeman issued the ticket to the poet.
	NCC	The station reported the fact that the policeman issued the ticket to the poet.
	VCC	The station revealed that the policeman issued the ticket to the poet.
	Relative	The station commended the policeman who issued the ticket to the poet.
2.	Matrix	The mayor made a speech, and the judge issued the verdict to the lawyer.
	NCC	The mayor stated the fact that the judge issued the verdict to lawyer.
	VCC	The mayor revealed that the judge issued the verdict to the lawyer
	Relative	The mayor spoke with the judge who issued the verdict to the lawyer.
3.	Matrix	The CIA agreed, and the FBI chief issued the jacket to the agent.
	NCC	The CIA reported the fact that the FBI chief issued the jacket to the agent.
	VCC	The CIA revealed that the FBI chief issued the jacket to the agent.
	Relative	The CIA scolded the CIA chief who issued the jacket to the agent.
4.	Matrix	The executive approved, and the banker issued the receipt to the client.
	NCC	The executive believed the fact that the banker issued the receipt to the client.
	VCC	The executive revealed that the banker issued the receipt to the client.
	Relative	The executive hired the banker who issued the receipt to the client.

Sold

1.	Matrix	The butler went to the market, and the baker sold the pastry to the nanny.
	NCC	The butler believed the fact that the baker sold the pastry to the nanny.
	VCC	The butler revealed that the baker sold the pastry to the nanny.
	Relative	The butler liked the baker who sold the pastry to the nanny.
2.	Matrix	The evidence was suppressed, and the employer sold the product to the dealer.
	NCC	The evidence supports the fact that the employer sold the product to the dealer.
	VCC	The evidence revealed that the employer sold the product to the dealer.

	Relative	The evidence exposed the employer who sold the product to the dealer.
3.	Matrix	The participant agreed, and the professor sold the photo to the journal.
	NCC	The participant reported the fact that the professor sold the photo to the journal.
	VCC	The participant revealed that the professor sold the photo to the journal.
	Relative	The participant revealed that the professor sold the photo to the journal.
4.	Matrix	The man read the label, and the grocer sold the rabbit to the butcher.
	NCC	The man stated the fact that the grocer sold the rabbit to the butcher
	VCC	The man revealed that the grocer sold the rabbit to the butcher.
	Relative	The man emailed the grocer who sold the rabbit to the butcher.

Showed

1.	Matrix	The agent called the dealer, and the writer showed the poem to the critic.
	NCC	The agent believed the fact that the writer showed the poem to the critic.
	VCC	The agent revealed that the writer showed the poem to the critic.
	Relative	The agent called the writer who showed the poem to the critic.
2.	Matrix	The columnist gossiped, and the curator showed the drawing to the artist.
	NCC	The columnist reported the fact that the curator showed the drawing to the artist.
	VCC	The columnist revealed that the curator showed the drawing to the artist.
	Relative	The columnist interviewed the curator who showed the drawing to the artist.
3.	Matrix	The activist protested, and the solider showed the orders to the pilot.
	NCC	The activist stated the fact that the solider showed the orders to the pilot.
	VCC	The activist revealed that the solider showed the orders to the pilot.
	Relative	The activist emailed the solider who showed the orders to the pilot.
4.	Matrix	The MBA studied, and the PhD showed the soda to the speaker.
	NCC	The MBA believed the fact that the PhD showed the soda to the speaker.
	VCC	The MBA revealed that the PhD showed the soda to the speaker.
	Relative	The MBA liked the PhD who showed the soda to the speaker.

Handed

1.	Matrix	The chef cooked the sauce, and the waitress handed the chicken to the author.
	NCC	The chef stated the fact that the waitress handed the chicken to the author.
	VCC	The chef revealed that the waitress handed the chicken to the author.
	Relative	The chef knew the waitress who handed the chicken to the author.
2.	Matrix	The supervisor ordered the sheets, and the helper handed the blanket to the marine.
	NCC	The supervisor reported the fact that the helper handed the blanket to the marine.
	VCC	The supervisor revealed that the helper handed the blanket to the marine.
	Relative	The supervisor called the helper who handed the blanket to the marine.
3.	Matrix	The landlady lost the bedding, and the renter handed the pillow to the landlord.
	NCC	The landlady believed the fact that the renter handed the pillow to the landlord.
	VCC	The landlady revealed that the renter handed the pillow to the landlord.
	Relative	The landlady trusted the renter who handed the pillow to the landlord.

4.	Matrix	The coach nodded, and the therapist handed the needle to the athlete.
	NCC	The coach stated the fact that the therapist handed the needle to the athlete.
	VCC	The coach revealed that the therapist who handed the needle to the athlete.
	Relative	The coach contacted the therapist who handed the needle to the athlete.

Promised

1.	Matrix	The architect wanted lunch, and the bricklayer promised the carrots to the builder.
	NCC	The architect reported the fact that the bricklayer promised the carrots to the builder.
	VCC	The architect revealed that the bricklayer promised the carrots to the builder.
	Relative	The architect hired the bricklayer who promised the carrots to the builder.
2.	Matrix	The staff waited, and the employer promised the scissors to the usher.
	NCC	The staff believed the fact that the employer promised the scissors to the usher.
	VCC	The staff revealed that the employer promised the scissors to the usher.
	Relative	The staff admired the employer who promised the scissors to the usher.
3.	Matrix	The teenagers wanted drinks, and the prom queen promised the sandwich to the escort.
	NCC	The teenagers stated the fact that the prom queen promised the sandwich to the escort.
	VCC	The teenagers revealed that the prom queen promised the sandwich to the escort.
	Relative	The teenagers ignored the prom queen who promised the sandwich to the escort.
4.	Matrix	The announcer waited, and the spokesman promised the medal to the scholar.
	NCC	The announcer reported the fact that the spokesman promised the medal to the scholar.
	VCC	The announcer revealed that the spokesman promised the medal to the scholar.
	Relative	The announcer worked with the spokesman who promised the medal to the scholar.

Appendix 3B: Filler items for the lexical priming study

The probe words are shown in the right-hand column. All comprehension questions are shown in italics with the sentence they followed.

	Matrix filler sentences	Probe
1.	The policeman liked the uniform, and the fireman loved the new red truck. *The policeman hated the uniform.*	hated
2.	The woman got ready, and the midwife put the soap by the bowl. *The midwife did not put the soap by the bowl.*	delivered
3.	The magician studied, and the assistant hung only the curtain on the rod. *The assistant hung the painting on the rod.*	waited
4.	The brick layer mixed the cement, and the welder melted the iron. *The brick layer did not mix any cement.*	hit
5.	The bassist found the pick, and the rock star brushed her hair. *The rock star found the pick.*	lost
6.	The snowboarder drank cocoa, and the skier licked the snow cone. *Only the skier drank cocoa.*	sipped
7.	Only the mailman slept in the van, and the courier climbed the stairs. *The courier slept in a van.*	trail
8.	Only the sailor napped, and the diver mended the suit. *The diver napped.*	sailor
9.	The professor ate the warm crumpets, and the dean drank the hot cappuccino. *The professor ate warm crumpets.*	fire
10.	The designer drew a picture, and the model drank the wine. *The designer drew a picture.*	found
11.	The hiker drank water, and the climber ate trail mix all day long. *The hiker drank water.*	soda
12.	The cat ran across the street, and the car hit the tree with a thud. *The car hit the tree.*	truck
13.	The baker kneaded the dough, and the chef stirred the stew. *The chef stirred the stew.*	soup
14.	The widower did not wait, and the monk pinned the cloth to the statue. *The widower didn't wait.*	sewed
15.	The cop wrote the report, and the lab washed the crime scene. *The cop wrote a report.*	lab
16.	The sheriff waited patiently, and the deputy crushed the peanuts. *The deputy crushed some peanuts.*	peanuts
17.	The clown drove a small car, and the cowboy rode the wild American mustang.	waited
18.	The dietitian baulked, and the chef did not wash the fat off the pan.	watched
19.	The vet prepared the shot, and the cat slept peacefully during the operation.	needle
20.	The woman chose the song, and the pianist set the music on the bench.	stirred
21.	The militia retreated, and the humanitarian wiped away the tears.	dough
22.	The parents napped, and the toddler played with the puppy.	woke
23.	The parents found the paper, and the friends wrapped the presents.	pipes
24.	The banjo player drank coffee, and the cowboy stuck the fork in the beans.	ironed
25.	The plumber found the pipes, and the carpenter twisted the screwdriver.	slept
26.	The aunt went to the concert, and the uncle watched the movie.	concert
27.	The censor gasped, and audience loved the performance.	censor
28.	The conductor called the musicians, and the composer placed the score on the stand.	called

29.	The monk polished the silver, and the priest scratched the glass accidentally.	silver
30.	The ship sailed away, and the pirate kissed the parrot sitting on the deck.	parrot
31.	The teacher turned, and the undergraduate slipped through the door.	teacher
32.	The son threw the paper away, and the garbage man picked up the trash.	trash

	Noun complement clause filler sentences	**Probe**
1.	The waitress denied the fact that the senator drank the martini. *The waitress confirmed the fact.*	held
2.	The child doubted the fact that the vet mended the dog's paw. *The child believed the fact.*	paw
3.	The students denied the fact that the lecturer climbed onto the stage. *The students admitted the fact.*	loved
4.	The chef had the belief that only the housewife crushed the garlic. *The chef believed that the husband crushed the garlic.*	had
5.	The astronaut denied the belief that the comet hit the spaceship. *The astronaut confirmed the belief.*	comet
6.	The neighborhood gossip held the belief that only the girl rode the bicycle. *The neighborhood gossip believed that everyone rode the bicycle.*	girl
7.	The swimmer expressed the belief that the lifeguard twisted the towel. *The swimmer expressed a belief about the lifeguard.*	students
8.	The newscaster announced the belief that the nun hit the clown. *The newscaster made the announcement.*	kicked
9.	The barber doubted the fact that the customer watched the final episode. *The barber doubted that the customer watched the final episode.*	believed
10.	The landlady denied the fact that the renter wiped down the walls. *The landlady denied the fact.*	painted
11.	The navigator announced the fact that the captain loved the sea. *The navigator made an announcement.*	navigator
12.	The matador held the belief that the coach hung the cape on the chair. *The matador held a belief about the coach.*	matador
13.	The tattletale announced the fact that the bully licked the lollipop. *The tattletale stated something about the bully.*	lollipop
14.	The princess held the belief that the servant pinned the drapes closed.	scissors
15.	The guitarist had the belief that the drummer scratched the instrument.	bass
16.	The instructor denied the fact that the company put the stock in the market.	president
17.	The barber doubted the fact that the apprentice washed the scissors.	water
18.	The teacher had the belief that the children watched the documentary.	saw
19.	The janitor expressed the belief that the landlord placed the trash in the can.	lent
20.	The storyteller announced that the witch stirred the caldron.	wiped
21.	The toddler denied the fact that the cat slipped passed the mother.	drummer
22.	The waitress doubted the belief that the truck driver wrapped the string around his finger.	placed
23.	The university president doubted the fact that the bursar picked the wrong name.	paper
24.	The mother had the belief that children sleep best on silk.	pan
25.	The performer expressed the belief that the play write kissed the manuscript.	expressed
26.	The child had the belief that elves ate the cupcakes.	child
27.	The paralegal doubted the belief that the mediator set the outline on the table.	paralegal
28.	The newscaster announced the fact that the heat wave melted the ice caps.	ice
29.	The diver expressed the belief that the shark scratched the boat.	boat
30.	The editor did not hold the belief that the proofreader brushed up the writing.	editor
31.	The gossip expressed the belief that the socialite played with her hair.	hair
32.	The tree hugger held the belief that the hunter stuck the knife in the dirt.	hunter

	Verb complement clause filler sentences	Probe
1.	The duchess did not know that the lord ate the warm chocolate pudding. *The duchess knew the lord ate warm chocolate pudding.*	bake
2.	The biker knew that the hiker mended the ripped backpack. *The biker did not know who fixed the backpack.*	kissed
3.	Only the inmate announced that the warden crushed the uprising. *The guard announced that the warden crushed the uprising.*	riot
4.	The newscaster announced that that matador climbed into the stands. *The newscaster made an announcement about the matador.*	newscaster
5.	The miner declared that only the supervisor hit the door with his fists. *The miner stated something about the supervisor.*	hands
6.	The valet knew that the driver hung the keys on the rack. *The valet knew what the driver did.*	crushed
7.	The journal stated that the actor slipped on the ice. *The journal made a statement about the actor.*	fell
8.	The director reported that the stunt double swept the glass. *The director reported something that the stunt double did.*	reported
9.	The film producer announced that the critics watched the director's cut.	rapper
10.	The report declared that the diplomat rode the visiting president's camel.	trusted
11.	The bookkeeper declared that the child rode the tricycle in the store.	climbed
12.	The king reported that the duke put the jewels in the tower.	castle
13.	The cashier stated that the custodian washed the floors with the mop nightly.	helped
14.	The steward reported that the passenger scratched the stewardess on the cheek intentionally.	ranger
15.	The photographer stated that the dictator kissed the guard to thank him.	hugged
16.	The translator announced that the diplomat slept all day long in the hotel.	tower
17.	The quilter stated that the microwave melted the plastic container.	froze
18.	The opposition party declared that the senator played with fire.	ignite
19.	The voice coach declared that the vocalist picked the duet.	whistle
20.	The laborer reported that the landscaper stuck the pitchfork in the mulch.	scratched
21.	The announcer reported that the wrestler pinned the opponent against the ropes.	won
22.	The scientist reported that the mouse drank the toxic chemicals.	scientist
23.	The novelist declared that the publicist loved the new manuscript.	declared
24.	The animator knew that the cartoonist put the sketches in the drawer.	animator
25.	The lumberjack announced that the ranger placed the sign in the path.	sign
26.	The disc jockey knew that the rapper kissed the Grammy.	knew
27.	The novelist stated that the pianist stirred the audience's emotions.	pianist
28.	The inspector reported that the notary set the stamp in the drawer.	report
29.	The pilot reported that the flight attendant wiped up the mess.	pilot
30.	The toddler declared that the babysitter wrapped the blanket around the baby.	blanket
31.	The bully stated that the child licked the spoon.	stated
32.	The electorate stated that the politician twisted the truth.	stated

	Relative clauses filler sentences	Probe
1.	The operator did not call the widower who hung the diploma on the wall. *The operator called the widower.*	glass
2.	Only the mayor condemned the vandal who set the fire in the building. *The governor condemned the vandal.*	apartment
3.	The tutor did not thank the student who melted the butter in the pan. *The tutor thanked the student.*	hammer
4.	The fans loved the referee who swept the plate. *The fans hated all the referees.*	home
5.	The neighbors called the dogcatcher who twisted the rope. *The dogcatcher did not twist the rope.*	nurse
6.	The tailor married the dressmaker who crushed only the roach. *The dressmaker crushed the fly.*	bug
7.	The farmer did not pay the banker who drank the cheap alcohol. *The farmer paid the banker.*	paid
8.	The senator liked the governor who ate the last brownie. *The senator hated the governor who ate a brownie.*	senator
9.	Only the fireman helped the child who climbed the tree. *The policeman helped the child.*	helped
10.	The sheriff noticed the deputy who loved the mountain of paperwork. *The deputy enjoyed all of the paperwork.*	diploma
11.	The man trusted the plumber who hit the pipes with the hammer. *The man trusted the plumber.*	loved
12.	The newspaper interviewed the nun who mended the broken heart. *The newspaper interviewed the nun.*	monk
13.	The foreign correspondent called the translator who licked the stamp. *The translator licked a stamp.*	letter
14.	The beekeeper met the environmentalist who ate the eggplant. *The environmentalist ate the eggplant.*	met
15.	The zookeeper trusted the trainer who kissed the monkey on the head. *The zookeeper trusted the trainer.*	monkey
16.	The children smiled at the butler who wiped up the spilt milk. *The children smiled.*	smiled
17.	The patient thanked the nurse who wrapped the sprained wrist. *The patient thanked the nurse.*	thanked
18.	The consultant helped the man who put the money in the savings account.	owed
19.	The celebrity dismissed the counselor who placed the tabloid in the trash.	wrist
20.	The host noticed the guest who washed the dirty dishes in the sink.	tub
21.	The street sweeper liked the garbage man who scratched the dog.	rat
22.	The weaver knew the potter who slept in the apartment next to the gym.	hired
23.	The doorman visited the renter who slept through the fire alarm.	dishes
24.	The man fired the sleuth who slipped on the stairs.	parents
25.	The politician wrote the loan officer who played with the investments.	nun
26.	The choreographer hired the dancer who picked the expensive costume.	stamp
27.	The seamstress met the quilter who stuck the pin in the cushion.	dog
28.	The parents thanked the lifeguard who pinned the medal on the girl.	lady
29.	The painter loved the reviewer who watched the old video daily.	video
30.	The acrobat loved the clown who rode the purple elephant for ten miles.	miles
31.	The queen thanked the prince who placed the treaty on the desk.	thanked
32.	The clown saw the magician who stirred the children's imaginations.	saw

Appendix 3C: Instructions used for the lexical priming study

Page 1:
Welcome.

In this experiment, you will hear sentences and then be asked to make judgments about them. It is important that you answer as quickly and accurately as possible.

You will also be asked to determine whether a word that appears on the computer screen occurred in the sentence you just heard.

You will see a "+" on the monitor. This is where the word will appear. After you hear the sentence, the "+" will disappear, and the word will appear in its place.

If the word occurred in the sentence, press the key marked "YES."
If the word did NOT occur, press the key marked "NO."

Keep your right finger over the "YES" key and your left over the "NO" key at all times, so you can make quick responses.

Page 2:
On some trials, you will see a statement after the word task. After reading the statement, decide if it is true ("YES") given the sentence you just heard or false ("NO").

If you answer incorrectly, you will see a red "X" on the screen. If you start to miss many questions, slow down and try to listen more closely to the sentences.

There will be three breaks throughout the experiment. If you need to pause, please do so during these breaks.

Page 3:
That's all there is to it. Just to review, this is how the experiment goes:

1. You will hear a sentence.

2. You will see a word and will determine if it occurred in the sentence ("YES") or not ("NO").

3. Sometimes you will see a statement after the word task and will need to determine if it is true ("YES") or false ("NO") given the sentence you just heard.

4. After you respond to the word task or the statement task, the computer will automatically play the next sentence.

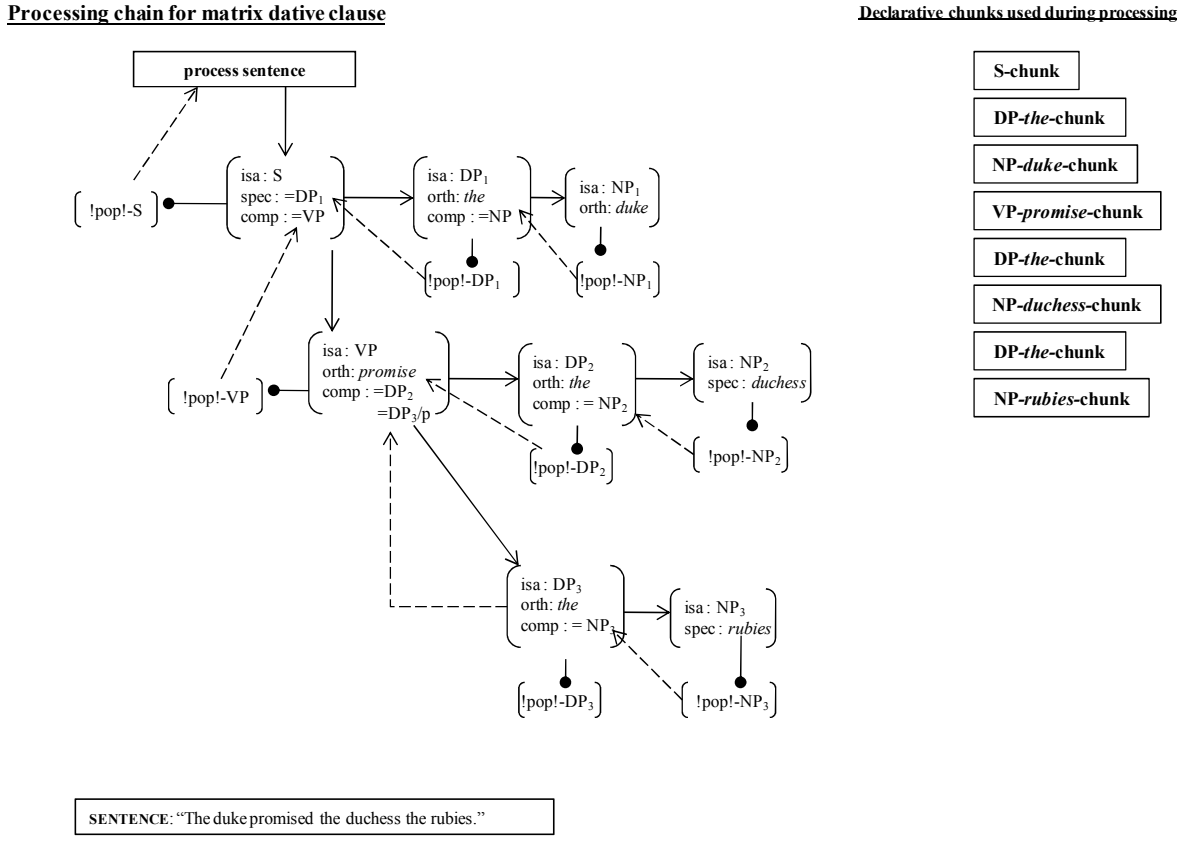
When the experiment is over, a screen will appear telling you to stop. At that point, you should let the experimenter know you are finished.

Again, please answer as quickly but as accurately as possible.
If you have any questions about the procedure, ask the experimenter now.

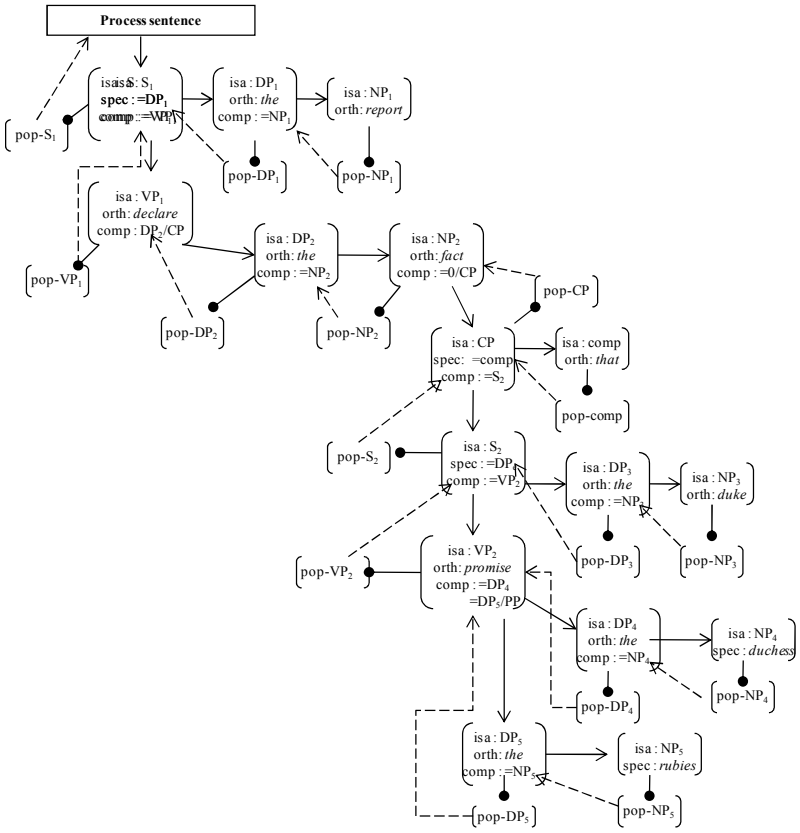
Page 4:
You will now have a few practice sentences.
If you need to change the volume, please do so during the practice slides.

Page 5:
You are now ready to begin the experiment. If you have any questions, please ask the experimenter now.

Appendix 3D: Diagrams of sentence processing and declarative chunks



Processing chain for a sentence with a dative verb in the internal complement of a noun

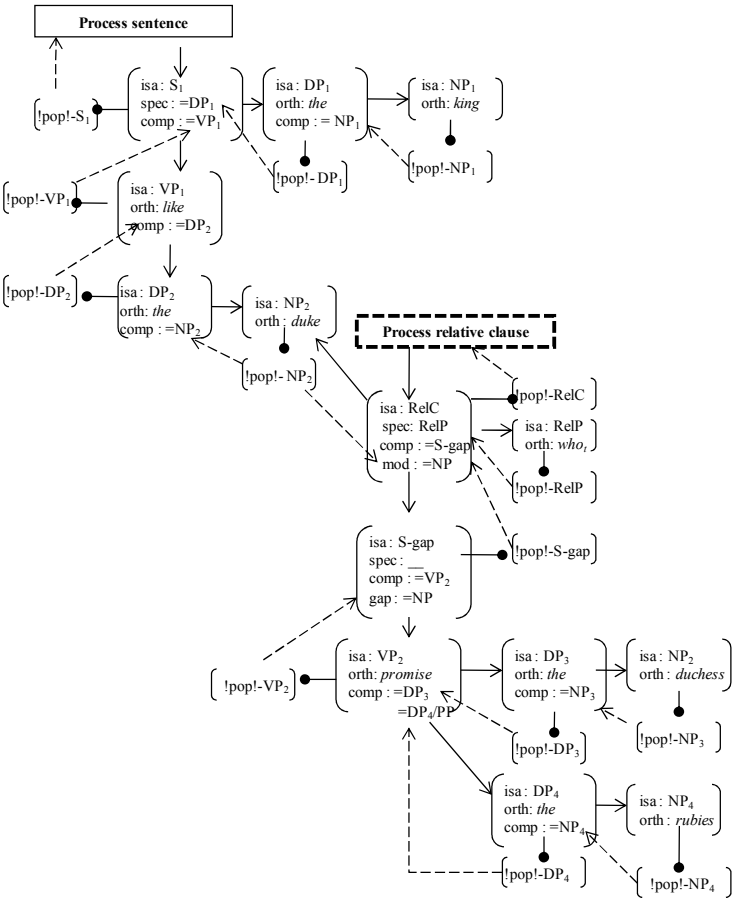


Declarative chunks used during processing

- **S-chunk**
- **DP-*the*-chunk**
- **NP-*report*-chunk**
- **VP-*declare*-chunk**
- **DP-*the*-chunk**
- **NP-*fact*-chunk**
- **CP-chunk**
- **Comp-*that*-chunk**
- **S-chunk**
- **DP-*the*-chunk**
- **NP-*duke*-chunk**
- **VP-*promise*-chunk**
- **DP-*the*-chunk**
- **NP-*duchess*-chunk**
- **DP-*the*-chunk**
- **NP-*rubies*-chunk**

SENTENCE: "The report declared the fact that duke promised the duchess the rubies."

Processing chain for a sentence with an object-modifying relative clause with a dative verb

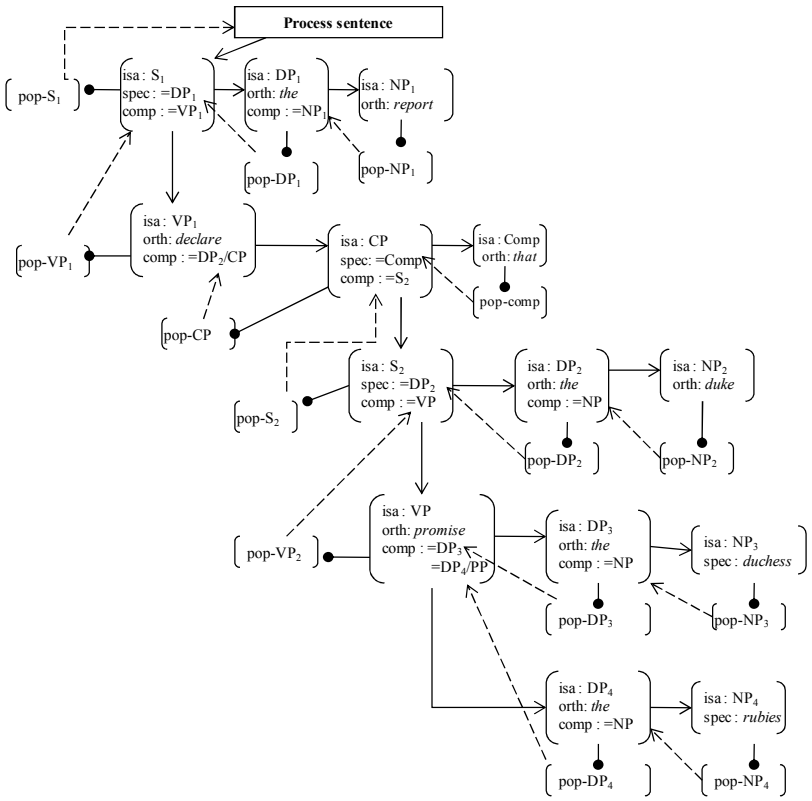


SENTENCE: "The king likes the duke who promised the duchess the rubies."

Declarative chunks used during processing

- **S-chunk**
- **DP-*the*-chunk**
- **NP-*king*-chunk**
- **VP-*like*-chunk**
- **DP-*the*-chunk**
- **NP-*duke*-chunk**
- **RelC-chunk**
- **RelP-*who*-chunk**
- **S-gap-chunk**
- **VP-*promise*-chunk**
- **DP-*the*-chunk**
- **NP-*duchess*-chunk**
- **DP-*the*-chunk**
- **NP-*rubies*-chunk**

Processing chain for a sentence with a dative verb in the internal complement of a verb



SENTENCE: "The report declared that the lord promised the duchess the rubies."

Declarative chunks used during processing

- **S-chunk**
- **DP-*the*-chunk**
- **NP-*report*-chunk**
- **VP-*declare*-chunk**
- **CP-chunk**
- **Comp-*that*-chunk**
- **S-chunk**
- **DP-*the*-chunk**
- **NP-*duke*-chunk**
- **VP-*promise*-chunk**
- **DP-*the*-chunk**
- **NP-*duchess*-chunk**
- **DP-*the*-chunk**
- **NP-*rubies*-chunk**

4 CHAPTER

Structural Priming

> The existence of forgetting has never been proved: We only know that some things don't come to mind when we want them. ~ *Friedrich Nietzsche*

In Chapter Three, I presented evidence that lexical priming was affected by the structural context in which the prime occurred. Lexical primes that occurred in the clausal complement of nouns (henceforth *noun complement clauses*) did not facilitate subsequent recognition as much as primes in matrix clauses, relative clauses, or the clausal complement of verbs (henceforth *verb complement clauses*). These findings suggest that the priming of lexical forms is sensitive to structural context, as claimed by PRICE:

> **Priming According to RICE (PRICE)**
> The processing of both a prime form and its structural context affects how the form is represented, and differences in these representations affect subsequent priming behavior.

In this present chapter, I continue to test PRICE's contention that structural context mediates priming behavior. To do so, I explore another form of priming, STRUCTURAL PRIMING, i.e. the tendency to reuse recently encountered structural forms (Bock 1986b, *inter alia*). The reason for testing structural priming in addition to lexical priming is that the two forms of priming rely on different types of knowledge. Lexical knowledge is often treated as part of declarative knowledge, whereas structure building knowledge (e.g. the knowledge necessary for building a

VP containing a verb and two NP complements) is treated as part of procedural knowledge (e.g. Anderson 2005, Anderson & Lebiere 1998, Bock 1986b). As such, there may be differences in how structural context affects the two types of priming.

In what follows, I present data that demonstrate structural priming's sensitivity to the larger structural context of the sentence in which the prime occurs. Specifically, I demonstrate that structural priming is possible from different structural contexts but that the strength of this priming varies over time for the different contexts. All of the structural contexts that I investigate support priming at short lags (i.e. when there is only one filler item, such as a sentence, between the prime and target). However, after a longer lag (i.e. when there are three filler items between the prime and target), primes embedded in verb complement clauses no longer demonstrated priming. These findings suggest that structural context affects structural priming, contrary to previous claims in the literature (e.g. Branigan, Pickering, McLean, & Steward 2006).

In section 1, I introduce the phenomena of structural priming and situate it within the current conflict between the Standard Account of Priming (SAP), i.e. that the larger structural context doesn't matter, and priming according to RICE hypothesis (PRICE), i.e. that the processing of the larger structural context does matter. In section 2, I summarize the predictions of the two accounts and introduce the experiments meant to test these predictions. This is followed by Section 3, in which I present the first experiment (i.e. priming from various structural contexts after a short lag of one filler item) and a discussion of its results. Section 4 presents the second experiment (i.e. priming from various contexts after a long lag of three filler items) and a discussion of the results as well as a comparison of the results from both

Experiment 1 and 2. Section 5 is a general discussion of the results and their implications followed by conclusions in section 6.

1. Structural priming

Structural priming refers to speakers' tendency to reuse recently encountered structural forms and is often assumed to help ease processing and to lead to long-term changes via implicit learning (Bock 1986b; Bock & Kroch 1989; Bock & Griffin 2000; Cleland & Pickering 2003; Ferreira 1996; Ferreira & Bock 2006; Frazier, Taft, Roeper, Clifton, & Ehrlich 1984; Levelt & Kelter 1982; Luka & Barsalou 2005). For example, speakers are more likely to describe a picture as in Figure 4.1 with a passive-voice sentence (e.g. "The house was struck by lightning") following a passive voice priming sentence as in (1a) than following an active priming sentence as in (1b).

(1a) Passive voice priming sentence
The car was hit by the truck.

(1b) Active voice priming sentence
The truck hit the car.

Figure 4.1: Target picture following passive voice or active voice prime

Structural priming can be found in the absence of shared lexical, phonological, or semantic

features between the prime and target (Bock & Loebell 1990; Pickering & Branigan 1998, 1999). Structural priming occurs even when the prime and target NPs are not equally as complex (Fox Tree & Meijer 1999).[1] It has been associated with a range of linguistic phenomena such as the dative alternation, passive/active voice, *that*-clauses versus infinitival complements, noun phrase structure, and the attachment position of prepositional phrases and relative clauses (Branigan, Pickering, & McLean 2005; Branigan *et al.* 2006; Cleland & Pickering 2003; Desmet & Declercq 2006; Griffin &Weinstein-Tull 2003; Potter & Lombardi 1998; Scheepers 2003). It has been found for both adults and children (Friederici, Schriefers, & Lindenberger 1998; Huttenlocher, Vasilyeva, & Shimpi 2004); in multiple languages such as English, Dutch, German (Desmet & Declercq 2006; Hartsuiker & Westenberg 2000; Scheepers 2003); in both naturalistic or corpus data and in experimental settings (Branigan, Pickering, & Cleland 1999; Dubey, Keller, & Sturt 2008; Gries 2005; Gries & Stefanowitsch 2004; Jaeger & Snider 2008; Levelt & Kelter 1982; Szmrecsanyi 2005; Tannen, 1987; Weiner & Labov 1983); and both in and between modes (Branigan, Pickering, & Cleland 1999; Hartsuiker & Westenberg 2000; Ledoux, Traxler, & Swaab 2007; Pickering, Branigan, & McLean 2002; Zervakis & Rubin 2002). Structural priming occurs regardless of whether the speaker produced the prime him- or herself or simply encountered it in the environment (Bock, Dell, Chang, & Onishi 2007; Boyland & Anderson 1997; Huttenlocher, Vasilyeva, & Shimpi 2004; Thothathiri & Snedeker 2008). It occurs cross-

[1] Fox Tree and Meijer (1999) had participants memorize target sentences and then read priming sentences. Both sentences contained a form of the dative alternation. The prime sentences had noun phrases with different levels of complexity. For example, the prime may have a relative clause as in "The nurse read the most recent letter to the soldier who was wounded" or just an adjectival phrase as in "The nurse read the most recent letter to the wounded soldier." Participants then repeated back the target sentence. The recall of the memorized sentence's alternation was influenced by the priming sentence's alternation. That is, if the priming sentence was a DO, the memorized sentence was more likely to be recalled as a DO than if the priming sentence was a PD. This tendency was not affected by the complexity of the prime sentence's DPs.

linguistically for Spanish-English bilinguals (Hartsuiker, Pickering, & Veltkamp 2004), Dutch-English bilinguals (Desmet & Declercq 2006) and German-English bilinguals (Loebell & Bock 2003) and has even been found in amnesiacs and aphasics (Ferreira, Bock, Wilson, & Cohen 2008; Saffran & Martin 1997). Speakers are even sensitive to the overall frequency of the form (Kaschak 2007; Kaschak & Borreggine 2008; Kaschak, Loney, & Borreggine 2006). Structural priming is truly a robust and ubiquitous phenomenon (for a fuller review see Pickering & Ferreira 2008).

To date, most structural priming models have assumed that the relevant domains for primes are the constituents that comprise the particular alternations (as is explained in greater detail below). Branigan *et al.* (2006) refer to this as the 'local account,' which is captured by what I call the Standard Account of Priming (SAP):

> **Standard Account of Priming (SAP)**
> Having recently encountered a linguistic form increases the likelihood of that form's subsequent reuse.

Recall that the model of language processing presented in Chapter 2 contends that the processing of the structural context affects the way the processor organizes information and, subsequently, the way memory represents information. Differences between these representations ultimately affect the reuse of information.

> **Priming According to RICE (PRICE)**
> The processing of both a prime form and its structural context affects how the form is represented, and differences in these representations affect subsequent priming behavior.

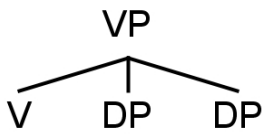
In Chapter 2 section 1, I presented evidence that structural context can affect the recall or

processing of linguistics elements. Specifically, focus constructions such as *it*-clefts and *wh*-clefts can facilitate subsequent processing (Almor 1999; Almor & Eimas 2008; Birch, Albrecht & Myers 2000; Birch & Garnsey 1995; Foraker & McElree 2007; Morris & Folk 1998). Given this evidence, along with the results from the lexical study in Chapter 3, it would be surprising if the structural context in which a structural prime occurs does not influence the efficacy of the prime. More likely is that the way an entire sentence is processed affects the representation of all elements within it (lexical and structural), and this, in turn, affects priming behavior.

1.1 Standard account of structural priming

The majority of structural priming research has implicitly—and occasionally explicitly—assumed a 'localist' account, i.e. that the relevant domain for a particular prime is the domain of the prime itself (e.g. Bock 1986b; Bock & Loebell 1990; Branigan, Pickering, McLean, & Stewart 2006; Kempen & Hoenkamp 1987; Pickering & Branigan 1998). Branigan *et al.* (2006) explicitly argue that the relevant domain for a prime includes only the constituents that constitute the particular prime structure and that changes to the overall sentence structure do not affect the efficacy of a prime within the sentence.

For example, if a structural prime is a VP structure, such as the VP structure below, only the constituents of the VP are relevant.



That is, only the verb and the form of its arguments (the number of and syntactic type of the arguments), if any, are relevant to priming. The co-occurrence of other phrases or clauses not

directly a part of the structural prime are irrelevant. Consider one commonly studied example of VP-level structural priming: the DATIVE ALTERNATION, i.e. the variable ordering of objects following dative verbs such as *give* and *show* (Bock 1986b; Bock & Loebell 1990; Pickering & Branigan 1998, *inter alia*). This alternation allows speakers to describe the same situation using two different linguistic forms:[1] either the DOUBLE OBJECT (DO) form as in (2), where the recipient (underlined) precedes the patient (in italics), or the PREPOSITIONAL DATIVE (PD) form as in (3), where the object precedes the recipient.

(2) Double Object
Seth showed the girl *the comic book*.

(3) Prepositional Dative
Seth showed *the comic book* to the girl.

While there are multiple factors that influence a speaker's choice of a particular alternate such as discourse status, animacy, and pronominal status (Bresnan 2007; Bresnan, Cueni, Nikitina, & Baayen 2007; Bresnan & Nikitina 2009; Doyle & Levy 2008; Green 1974; Oehrle 1976), one key factor is recent experience. If a speaker recently heard or produced a DO form, she is more likely to produce another DO than if she had just recently heard or produced a PD (e.g. Bock 1986b) especially if the prime verb and the target verb are the same (Pickering & Branigan 1998).

According to the localist account (henceforth the standard account of structural priming (SAP)), the domain for structural priming for an alternate of the dative alternation is the dative

[1] Although there is some debate about whether the alternates are truly synonymous (e.g. Gropen, Pinker, Hollander, Goldberg, & Wilson 1989; Pinker 1989; Levin 1993), I treat them as being truth-conditionally equivalent. The reason for doing so is that both alternates have the same truth conditions. For example, if it is true that "Stephen sent Biak the postcard," then it is also true that "Stephen sent the postcard to Biak."

verb and its arguments. Whether the dative verb and its arguments occur in a matrix clause or an embedded clause should not influence structural behavior. The standard account predicts that the double object primes in sentences (4) and (5) below should not differ from the double object prime in (2).

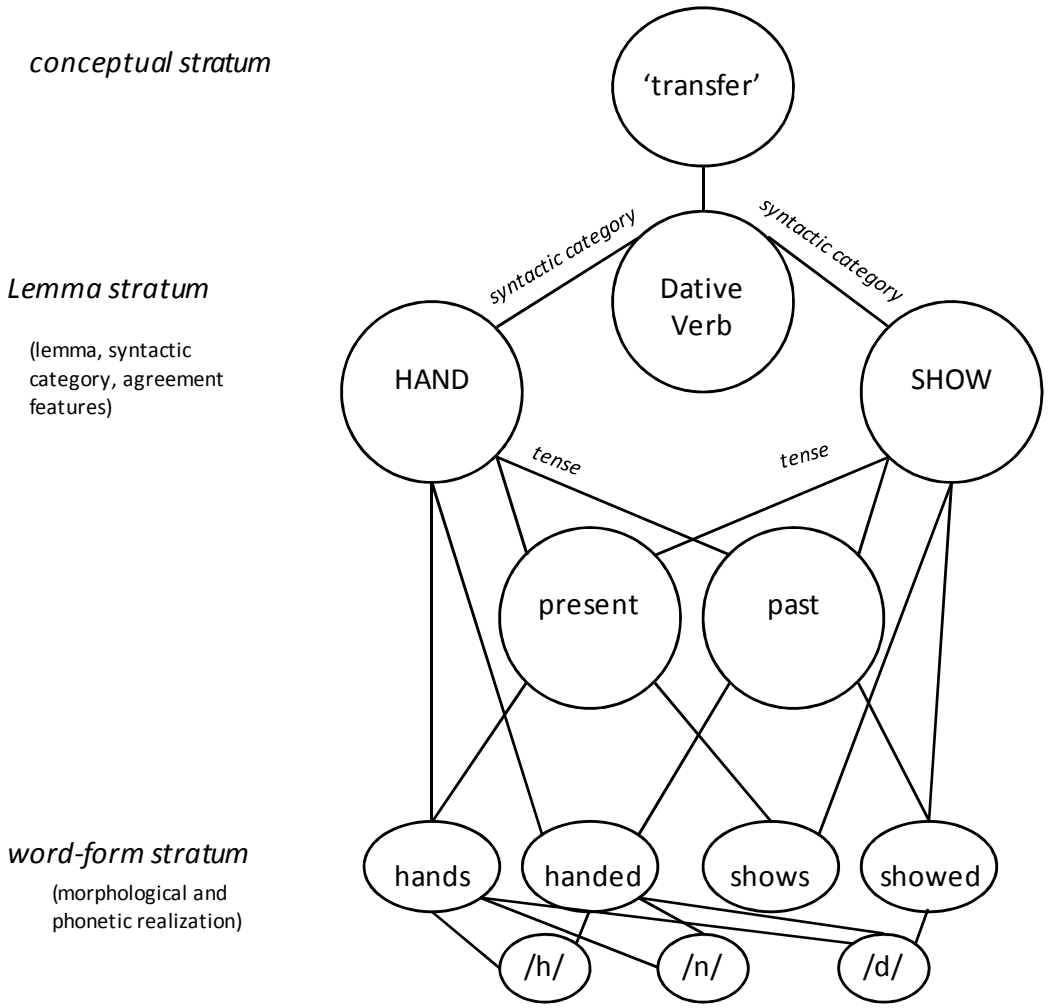
(4) Double object prime in complement clause
Gina knew [that Seth showed the girl *the comic book*.]

(5) Double object prime in relative clause
Gina liked the man [who showed the girl *the comic book*.]

In (2), (4), and (5), the same double object prime ("showed the girl the comic book") occurs in the same linear position (i.e. at the end of a sentence) but in different structural configurations (i.e. as the final verb phrase in a verb complement clause (4) and a relative clause (5)). If the prime is simply the combinatory pattern associated with the double-object construction (V-DP-DP, "showed the girl *the comic book*"), then only the pattern of dative verb, noun phrase, noun phrase is relevant. Factors such as whether the structural prime occur in a verb complement clause or relative clause are irrelevant.

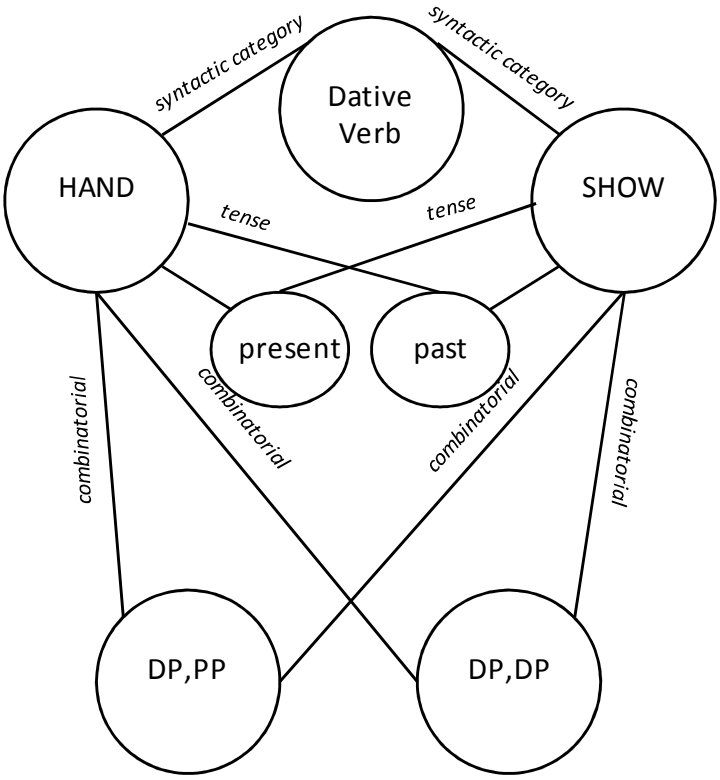
Pickering and Branigan (1998, henceforth P&B) present a standard account of this sort. The P&B model builds off of Roelofs's (1992, 1993) conception of the lexicon, in which there is a network of representations for linguistic forms, such as nouns and verbs. These representations contain information at the word-form (e.g. specific morphological forms such as *hands, handed, handing* and the phonological representation) and the lemma level (e.g. an abstract representation associated with the meaning and the syntactic category such as *HAND-verb*), and the conceptual stratum (e.g. such as the concept 'transfer' associated with HAND).

Figure 4.2: Roelofs's (1992, 1993) model of the lexicon



P&B extend Roelofs's model, adding another layer to the lemma stratum: combinatorial information (e.g. subcategorization frames). Just as there are links between word-form nodes and lemma nodes, so too are there links between lemma nodes and their combinatorial pattern nodes, as shown in Figure 4.3 below.

Figure 4.3: P&B (1998) amended lemma stratum

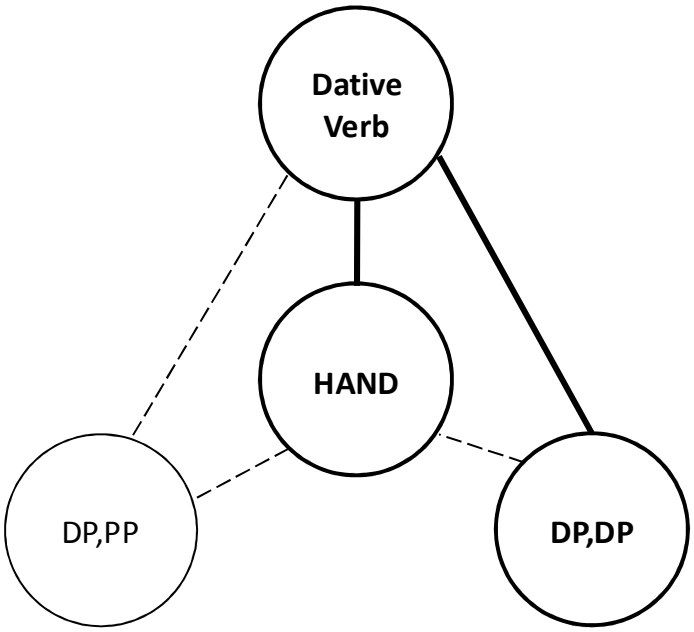


P&B proposes a model of structural priming in which the use (activation) of a lexeme (e.g. *showed*) affects the lemma node (e.g. SHOW) for that particular word form. Likewise, the use of a combinatorial pattern activates the node associated with it. These lemma and combinatorial nodes are linked and can be mutually activated (e.g. by the processing of the phrase "showed the girl the comic book"). For example, following the processing of the sentence (2) (repeated below), the lemma node for the verb *showed* (i.e. SHOW) is activated (as denoted in Figure 4.4 by the bold circle). SHOW's syntactic category, i.e. Dative Verb, is also activated as is the node for the combinatory pattern used in the sentence (i.e. for the DO's

DP,DP) and the links between nodes.

(2) Double Object
Seth showed <u>the girl</u> *the comic book*.

Figure 4.4: The activation of nodes for "showed the girl the comic book"



After processing (2), the category 'Dative Verb' node, the lemma HAND node, and the combinatorial DP,DP node all have heightened activation. As discussed in Chapter 2, section 3.3.2, the activation of a form's representation in long-term memory gives a boost to its activation weight, i.e. the history of use for a given form. This activation boost begins to wane after the form has been used and the processor has moved on to the next form or stage of processing. As the activation wanes (decays), the form's activation weight is still likely to be higher than the weight of its alternates. This heightened activation makes it easier for the

processor to locate the form, and this, in turn, facilitates retrieval, leading to priming effects.

The P&B model of priming assumes that structural priming arises due to the activation of a lemma and the combinatory pattern nodes it associates with. Like other activation-based models, P&B assume that after words are retrieved, they receive a boost in their activation weights and then are subject to a function of activation decay over time. They extend this assumption to structural patterns and combinatorial forms.

The P&B model places a great deal of emphasis on the role of the lexicon and the similarity of structural priming to lexical priming unlike connectionist models of structural priming (e.g. Chang *et al.* 2006). These connectionist models approach structural priming as a form of error-based learning in which the weights associated with producing the different alternates are adjusted during processing. Exposure to the different alternates affect the likelihood of producing a particular alternate by raising the probability associated with the alternate. Thus, having processed a particular form (e.g. the PD dative) makes the processor more disposed to generating the same form. These types of models predict both long-term effects in which the overall frequency of a form's use rises and also short-term effects in which the most recently encountered form affects priming. These two effects can be distinct: the first reflecting changes to the general baseline, the second reflecting the effects of the most recent tuning event.

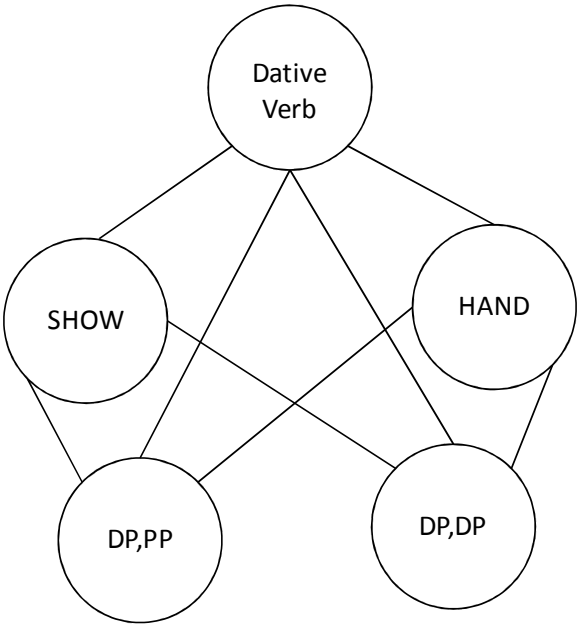
Models such as P&B's focus more on changes within and the activation of the lexicon. Because of their focus on spreading activation through the lexicon and the links between lexical nodes (e.g. *hand*) and combinatorial pattern nodes (e.g. DP,DP), P&B predict a much greater influence of the lexicon than the error-based, connectionist models. Specifically, they predict two

major effects: (i) that repeating the same verb increases priming[1] and (ii) that priming may be short-lived. Their emphasis on repeated verbs means that when the prime and target are the same dative verb, the subsequent priming is greater due to strengthening of retrieval cues. In other words, by reactivating the lemma for the prime word, both the lemma and the combinatorial pattern nodes receive additional activation, compounding the boost from the previous, recent activation.

Although their model emphasizes the role of lexical reactivation, it does not contend that such reactivation is necessary. The primed-for combinatory patterns are connected to other words and have activation weights independent of them. Other, similar words (e.g. other dative verbs) are also linked to the primed combinatorial node. This primed combinatorial node still has residual activation from its recent processing and can, therefore, increase the likelihood of its reuse with other verbs. For instance, consider the representation of the lexicon in Figure 4.5.

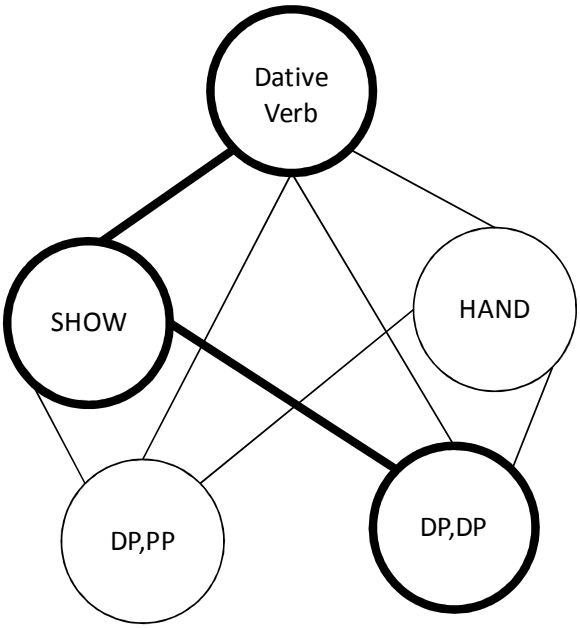
[1] Word repetition has been found to be relevant for priming, particularly lexical and semantic priming (see McNamara 2005 for discussion).

Figure 4.5: SHOW and HAND in the lexicon



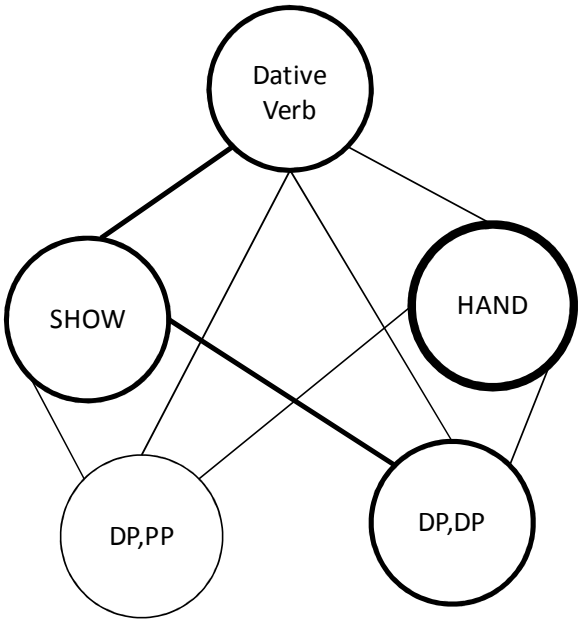
Here we see that the lemma for both SHOW and HAND are connected to the same combinatorial nodes (DP,PP and DP,DP) and the same syntactic form (Dative Verb). After the processing of "showed the girl the comic book," the links are active as shown in Figure 4.6.

Figure 4.6: Activation after "showed the girl the comic book"



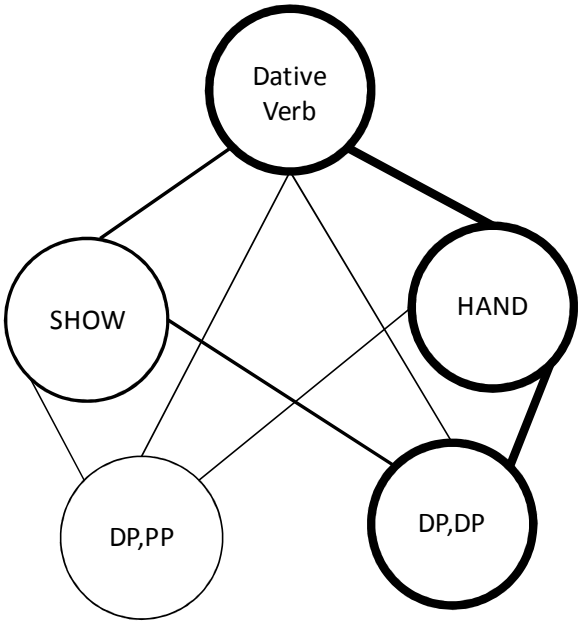
Now, suppose the processor encounters the sentence fragment "Julie handed…." This fragment activates the lemma HAND, as shown in Figure 4.7.

Figure 4.7: Activation of nodes during the processing of "Julie handed…"



Here we see a depiction of the lexicon upon hearing the word *handed.* Note that the HAND lemma is the most active node in the network (as denoted by the thick circle). However, SHOW and its connected nodes (i.e. Dative Verb and DP,DP) all have residual activation from the recent processing event. This residual activation boosts the activation of both the Dative Verb and—crucially—the DP,DP node. The additional activation increases the likelihood of the reuse of the DP,DP combinatorial pattern, as shown by the bolded lines and circles in Figure 4.8.

Figure 4.8: Activation after processing "Julie handed…"



Because DP,DP still has residual activation from the processing of "Seth showed the girl the comic book" (as shown by the lighter black line leading from DP,DP to SHOW) it is more likely to be selected than its alternate form: DP,PP. Although this type of cross-lemma priming is possible, cases in which the same verb serves as the prime and target lead to greater structural priming, as predicted by P&B. This 'repeated-verb' effect has been found in numerous structural priming studies, but it may be a separate contribution of short-term, lexical priming rather than true structural priming because it tends to disappear after one intervening item, e.g. one filler item (Chang *et al.* 2006; Hartsuiker, Benolet, Schoonbaert, Speybroeck, & Vanderelst 2008; Pickering & Branigan 1998). However, important for our discussion of SAP and PRICE is the P&B assumption that only the lemma node and the activated combinatory pattern node are relevant for subsequent structural priming. The larger structural context in which the prime occurs should not affect priming behavior for the target form.

In keeping with the assumptions of the SAP, Branigan *et al.* (2006) found that changes to the 'global' syntax do not affect the behavior of dative primes. By 'global' structure, Branigan *et al.* refer to the "aspects of the sentence that do not form part of the [prime] structure" (p 976). Whether the prime occurs in a matrix clause or a verb complement clause or is preceded by an adverbial clause should not affect priming behavior. They found that structural context did not affect the amount of priming for alternates of the dative alternation. In their experiments, Branigan *et al.* presented speakers with prime sentences in which the prime occurred in either a matrix or an embedded position within sentences displaying various sentence patterns as in (6) – (9).

(6) **Prime in matrix clause**
The racing car driver showed the mechanic . . .

(7) **Prime in matrix clause of a sentence with an adverbial phrase**
On Friday, the racing car driver showed the mechanic . . .

(8) **Prime in matrix clause of a sentence with an adverbial clause**
As the Anne claimed, the racing car driver showed the mechanic. . .

(9) **Prime in verb complement clause**
The report claimed that the racing car driver showed the mechanic…

They found equal priming from matrix clause position for sentences such as (6), matrix clause position of sentences with introductory adverbial phrases and clauses (e.g. (7) and (8)), and embedded positions of sentences with verb complement clauses (e.g. (9)). Because the amount of priming was consistent across syntactic contexts, they concluded that a prime's position in the larger structural context does not matter; only whether it occurs matters. They contend that, although speakers could attend to the larger structural context, they do not need to and,

furthermore, the larger structural context is not likely to affect priming behavior.

This is where the SAP and PRICE differ. PRICE contends that processing one unit interacts with the processing of other units and, thereby, affects priming behavior. Although both accounts make the same predictions for simple, single-clause sentences, such as (2) above ("Seth showed the girl the comic book"), they make different predictions about the effects of larger structural context on multi-clause sentences, such as (4) "Gina knew that Seth showed the girl the comic book."

1.2 The RICE hypothesis and PRICE

A significant amount of work in structural priming supports the general claim that recent encounters with a particular form (e.g. DO datives) exert considerable influence over the next production. For example, the Pickering and Branigan (1998) and the Branigan *et al.* (2006) studies demonstrate the immediate effects of structural priming. Even studies that demonstrate long-term effects of repeated exposure (e.g. Kaschak 2007) have found that the most recently-occurring form exerts additional pressure on the processor. For example, Kaschak (2007) found that participants' baseline use of an alternate were affected by different amounts of exposure to the alternates but that the most recent prime had an separate, extra effect on priming behavior. In his experiment, participants received different amounts of DO or PD completions during a training phase. Then, they received a priming phase in which they completed prime fragments and target fragments. Although their overall use of an alternate reflected differences in the baselines presented during the training phases, the participants' were still sensitive to the most recently-occurring prime type, i.e. they were more likely to produce a DO following a DO prime

than following a PD prime, even if PDs were more frequent in the training phase than DOs were. These findings, along with the findings of Branigan *et al.* (2006), indicate that the most recent encounter is particularly influential, suggesting that recency of experience predicts subsequent performance.

Studies, such as Branigan *et al*.'s (2006), take this recency account a step further by arguing that the effects of recency are insensitive to the larger structural context –that structural context is irrelevant. However, there is evidence that indicates that the larger structural context is relevant for predicting structural priming. Specifically, research with the attachment levels of relative clauses suggests that speakers attend to the global structure of prime sentences and match their productions to patterns larger than the localist account would predict. Scheepers (2003) and Desmet & Declercq (2006) demonstrate this sensitivity to ordering in their studies, which found that speakers could be primed for the attachment level of relative clauses. They argue that participants keep track of not only the phrases or clauses that comprise a sentence but also the structural relations among theses phrases and clauses. Specifically, they found that speakers were more likely to produce a high-attaching relative clause following high-attaching primes than following low-attaching primes. For example, Scheepers (2003) gave native German speakers priming sentence fragments such as (10) and (11) that primed for either a high- or low-attachment. After completing these primes, speakers completed fragments such as (12), which allowed for either a high-attaching or low-attaching completion.

(10) High-attaching prime
Die Assistentin verlas den Punktestand der Kandidatin, der . . .
"The assistant announced the score [mas,sing] of the candidate [fem,sing] that [mas,sing]"

(11) Low-attaching prime
Die Assistentin verlas den Punktestand der Kandidatin, die …
"The assistant announced the score [mas,sing] of the candidate [fem, sing], that [fem,sing]"

(12) Target
Der Rentner schimpfte uber die Autorin der Flugblatter, die …
"The pensioner railed about the author [fem,sing] of the fliers [neut,plur] that [?]"

In primes (10) and (11), the gender of the pronoun prompts either a high-attaching or low-attaching relative clause respectively. Scheepers found that speakers tended to match their productions for targets such as (12) to the prime's form. He argues that because speakers are recreating the attachment levels of relative clauses, they are sensitive not only to particular structural forms but also to the structural position in which these forms occur. Furthermore, he contends that these preferences are not driven by semantics or pragmatics but instead by speakers' sensitivity to larger structural patterns.

Desmet and Declercq (2006) found the same sensitivity with Dutch-English bilinguals. After the participants saw Dutch versions of primes like (10) – (11) above, they completed English target items. Here again, speakers tended to match their production to the form of the prime. High-attaching completions were more likely following high-attaching primes than following low-attachment primes. This tendency to repeat larger, structural configurations suggests that speakers are sensitive to the way phrases are arranged in relation to one another, indicating that the processor tracks more than just the occurrence of a prime.

According to the SAP, structural priming stems from the activation of nodes and residual

activation of these nodes. This activation and subsequent retrievability is independent of the larger structural context. The model of language processing presented in Chapter 2 contends that the larger structural context in which the prime occurs affects priming behavior.

P&B's (1998) account treats structural priming as similar to the type of activation we saw in Chapter 3, where linguistic forms in declarative memory are activated and retrieved. I contend that structural priming involves the activation and retrieval of a certain type of procedural knowledge: production rules (Chapter 2, section 3.3). Structure building is determined by the patterns of rule firing used to process different combinatorial patterns. Specifically, the processor determines which pattern of retrievals, pushings, and poppings is most likely to accomplish a goal (e.g. 'process sentence') given previous experience and the current context. For example, when presented with the goal 'process sentence,' the processor is prepared to fire rules that retrieve DPs, VPs, and NPs. However, when presented with the goal 'compute equation,' the processor is prepared to fire rules that retrieve numbers and operations. Because structure building is a series of product rule firings, structural priming is a form of production rule priming rather than the sort of chunk—or 'node'—priming displayed by lexical priming (see Chapter 3 for discussion).

The processing of a particular combinatorial pattern, such as an alternate (double object, prepositional dative) of the dative alternation, results in a series of production rule retrievals and firings. Because these patterns result from the application of numerous individual rules, the STRENGTH and UTILITY of individual rules play an important role in explaining the distribution of a certain combinatorial pattern (see Chapter 2, section 3.3.2 and Anderson 1993, 1995; Anderson

& Schuun 2000; Lebiere 1998, *inter alia*). A rule's strength reflects a rule's history, i.e. the frequency and recency of its use:

$$S_p = \ln \sum_{j=1}^{n} tj^{-d} \qquad \text{Production strength}$$

The strength of a production rule *p* is determined by how many times it has been used *n* along with the amount of time *t* of its most recent use *j* minus a function of decay *d*. During processing, rules with greater strength are more likely to be retrieved. However, strength is not the only factor the processor considers in the selection process. Utility also affects the likelihood of a rule's use.

Utility is determined by estimating the expected gain associated with a rule minus the cost associated with the use of a rule as expressed in the formula below (Anderson 1993, 1995; Anderson & Schuun 2000; Liebere 1998):

$$\mathbf{U = PG - C} \qquad \text{Utility}$$

P stands for the probability of success, *G* for the value of the particular goal, and *C* for the cost associated with applying the rule. *P* is estimated using the formula

$$\mathbf{P = qr/(1-(1-q)f)} \qquad \text{Probability of success}$$

where *q* is the likelihood that the particular rule has its intended effect, *r* is the likelihood that the use of the rule leads to the completion of the larger goal, and *f* measures the decline in the probability of completing the goal if the rule fails. C is estimated using the formula

$$\mathbf{C = a + b} \qquad \text{Associated cost}$$

where *a* is the cost of using the particular rule and *b* is the cost of using all the subsequent rules needed to achieve the larger goal given the use of the particular rule.

The processor derives the values of *q* and *a* from the rule itself, whereas *r* and *b* must be derived from the expected states and outcomes. To estimate the values of these expected states, the processor considers the processing that has already occurred and the amount of processing that is likely to occur before the completion of the goal. Thus, the rules with the highest utility scores are those with the highest likelihood of success and the lowest associated cost.

In Chapter 2, section 3.3.2, I argued that utility is sensitive to context. The rules associated with processing a context determine how likely a particular rule's retrieval is. As the processing difficulty increases, the likelihood of a particular rule's use decreases. Given this, I propose that structural priming is sensitive both to recency of use—as reflected by a rule's strength—and structural context—as determined by utility.

In what follows, I take the structural context of a prime to be the unification chain that the prime is associated with. Unification refers to the operation that merges the information of two chunks to produce a new chunk that is equally as complex as or more complex than the two unified chunks (Chapter 2, section 3.4). The term 'unification chain' refers to the series of unification events, or 'unification cycles,' that occur in pursuit of completing a goal (Chapter 2, section 3.4.1). The importance of these chains is that they affect the way linguistic forms are represented in and retrieved from LTM. Rather than retrieving just the specific chunks or rules

used in a sentence, the processor retrieves the unification chains generated during the processing of the sentence. The features of these chains (e.g. length) affect the ability of the processor to evaluate their contents and, hence, to use the information within them to estimate the utility of rules (or rule patterns) associated with the chains. The consequence of this is that rules that are associated with some chains have lower utility scores than those associated with other chains. I return to this point in section 5.

1.3 Summary of the SAP and PRICE

SAP and PRICE make different predictions about priming behavior from different sentence types and different structural contexts. According to SAP, only recency matters. Whether the primes occur in main clauses or in embedded clauses or in argument or adjunct clauses should not affect priming behavior. The larger structural context in which a prime occurs does not strengthen or weaken the priming. However, the PRICE account contends that the structural configurations in which primes occur do affect priming behavior.

2. Testing the predictions of the accounts

The experiments discussed in this section explore structural priming behavior from different structural contexts. The experiments varied both the structural contexts in which the primes occur and number of filler tasks between the prime and target. I examined four different sentence types:

i) **Sentences with introductory adverbial clauses and a matrix clause**
As the newspaper noted, the writer was born in January.

ii) **Sentences with verb complement clauses**

The newspaper noted that the writer was born in January.

iii) **Sentences with subject-modifying relative clauses**
The writer who liked Lily was born in January.

iv) **Sentences with object-modifying relative clauses**
Lily liked the writer who was born in January.

These sentence types were chosen to control for the linear position of the prime (i.e. always within the final clause of the sentence) while varying the structural contexts in which the prime occurred. In (i) and (iii), the prime occurs in a matrix clause. In (ii), it occurs in a verb complement clause, and in (iv) it occurs in a relative clause. I used sentences with introductory adverbial clauses (e.g. (i)) and verb complement clauses (e.g. (ii)) specifically to replicate Branigan *et al.*'s (2006) materials, whereas the use of sentences with relative clauses (e.g. (iii) and (iv)) allowed me to explore structural contexts that they did not examine. One reason to explore different contexts, such as relative clauses, is that processing of sentences with adjunct clauses (such as relative clauses) may affect structural priming behavior differently than the processing of sentences with argument clauses (such as verb complement clauses).

In addition to varying the structural contexts, I also varied the number of filler sentences or sentence fragments, as explained in section 2.1.2 below, between the prime and the target. In Experiment 1, only one filler item occurred between the prime and target. In Experiment 2, three items occurred between the prime and target. Using these different intervals allows us to explore the effects of structural priming over time.

2.1. Overview of the experiments

For each of the following experiments, there were a few constant features. All of the

experiments used the same four sentence types (i.e. sentences with an introductory adverbial clause and a prime in the matrix clause, sentences with a prime in a verb complement clause, sentences with a relative clause and a prime in the matrix clause, and sentences with a relative clause and a prime in the relative clause) and the same set of primes, targets, and filler items. The only the factor that varied between Experiment 1 and Experiment 2 was the number of filler items between the prime and target.

In the experiments, participants were exposed to forms of the dative alternation in one of four complex sentences frames.[1] By 'complex sentences,' I mean sentences that contain two clauses where one is structurally subordinate to the other (i.e. one matrix clause and one embedded clause). The four sentence types are shown in (13)-(16). The primed dative alternation is in brackets, the recipient/benefactor (indirect object) is in italics, and the patient (direct object) is in bold. The embedded clause is underlined.

(13) Matrix position with adverbial clause
As the report disclosed, the mother **[**promised *the child* **the ring]**.

(14) Embedded in verb complement clause
The report disclosed that the mother **[**promised *the child* **the ring]**.

(15) Matrix position with relative clause
The mother who knew the neighbors **[**promised *the child* **the ring]**.

(16) Embedded in relative clause
The neighbors knew the mother who **[**promised *the child* **the ring]**.

Thus, the prime alternation occurs in two matrix positions ((13) and (15)) and two embedded positions ((14) and (16)). The prime in the verb complement clause (14) occurs in an argument

[1] Some of the prime sentences were adapted from previous studies such as Bock & Griffin 2000, Pickering & Branigan 1998, and Branigan *et al.* 2006. All of the experimental items are in Appendix 4A.

clause, whereas the prime in the relative clause (16) occurs in an adjunct clause. Recall that according to the model presented in Chapter 2, adjunct clauses form separate unification chains from other elements of the sentence, and because of this, sentences such as (15) and (16), which contain adjunct clauses, have more unification chains associated with them than sentences with argument clauses (e.g. (14)). PRICE claims that differences in the sizes of the unification chains that primes are associated with affect priming behavior. Thus, the presence of argument and adjunct clauses is relevant for predicting priming behavior.

I divided these four sentence types into two experimental conditions that are labeled according to the form of embedding in which the prime occurs in one of the two sentence types within each of the two experimental conditions:

Verb Complement Clause (VC)

Matrix position with adverbial clause
As the report disclosed, the mother **[**promised *the child* **the ring]**.

Embedded in verb complement clause
The report disclosed that the mother **[**promised *the child* **the ring]**.

Relative Clause (RC)

Matrix position with relative clause
The mother who knew the neighbors **[**promised *the child* **the ring]**.

Embedded in relative clause
The neighbors knew the mother who **[**promised *the child* **the ring]**.

Henceforth, I refer to each of these experiment conditions as either the VC or RC condition.

2.1.1 Experimental items

I chose 16 dative verbs based on their collocations with DO and PD structures as

presented in Gries 2005. Gries contends that most dative verbs do not have significant biases, with only 86 out of the 316 dative verbs in his corpus demonstrating a strong preference. When choosing the set of verbs to use, my primary goal was to make sure that there was a wide array of verbs with different biases. However, I avoided verbs, such as *give*, that are significantly biased toward one of the alternates. Of the verbs I chose, *hand, pass, sell* and *throw* tend to take PD completions and *award, offer, show* and *teach* tend to take DO completions. I assigned eight of the verbs to matrix position and eight to embedded positions as shown in (17) and (18) below.

(17) Matrix verbs: *award, buy, feed, issue, lend, pass, teach, throw*

(18) Embedded verbs: *bake, hand, offer, owe, promise, sell, serve, show*

For example, *bake* occurred inside the verb complement clause in the VC condition and in the relative clause in the RC condition. The nouns associated with the primes and the targets did not change across the two conditions, meaning that the external and internal arguments of a given dative verb were consistent. In the VC condition, the verbs in the non-prime clauses for the prime and target sentences (e.g. the introductory adverbial clause or the embedding matrix clause) were taken from Branigan *et al.* (2006) and included verbs such as *declare*, *reveal*, and *report*. For the RC condition, the verbs used in the non-prime clauses of the prime and target sentences included verbs such as *know*, *see*, and *marry*.

Each of the two conditions (VC and RC) had two versions DO-matrix/PD-embedded and PD-matrix/DO-embedded. In these versions, the alternates were categorically associated with a specific structural position. For instance, either DO always occurred in matrix, or PD always occurred in matrix. Thus, there were a total of four versions:

Table 4.1: Versions for VC and RC conditions

Verb Complement Clause (VC)	Relative Clause (RC)
DO-PD	DO-PD
PD-DO	PD-DO

Each participant was assigned to one of these four groups, such that they would see either DO in matrix position (with PD embedded) or PD in matrix position (with DO embedded) in either the VC or RC condition. Participants encountered stimuli corresponding to either the VC or RC conditions as in shown in Table 4.2.

Table 4.2: Example of experimental materials

Verb Complement Clause Condition	
Primes	
Matrix	As the report disclosed, the diver **[**threw *the coach* **the towel]**.
Embedded	The report disclosed that the mother **[**promised *the child* **the ring]**.
Targets	
Matrix	As the paper declared, the pitcher . . .
Embedded	The paper declared that the lord . . .
Relative Clause Condition	
Primes	
Matrix	The diver who dated the trainer **[**threw *the coach* **the towel]**.
Embedded	The neighbors knew the mother who **[**promised *the child* **the ring]**.
Targets	
Matrix	The pitcher who loved the fans . . .
Embedded	The king liked the lord who . . .

The primary tasks in the experiment were to read and complete sentences. In the READ task, participants read aloud whole sentences, which were either primes or fillers. In the COMPLETE task, participants completed experimental and filler sentence fragments aloud using a set of given words. There was always at least one filler item (either a READ or COMPLETE slide) between the prime and the target. Participants saw either one filler item between the prime and target

(Experiment 1) or three filler items (Experiment 2). An example of a prime-filler-target sequence for Experiment 1 is given in Figure 4.9.

Figure 4.9: Example of prime-filler-target sequence

For all the COMPLETE slides (e.g. the rightmost slide marked 'Target' in the above Figure 4.9), the verbs always occurred at the bottom of a list of two other words, which were either noun-adjective pairs (for some of the filler items) or a noun-noun pairs (for some of the filler items and all of the experimental ones). For the experimental COMPLETE slides, the ordering of the noun pairs was counterbalanced between and within subjects. For the filler COMPLETE slides, the nouns pairs or noun-adjective pairs were counterbalanced within subjects.

Previous research has found that the choice between DO and PD alternates is sensitive to factors including discourse features (e.g. the information status) of the two internal arguments of the dative verb, and morphological and phonetic characteristics (Bresnan & Nikitina 2009; Green 1974; Oehrle 1976 *inter alia*). To help control for the effects of discourse status in the primes, all primes had definite NPs for the two internal arguments of the dative verb. I attempted to control for the other aforementioned relevant factors in the targets by giving the participants noun-noun pairs for the experimental target items. These noun pairs were matched on four features:

morphological complexity, segmental length, number of syllables, and frequency.[1]

Primes and targets were matched such that the prime and target verbs and structural appeared in the same structural context (e.g., if a participant read *promise* in embedded position, she completed a target that had *promise* in embedded position). By controlling the features of the nouns and repeating the prime verbs and structural contexts, I tried to create target environments with as much overlap as possible in order to encourage priming while at the same time attempting to control for extraneous factors (e.g. syllable length).

2.1.2 Filler items

The filler items consisted of four sentence types with equal numbers of full sentences and sentence fragments for each type.[2] The same set of fillers was used in all of the studies. The four sentence types were two-place predicates (19), object-control sentences (20), finite clause complements (21), and *where*-clause sentences (21).

(19) Two-place predicate filler item
The couple put the gifts in the closet.

(20) Object-control filler item
The father persuaded the girl to be patient.

(21) Finite clause complement filler item
The clique believed that the substitute was cool.

(22) *Where*-clause filler item
The lawyer knew where the documents were.

Verbs occurring in the filler sentences also occurred in a subset of the filler fragments used in

[1] Ratings for these factors were obtained from CELEX (Baayen, Piepenbrock, & Gulikers 1995). Overall, the direct objects and indirect objects did not significantly differ in frequency, morphological complexity, or segmental length. All of the nouns were matched for number of syllables, ranging from one to three syllables.
[2] All of the filler items are given in Appendix 4B.

the COMPLETE slides (e.g. "The sergeant encouraged . . . recruits/ active/BE"). The fillers were randomized and grouped into 18 blocks of 4 items (either full sentences or sentence fragments). Each block also contained one prime-target pair.

2.1.3 Instructions

Participants were told that they would perform three tasks: reading sentences, completing partial sentences, and taking a memory test at the end of the experiment.[1] This memory task was mentioned to distract participants from the real manipulation and to encourage them to attend to the sentences they were reading and completing. It was, however, never given. Participants were instructed to read the READ slides aloud as accurately as possible. For the COMPLETE slides, they were told they would first see a partial sentence, and then after hitting the space bar, they would see a list of three words. The bottom word in all capital letters was to be the primary verb of the second part of the sentence. They were told that they had to use all three of these words in their completion but could change the tense of the verb or ordering of the words and could add articles or prepositions as necessary. They were further warned that they should not do more than what was necessary.

The participants used a set of training materials to familiarize themselves with the reading and completing tasks. During the practice set, if the participants had questions or failed to use all the words correctly, they were reminded of the instructions. After the training, they began the testing phase, which was recorded for subsequent analysis. Participants were recorded individually in a sound-attenuated booth.

[1] The instructions given to participants are in Appendix 4C.

3. Experiment 1: The short-term effects of structural context on structural priming

Experiment 1 had two primary goals: (i) to replicate Branigan *et al.*'s (2006) findings using a different methodology and (ii) to extend their research to different forms of embedding (i.e. relative clauses). In Branigan *et al.* 2006, participants completed partial sentences that were meant to prime for either a PD or DO completion such as (23) and (24). These primes were immediately followed by a target item that would allow for either completion (e.g. (25) and (26)).

Prime types in Branigan *et al.* 2006:

(23) Matrix PD-inducing prime
The racing driver showed the torn overall . . .

(24) Embedded PD-inducing prime
The report claimed that the racing driver showed the torn overall . . .

Target types in Branigan *et al.* 2006:

(25) Matrix target
The patient showed . . .

(26) Embedded target
The rumors alleged that the patient showed . . .

Over the course of their studies, they varied the structural context of the prime and target such that each pairing was tested. In my experiment, I departed from their design in three significant ways. First, one filler item was placed between the prime and target to minimize the possible contribution of strictly lexical priming while still allowing for some amount of overlap (Hartsuiker *et al.* 2008). Second, I supplied the target nouns. Third, I tested a form of embedding not considered in Branigan *et al.*'s experiment, namely relative clauses. The reason for adding another form of embedding was to test the prediction that different structural contexts (i.e.

argument versus adjunct clauses) lead to different patterns of priming behavior.

A baseline experiment was run to determine whether the target sentences had any pre-existing biases. In the baseline experiment, participants received all of the experimental and filler fragment sentences (as explained in detail in section 2.1.1). They completed these fragments using words listed under the fragments. Responses to the experimental fragments were scored as either DO or PD completions according to the scoring conventions discussed in section 3.2 below. A total of 30 native speakers from the Northwestern University community took part in the study for either partial fulfillment of course requirements or pay. The data revealed a slight preference for PD completions in embedded clauses in both the VC and RC conditions. Given the preferences as presented in Gries 2005, these results were counter to the expectation of a slight DO preference. Overall, speakers were more inclined to complete the fragments using PD completions, both for the matrix primes (49%) and embedded primes (53%). Table 4.3 contains the averages and standard deviations for each cell.

Table 4.3: Percent of PD completions for baselines of target items

	Matrix	Embedded
Verb Complement	49% (0.24)	52% (0.26)
Relative Clause	50% (0.25)	53% (0.22)
Total	49%	53%

3.1 Participants

A total of 123 native speakers of North American English from the Northwestern University community participated for pay or for partial fulfillment of course credit. Participants were randomly assigned to one of four groups corresponding to the four versions of the

experiment (i.e. the DO-matrix/PD-embedded version of the VC or RC conditions or the PD-matrix/DO-embedded version of the VC or RC conditions). Data from 30 participants were used for each of these four groups. Data from three participants were excluded for reasons to be discussed below. Participants proceeded at their own pace, and the entire experiment took under 30 minutes on average.

3.2 Scoring conventions

There were three possible scores for a response: DO, PD, or OTHER. For a target production to be scored as DO or PD, participants had to read the prime sentence correctly, save minor disfluencies for all words but the dative verb. For example, if a speaker misread the experimental verb *owed* as *owned*, the subsequent target response was marked as OTHER. Likewise, in reading the prime, if a participant used the wrong preposition (e.g. *with* instead of *to/for*) or omitted one of the arguments, the subsequent target production was scored as OTHER. If a participant accidentally skipped a prime, the associated target was also scored as OTHER.

If the prime was read correctly, the response was scored as either OTHER, DO, or PD based on the actual target completion. Given the oral nature of the task, subjects would at times correct themselves, either for pronunciation or syntax. Only the final responses were scored. For a completion to count as either a PD or DO, the target verb had to be the main verb of the embedded or matrix completion, and, if it was a relative clause completion, it had to be a subject-relative clause. Non-subject relative clauses were excluded to maintain consistency and because it was not always clear whether subjects were producing PD or DO completions in object-relative clauses.

For a token to count as a DO, the target dative verb had to be followed by two DPs, the first of which could be the recipient or benefactor argument of the dative verb, the second being the patient argument of the dative verb. For a completion to count as a PD, the target dative verb had to be followed by a DP that could be the patient argument of the dative verb followed by prepositional phrase headed by either *to* or *for*. Depending on the verb, the DP could be the recipient or benefactor argument of the dative verb. Using these criteria, the following responses would have been scored as an OTHER, DO and PD respectively.

(27) OTHER response
The fans all loved the pitcher who **[**threw the ball at the coach**]**.

(28) DO response
The fans all loved the pitcher who **[**threw the coach the ball**]**.

(29) PD response
The fans all loved the pitcher who **[**threw the ball to the coach].

The total number of useable responses was 92% of the completions, with OTHER responses constituting 8% of the data (stdev = 0.07). This rate of useable responses is slightly higher than other rates reported in the literature.[1] Three speakers had significantly higher rates of OTHER responses due to skipping slides or generating non-standard responses, suggesting that they were having difficulty with the task. Their data were excluded.

3.3 Analysis

The data from the two conditions (i.e. the RC and VC) were analyzed first independently and then together. Each of these three analyses (i.e. RC, VC, and the full set of data) was analyzed using a generalized linear mixed model regression with subjects and items as error

[1] Branigan *et al.* (2006) used about 90% of their responses, and Bock and Griffin (200) used about 80%.

terms.[1] For ease of presentation, I show the percentages of PD completions. The percentages reflect the number of PD completions against the total number of PD and DO completions (i.e. PD/(PD+DO)) in a given cell. Each participant had two percentages: one for the number of PD completions following PD primes (in either matrix or embedded position depending on the participant's specific condition, e.g. PD-matrix/DO-embedded), and one for the number of PD completions following DO primes.

3.4 Results

3.4.1 Relative clause condition

Two generalized linear mixed model logistic regression models (i.e. a main effects and an interaction model) with contrast coding were applied to the RC data. The contrast coding compared (i) the baseline results to the results following PD primes and (ii) the results following PD prime to those following DO primes. This manner of coding allows us to see the general effect of priming by determining whether DO primes and PD primes differed while also allowing us to see a more specific effect of priming, i.e. whether the PD primes differed from the baselines.

A comparison of models found that the interaction did not improve the fit ($\chi^2(2, N = 90) = 0.07, p = 0.97$), so the results from the main effects model are reported here.[2] Figure 4.10 contains the percent of PD completions for the RC baselines as well as for the PD and DO primes with their standard error bars. Table 4.4 contains the regression results.

[1] This analysis was used to factor out any potential noise due to participants' or verbs inherent biases (e.g. some participants may prefer to use DOs) and the possibility that a participant's earlier responses may affect his or her later responses (autocorrelation).

[2] Appendix 4D contains tables from the various models for Experiments 1 and 2.

Figure 4.10: Percent of PD completions for RC with baseline by Position*Lag 1

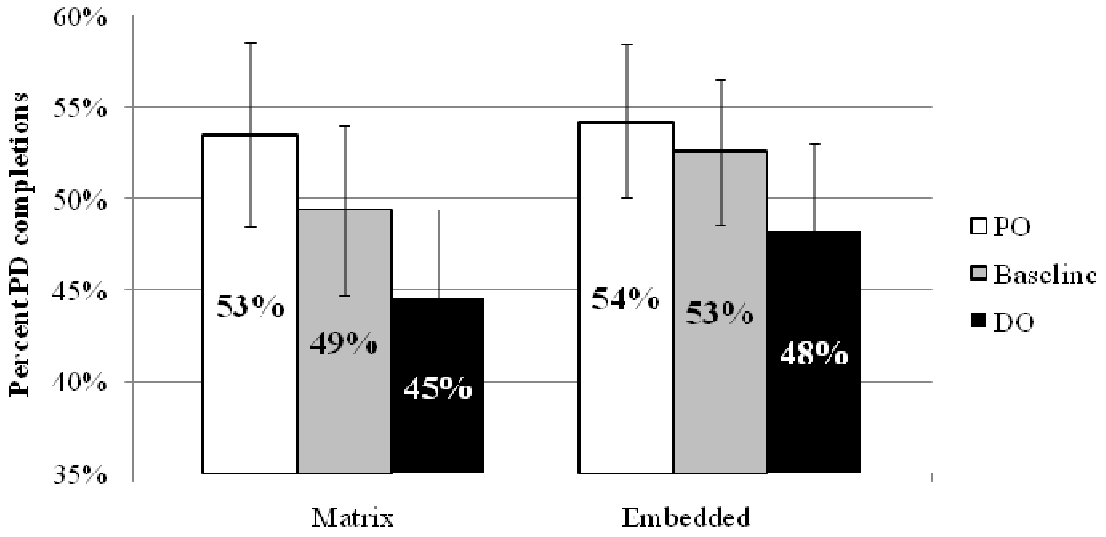


Table 4.4: Regression results RC main effects with baseline at lag of 1

	Estimate	Standard Error	z	P-value
Intercept	0.08	0.28	0.27	0.78
Baseline & PD	0.06	0.17	0.36	0.72
PD & DO	-0.22	0.11	-2.00	0.05*
Position	-0.10	0.37	-0.28	0.78

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

DO primes differed significantly from PD primes (N(90) z = -2.00 $p<0.05$). However, as Figure 4.10 suggests, PD primes did not significantly differ from the baseline (z = 0.36 p=0.72). Furthermore, there was no effect of position (z = -0.28, p = 0.78). Overall, participants were more likely to produce a PD completion following PD primes (54%, stdev = 0.24) than following DO primes (46%, stdev = 0.25). When collapsed across the matrix and embedded positions, the baseline PD productions (51%, stdev = 0.24) did not differ from the collapsed results for either

the PD or DO primes ($z = -1.05$ $p = 0.30$).[1] I return to this result in the discussion. The significance trends were independent of position. Specifically, PD primes led to more PD completions than DO primes in both matrix contexts (following PD primes 53%, following DO primes 45%) and embedded contexts (following PD primes 54%, following DO primes 48%). However, there was no difference among the prime times and the baselines in either position.

These results indicate that, although the PD primes did not differ from the baselines significantly, they did differ from the DO primes, suggesting that the two prime types do lead to priming.

3.4.2 Verb complement clause condition

The same two generalized linear mixed model logistic regressions (i.e. a main effects and a interaction model) with the same contrast coding were applied to the VC data. As with the RC data, the interaction model did not improve the fit ($\chi^2(2, N = 90) = 0.15$, $p = 0.93$), so only the results from the main effects model are reported here. Figure 4.11 contains the percent of PD completions for the RC baselines as well as for the PD and DO primes with their standard error bars. Table 4.5 contains the regression results.

[1] Another generalized linear logistic regression was run with treatment coding to determine whether there was a main effect of prime when the DO primes were compared directly to the baselines. The results from this analysis are included in Appendix 4D.

Figure 4.11: Percent of PD completions in VC with baseline by Position*Lag of 1

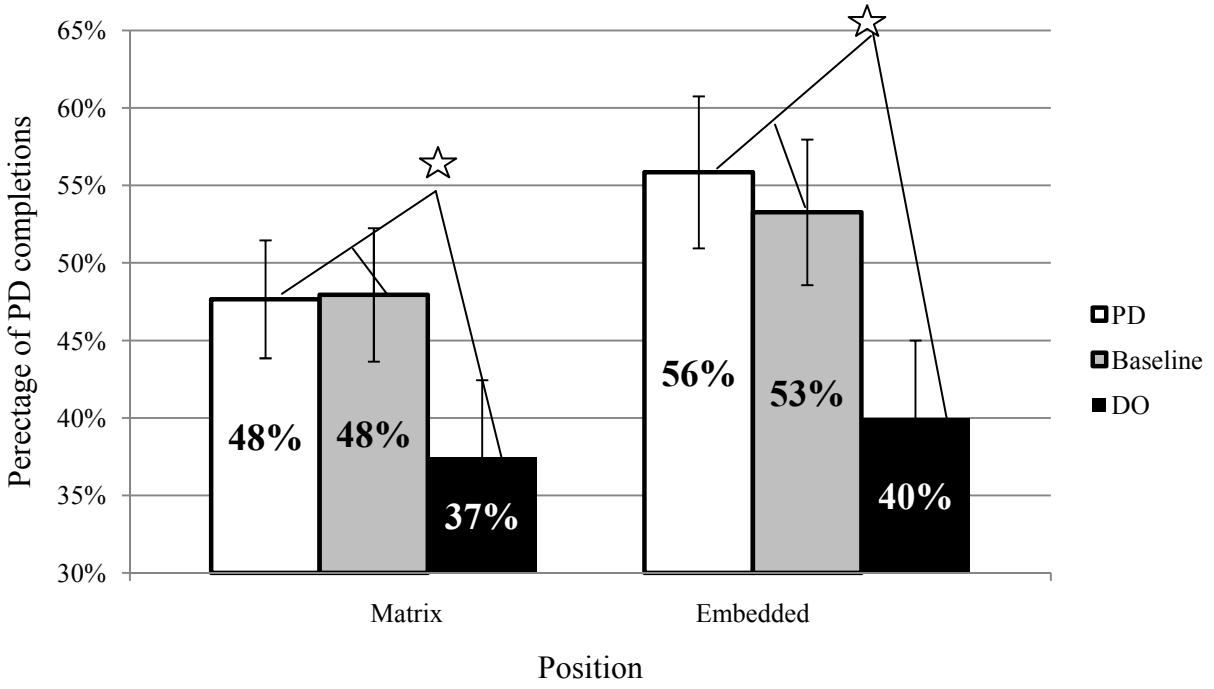


Table 4.5: Regression results VC main effects with baseline at lag of 1

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.26	0.00	0.99
Baseline & PD	0.18	0.17	1.08	0.28
PD & DO	-0.43	0.11	-3.82	0.001***
Position	-0.20	0.34	-0.57	0.57

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

DO primes differed significantly from PD primes (N(90) z = -3.82 p<0.001). However, as Figure 4.11 suggests, PD primes did not significantly differ from the baseline (N(90) z = 1.08 p=0.28). As in the RC data above, there was no effect of position (z = -0.57, p = 0.57). When collapsed over both positions, participants were more likely to produce a PD completion following PD

primes (52%, stdev = 0.26) than following a DO prime (39%, stdev = 0.24). When collapsed across the matrix and embedded positions, the baseline PD productions (51%, stdev = 0.24) did not differ from the collapsed results for the PD, but it did from the DO primes ($z = -2.34$ $p < 0.05$).[1] Thus, there was a lack of significant differences between PD primes and the baseline but a significant difference between PD primes and DO primes and DO primes and the baseline. This overall pattern was repeated in both matrix and embedded position. Hence, the trends were independent of position. Specifically, PD primes led to more PD completions than DO primes in both matrix contexts (following PD primes 48%, following DO primes 37%) and embedded contexts (following PD primes 56%, following DO primes 40%). The difference between baseline primes and DO primes was significant in both matrix position ($t(29) = 2.33$, $p < 0.05$) and embedded position ($t(29) = 1.77$, $p < 0.05$). However, there was no difference among the PD primes and the baselines in either matrix position ($t(29) = 0.01$, $p = 0.57$) or embedded ($t(29) = 0.48$, $p = 0.32$).

3.4.3 Comparing the relative clause (RC) and verb complement clause (VC) conditions at lag of 1

A generalized linear mixed model logistic regression was also run comparing the RC to the VC. These models contained the same contrast coding as the previous, i.e. the PD primes were compared to the baselines then separately to the DO primes. Two models were compared, a single-interaction model (Position*Condition) and model with all possible interactions. The single interaction model allows us to determine whether the two types of embedding led to different patterns of priming behavior. The full interaction model allows us to determine whether

[1] As in the RC data above a separate regression with treatment coding compared DO primes directly to the baselines. See Appendix 4D for the results.

any of the factors interacted to lead to different patterns of behavior. No significant difference between the models was found ($\chi^2(6, N = 180) = 2.31, p = 0.89$), so the results from the single-interaction model are presented below. If there were differences in the amount of priming from relative clauses or from complement clauses or from the two different matrix positions, we would find an effect of condition and/or an interaction. Table 4.6 contains the regression results for the single interaction model. Figure 4.12 contains the percent of PD completions following the PD and DO primes by condition and position. The baseline completions for the RC and VC conditions are not shown for ease of presentation.[1] In this graph, the stripped bars represent the results from the RC condition, and the solid bars represent those from the VC condition. The light-colored bars represent completions following PD primes, and the dark ones represent those following DO primes.

[1] Readers should refer to the RC and VC subsections above to compare the completions following the primes against the baselines.

Figure 4.12: Percent of PD completions for RC and VC by Position*Lag of 1

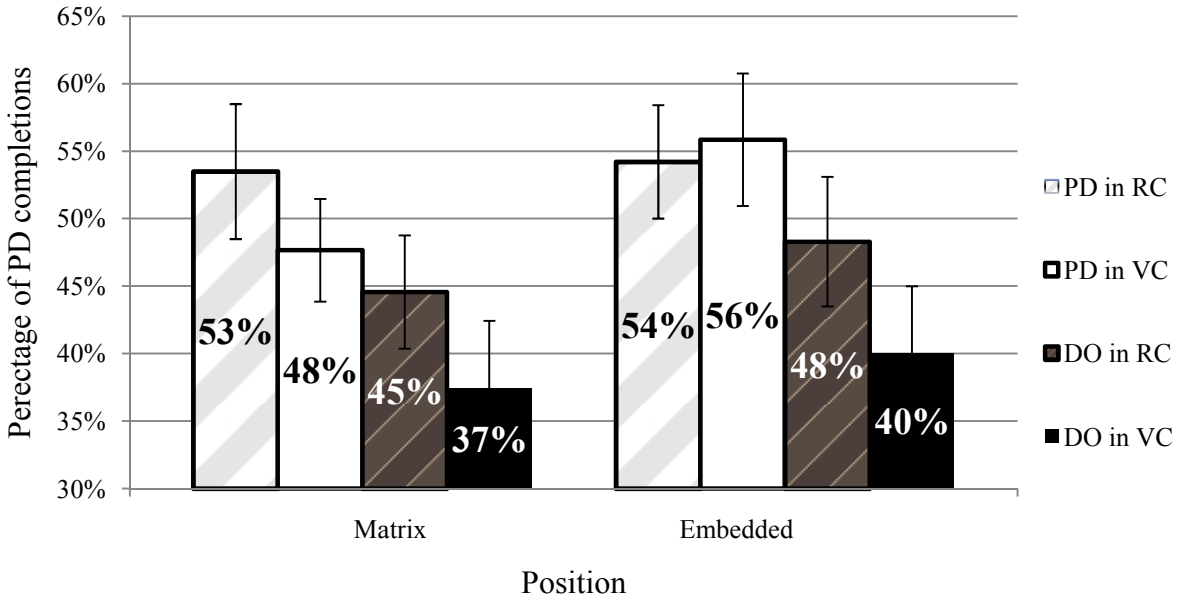


Table 4.6: Regression results from Position*Condition interaction (RC & VC) at lag of 1

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.27	-0.01	0.99
Baseline & PD	0.12	0.12	1.02	0.31
PD & DO	-0.32	0.08	-4.11	0.001***
Position	-0.21	0.35	-0.59	0.56
Condition	0.08	0.18	0.44	0.66
Position*Condition	0.11	0.17	0.67	0.51

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Firstly, there was no significant difference between the PD primes and the baselines (N(180) z = 1.02, p = 0.31). This is the same pattern as found in both the RC and VC conditions in which there was also no significant difference between PD primes and the baselines. However, there was a significant difference between PD primes and DO primes (z = -4.11, p <0.001), indicating that overall there was a main effect of prime. The regression indicates that there was no effect of

position ($z= 0.59$, $p = 0.56$), meaning that being in a matrix or embedded position did not affect the overall pattern of priming behavior. At the same time, there was no main effect of condition (i.e. RC versus VC) ($z = 0.44$, $p = 0.66$). Thus, primes that occur in sentences with relative clauses (i.e. those in the RC condition) did not differ significantly from those that occurred with adverbial introductory clauses or verb complement clauses (i.e. the VC condition). Furthermore, there was no significant interaction among the primes occurring in different positions in the different conditions ($z = 0.67$, $p = 0.51$). These results indicate that PD completions were more frequent following PD primes than following DO primes in all sentence types and from all positions, and no position or sentence type differed from the others in this tendency.

3.5 Discussion

These data suggest that structural context does not affect structural priming at short lags even in the absence of a strictly 'lexical boost' that may have inflated Branigan *et al.*'s (2006) results. Furthermore, these data suggest that priming from different embedded positions may be equal. It appears that the best predictor of a completion for these stimuli is the form of the preceding prime. PD completions were more likely following PD primes than following DO primes regardless of the structural context in which the prime most recently occurred.

One interesting trend to note is that the primed responses did not always differ significantly from the baseline responses. This is particularly noticeable in the RC condition where neither PD nor DO prime completions differed from the baseline though they did differ from one another (i.e. PD completions were more likely following PD primes than following DO

primes). In the VC condition, the DO primes differed from the baseline, but the PD primes did not.

It is not abundantly clear why DO priming appears to be stronger than PD priming in the current experiment. Previous research has also found stronger priming from DO primes (Gries 2005 and Bock 1996, see Potter & Lombardi 1998 for evidence of the opposite pattern). The current pattern may just be in keeping with this general finding. Also, previous research has found a slight bias for DO completions overall. It has also found that when a prime goes against a verb's usual bias (e.g. *give* is highly biased toward DO completions, so a PD prime would go against its bias), there is less significant priming (Gries 2005). Given these observations, it could be that PDs showed less priming due because in general the verbs I chose tend to occur more frequently with DO completions.

However, there are a couple other factors specific to the current materials to consider. One is the primes. The other is the targets. Both of these factors are experiment-specific in that they are sensitive to the materials of the current studies and the relation among agents, recipients/beneficiaries, and patients. Changing these relations may moderate the pattern of priming. The semantics of the prime sentences or the target sentences may have interacted with the priming, creating experiment/material-specific biases. For example, the semantics of a priming sentence (e.g. the semantics related with an act of a mother's promising with her daughter involving a ring) may have led to a preference for one alternate over another (e.g. the prepositional dative "the mother promised the ring to the girl"). The semantics of a sentence can bias toward different completions (Gries & Stefanowitsch 2004), and these biases may have led

to expectation for one completion over the other. If this expectation was violated (e.g. there was a bias for expecting a PD completion but the participant received a DO completion), it may have led to 'inverse priming,' i.e. the tendency for less-frequent or less-expected forms to lead to greater priming (Hartsuiker & Westenberg 2000, Pickering & Ferreira 2008, Scheepers 2003, Jaeger & Snider 2008).

The second source that may have led to greater priming for DO completions could have been the targets themselves. Recall that there was slight preference for PD completions for the target sentences, as noted by the baseline study discussed in section 3.1. This baseline rate for PD completions may have high enough such that it acted as a ceiling. The additional pressure exerted by PD primes didn't affect actual production simply because PD completions where already as high as possible.

That caveat in mind, the productions following PD and DO primes differed. PD completions were more likely following PD primes than DO primes. This tendency was found in each condition's main effects. Priming was not restricted by position or sentence type, meaning that being embedded in a matrix or embedded clause or in a relative clause or verb complement clause did not adversely affect priming. These data replicate Branigan *et al.*'s (2006) results and suggest that SAP may be correct. At short lags, the memory traces for forms of the dative alternation are active enough to prime subsequent behavior, and this priming is independent of—or at least not noticeably affected by—the larger structural context. These results are in keeping with the SAP, which states that structural context should not affect structural priming. However, these results do not provide strong evidence against PRICE.

As mentioned in section 1.2 above, the model of language processing presented in Chapter 2 states that both strength and utility affect the likelihood that a rule (or rule pattern) is retrieved. Strength is determined, in part, by recency of use. At short lags, such as the one used in this experiment, the rules may still be strong enough to affect subsequent behavior. Utility is determined by the estimated probability of success and cost associated with a rule, and this is affected by the number of rules associated with the particular rule of interest. I contend that utility is a function of structural context (see section 1.2 above) and strength is a function of recency. Given this distinction, it is possible that the effects of structural context (as measured by utility) cannot arise until after a longer delay when the effects of recency on a rule's strength have waned. If this prediction is correct, then differences in priming behavior should arise at longer lags.

If SAP is correct and structural context does not affect priming, then primes in all structural contexts should continue to show the same pattern of priming behavior over time. However, if structural context (in particular, the way different structural contexts are processed) affects the way forms are represented in and retrieved from memory, priming behavior should vary among the structural contexts over time, as claimed by PRICE.

4. Experiment 2: The long-term effects of structural context on structural priming

Previous research on structural priming has found stable long-term effects for structural priming lasting up to ten filler items after the prime (Bock & Griffin 2000). However, there is usually a slight decline following one filler item and then perhaps a slight increase (Bock & Griffin 2000,

Hartsuiker *et al.* 2008, Ferreira *et al.* 2008). SAP predicts that if there is any long-term effect of structural priming, it should affect all primes similarly, regardless of the structural context in which the prime occurred. PRICE claims that there may be differences.

Experiment 2 explores the possibility that different priming effects arise from various structural contexts after longer intervals (or 'lags') between the prime and target. As mentioned above, SAP predicts that there should be no variation among the different structural contexts. All primes should demonstrate the same pattern of behavior.

4.1 Changes to the materials and methods

To manipulate the lag in Experiment 2, I redistributed the filler items that occurred within a given block. Recall that in Experiment 1 there were four total filler items per block and one prime-target pair. In Experiment 1, one filler item occurred between the prime-target pair, and three other fillers occurred elsewhere in the block. In Experiment 2, I moved two of the other filler items to occur between the prime and target. However, I kept the filler-target pair that occurred in Experiment 1 the same in Experiment 2, as shown in Table 4.7.

Table 4.7: Block design for Experiment 1 and 2

Experiment One	**Experiment Two**
Filler 1	**Prime**
Filler 2	Filler 1
Prime	Filler 2
Filler 3	*Filler 3*
Target	**Target**
Filler 4	Filler 4

4.2 Participants

One-hundred and twenty-two native speakers of North American English from the

Northwestern University community participated for pay or partial course credit. None of the participants who took part in Experiment 1 took part in Experiment 2. Two participants exhibited difficulty with the task, and their data were excluded from the final analysis.

4.3 Scoring and analysis

The scoring conventions were the same as in Experiment 1. Similar to Experiment 1, OTHER responses accounted for 8% of responses (stdev = .08). All the data were analyzed using a generalized linear mixed model regression with subjects and items as error terms.

4.4 Results

Below, I first consider the results from the RC condition and then compare the results for this condition to the results for the RC condition at the short lag (Experiment 1). This is followed by the same analyses for the VC condition, starting with the results from the current experiment and then a comparison between the VC conditions in short lag (Experiment 1) and the long lag (Experiment 2). After these analyses, I compare the results from the RC and VC conditions of the current experiment (lag 3), and then I compare the results for Experiments 1 and 2 combined.

4.4.1 Relative clause condition

As in Experiment 1, I used generalized linear mixed model logistic regression models (i.e. a main effects and an interaction model) with contrast coding to analyze the data. Two models were compared: a main effects and an interaction model. The interaction did not improve the fit ($\chi^2(2, N = 90) = 0.14, p < 0.93$) , so the results from the main effects model are reported below. Figure 4.13 contains the percent of PD completions for baseline productions PD and DO prime completions. Table 4.8 contains the results from the regression.

Figure 4.13: Percent of PD completions in RC with baseline by position at lag of 3

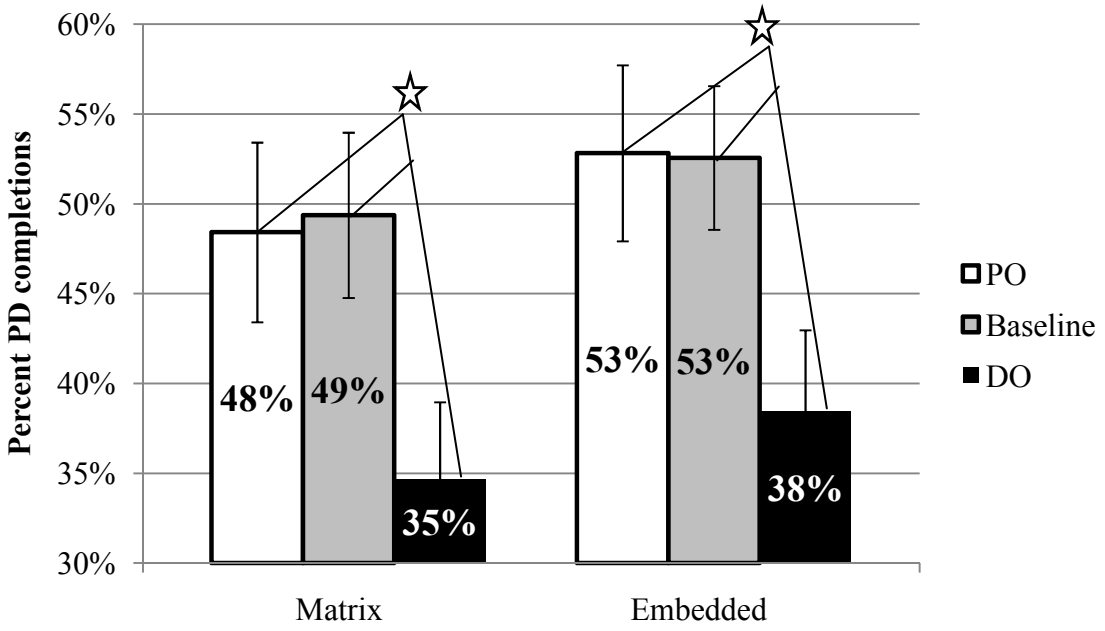


Table 4.8: Regression results RC with baseline for main effects at lag of 3

	Estimate	Standard Error	z	P-value
Intercept	-0.14	0.29	-0.47	0.64
Baseline & PD	0.29	0.18	1.63	0.10
PD & DO	-0.54	0.12	-4.47	0.001***
Position	-0.14	0.38	-0.37	0.72

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

As in Experiment 1, there was a significant difference between PD and DO primes (N(90) $z = -4.47$, $p < 0.001$) but no significant difference between PD primes and the baseline ($z = 1.63$, $p = 0.10$). There was also no effect of position ($z = -0.37$, $p = 0.72$) for the RC data. Overall, speakers were more likely to produce a PD completion following a PD prime (51%, stdev = 0.24) than a DO prime (37%, stdev = 0.27). PD primes led to more PD completions in both

matrix (48% following PD prime, 35% following DO prime) and embedded (53% following PD prime, 38% following DO prime) positions. These differences, i.e. between PD and DO primes, were significant both for matrix primes ($t(29) = 1.92$, $p < 0.05$) and for embedded primes ($t(29) = 2.09$, $p < 0.05$).

Recall that in Experiment 1, I did not find a significant difference between the baseline and the PD primes. The data from Experiment 1 seem to indicate that, DO primes led to more significant priming and that PD primes may have been at ceiling. This pattern of results repeats itself in the current RC data for Experiment 2. Post-hoc t-tests revealed that the DO primes differed from the baseline in both matrix position ($t(29) = 2.25$, $p < 0.05$) and embedded position ($t(29) = 2.14$, $p < 0.05$). However, the PD primes did not differ significantly from the baseline in either matrix position ($t(29) = 0.41$, $p = 0.66$) or embedded ($t(29) = 0.23$, $p = 0.41$).

Comparing lag 1 and 3 for RC: A main effects and two interaction models (a position*lag model and a prime*position*lag) model were tested. Neither the model with a single interaction ($\chi^2(2, N = 120) = 0.11$, $p < 0.74$) nor the model with all the interactions ($\chi^2(4, N = 120) = 3.10$, $p = 0.54$) fit the data better, so the results from the main effects model are presented. Figure 4.13 presents the data from Experiment 1 and 3 for the RC condition. In the figure below, the priming results from the Experiment 1 (lag of 1) are depicted on the left, and the results from Experiment 2 (lag of 3) are on the right. The difference between the gray lines (PD) and the black lines (DO) marks the main effects of priming, and the closeness of the solid lines (matrix position) and the dotted lines (embedded positions) depicts the effect of position or, technically, the lack of an effect. Table 4.9 contains the results from the main effects regression.

Figure 4.14: Percent of PD completions for RC by lag and position

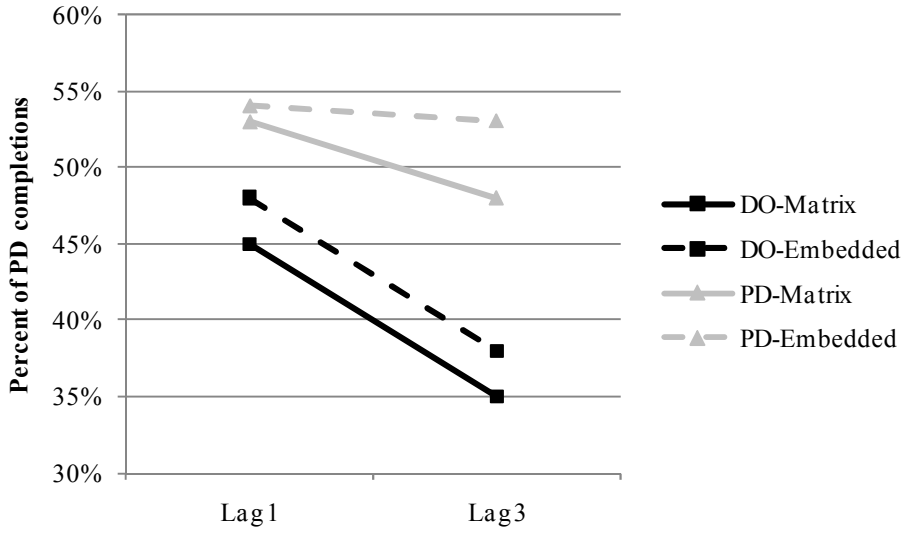


Table 4.9: Regression results RC at lag 1 and 3 main effects

	Estimate	Standard Error	z	P-value
Intercept	-0.04	0.36	-0.14	0.89
Prime	0.58	0.11	5.38	0.001***
Position	-0.14	0.37	-0.38	0.70
Lag	-0.17	0.11	-1.56	0.12

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

There was a main effect of prime (N(120) z = 5.38, p < 0.001). PD completions were common following PD primes than DO primes. However, there was no effect of position (z = -0.38, p = 0.70) or lag (z = -1.56, p = 0.12). Figure 4.23 below depicts the percentage of PD completions for each position at short lag of one filler item between the prime and target (Experiment 1) and the long lag of three (Experiment 2) for the RC condition.

The slope between the DO points on the left and those on the right in Figure 4.13 suggests an increase in priming for DO primes from lag 1 to lag 3. That is, there are fewer PD completions following DO primes at lag of 3 than there were at lag of 1. Subsequent post hoc

analysis found that, although there was no main effect of lag, there was an effect of lag for DO primes ($t(59) = 2.31$, $p < 0.05$) but not for PD primes ($t(59) = 0.65$, $p = 0.52$). When we consider all of the DO primes at lag1 and at lag 3, we find a difference. However, this effect did not arise when each position was analyzed by itself. I return to this effect in section 5 below.

4.4.2 Verb complement clause condition

A comparison of a main effects model and an interaction model found that the interaction model better fit the data ($\chi^2(2, N = 90) = 6.02$, $p < 0.05$, difference in log likelihood = 3.01). The results from this interaction model are presented below. Figure *** contains the percent of PD completions for the VC baselines and the PD and DO primes. Table 4.10 contains the results from the interaction regression.

Figure 4.15: Percent of PD completions in VC with baseline by position at lag of 3

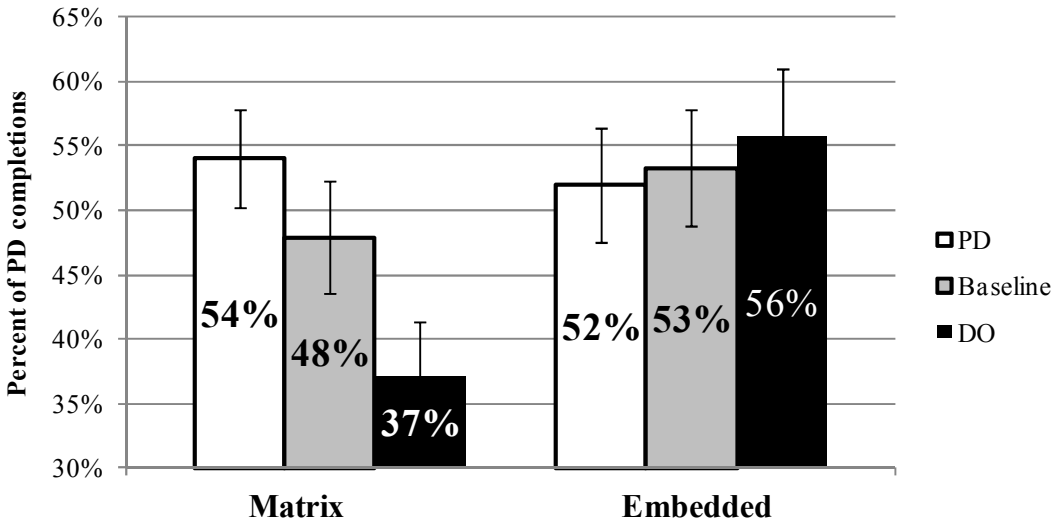


Table 4.10: Regression results VC at lag of 3

	Estimate	Standard Error	z	P-value
Intercept	0.25	0.27	0.94	0.35
Baseline & PD	-0.08	0.17	-0.47	0.64
PD & DO	0.22	0.17	1.30	0.19
Position	-0.36	0.36	-1.00	0.32
Baseline & PD*Position	0.14	0.17	0.81	0.42
PD & DO*Position	-0.68	0.27	-2.51	0.01*

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

This model found no difference between the baselines and completions following PD primes (N(90), z = -0.47, p = 0.64) or between completions following PD primes and those following DO primes (z = 1.30, p = 0.19). Overall, speakers were equally as likely to produce a PD completion following a PD prime (53%, stdev = 0.22) as they were following a DO prime (46%, stdev = 0.26). The model did not find a main effect of position either (z = -1.00, p = 0.32). However, it did find a significant interaction between PD and DO primes and position (z = -2.51, $p < 0.05$). The lack of a priming effect and the significant interaction between prime and position appears to be driven by the position of the prime in an embedded clause in the VC condition. Primes in a matrix clause demonstrate the pattern of priming that we would expect. For matrix positions, PD completions were more likely following PD primes (54%) than following DO primes (37%), and this difference was significant ($t(29) = 2.67$, $p<0.05$). However, primes in an embedded clause seem to show no effect. PD completions were just as likely following a PD prime (52%) as they were following a DO prime (56%) ($t(29) = 0.55$, $p = 0.59$). Furthermore, there was a significant difference between matrix DO primes and embedded DO primes ($t(29) = 2.30$, $P < 0.05$) but no difference between embedded DO primes and embedded PD primes (t(29)

= 0.66, p = 0.75) or matrix PD primes (t(29) = 0.55, p = 0.71).These data suggest that where the prime occurred (i.e. its larger structural context) affected the efficacy of the prime. Specifically, primes in verb complement clauses did not affect subsequent behavior at long lags, but those in matrix clauses did.

Comparing lag 1 and lag 3 for the VC condition: Three regressions were run to compare the results from the VC condition at lag of 1 (Experiment 1) and lag of 3 (Experiment 2): (i) a model for main effects only, (ii) a model with an interaction between position and lag, and (iii) a model for all possible interactions (prime*position*lag). A comparison of models found no significant difference between the main effects model and the single interaction model ($\chi^2(1, N = 120) = 1.21$ p = 0.27). However, there was a significant difference between the main effects model and the model with all possible interactions ($\chi^2(4, N = 120) = 11.33$, p = 0.02, difference in log odds = 5.6). The results from this more complex model are presented in Table 4.11 below.

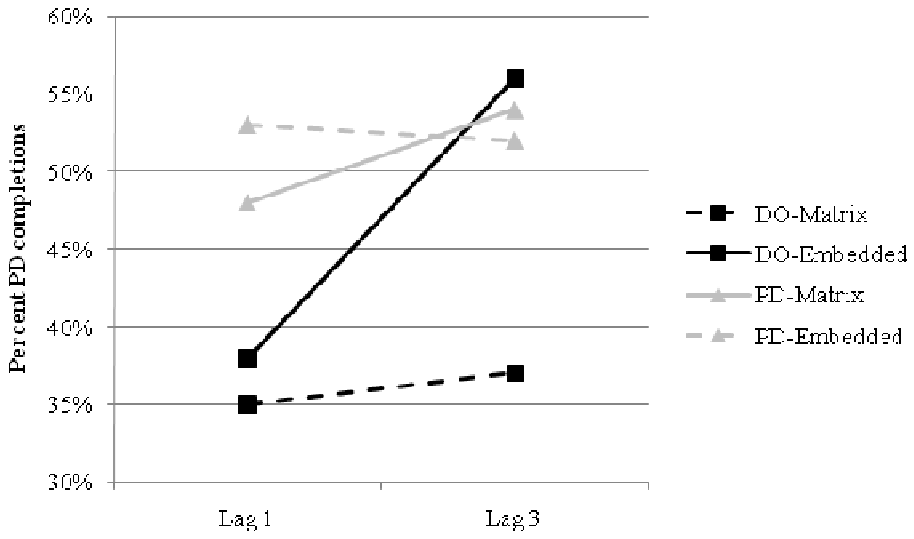
Table 4.11: Regression results VC Position*Prime*Lag at lag 1 and 3

	Estimate	Standard Error	z	p-value
Intercept	-1.04	0.40	-2.58	0.01**
PD Prime	1.44	0.47	3.08	0.01**
Position	0.45	0.58	-0.79	0.43
Lag	0.51	0.14	3.42	0.001***
Prime*Position	-1.04	0.82	-1.27	0.21
Prime* Lag	-0.60	0.21	-2.86	0.01**
Position*Lag	-0.50	0.21	-2.33	0.02*
Prime*Position*Lag	0.75	0.37	2.05	0.04

The regression found a significant effect of priming overall (N(120) z = 3.08, p < 0.01). PD completions were more likely to follow PD (52%) primes than following DO primes (42%). The regression model suggests that position was not significant (z = -0.79, p = 0.43). However, lag was significant (z = 3.42, p < 0.001). In addition, there were significant interactions between

prime and lag ($z = -2.86$, $p < 0.01$) and position and lag ($z = 2.05$, $p < 0.05$). The directions of these interactions become more apparent when we consider Figure 4.16.

Figure 4.16: Percent of PD completions for VC by Prime*Position*Lag



The black solid line (DO embedded position) in Figure 4.16 crosses over the PD solid and dotted lines (PD in embedded and matrix positions) at lag 3. This crossover denotes the lack of priming from DO primes in verb complement clauses. Recall that neither the PD nor the DO primes in embedded positions at lag of 3 differed from the baseline nor did they differ from one another. This suggests that there was no priming from embedded positions at a lag of 3.

To explore this possibility more closely, I ran another regression considering only the DO primes. In previous analyses, the DO primes were the only ones that led to significant differences from the baseline. Earlier, I argued that PD primes may not demonstrate priming because PD completions are already at ceiling and, thus, only DO primes demonstrate priming behavior. By considering only the DO primes, we can more closely attend to possible interactions between position and lag. I used another generalized linear mixed model logistic regression with an

interaction between position and lag. Table 4.12 contains the results from this regression.

Table 4.12 Regression results for DO primes only for Position*Lag at lag 1 and 3

	Estimate	Standard Error	z	P-value
Intercept	-0.96	0.41	-2.32	0.02*
Position	0.35	0.58	0.60	0.55
Lag	0.47	0.16	2.89	0.01**
Position*Lag	-0.46	0.23	-2.00	0.05*

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Here we see that there is a significant effect of lag (N(120) $z = 2.89$, $p < 0.01$). There were more PD completions following DO primes at lag 3 (47%) than at lag 1 (37%). Also, there is a significant interaction between position and lag. For matrix DO primes, the number of DO completions remained about the same for matrix primes across the lags (35% at lag 1, 37% at lag 3), but the number of DO completions following DO primes at lag 3 was significantly lower than at lag 1 (38% at lag 1 56% at lag 3). The fact that there is a significant effect of lag and an interaction between lag and position for DO primes suggests that lag is relevant. I return to this discussion in section 5 below.

4.4.3 Combined results from both conditions in Experiments 1 and 2

Two final series of analyses were run. The first compared the results of the RC and VC conditions of Experiment 2 (lag of 3), and the second compared the compiled results for all conditions from each experiment. These analyses are presented in turn.

RC and VC at lag of 3: A generalized linear mixed model logistic regression was run to compare the RC to the VC. Two models were compared, a single-interaction model (Position*Condition) and a model with all possible interactions. There was no significant

difference between them ($\chi^2(3, N = 120) = 6.58$, $p = 0.09$), so the results from the single-interaction model are presented below.

Figure 4.17 contains the percent of PD completions following PD and DO primes in matrix and embedded positions, and Table 4.13 contains the results from the regression.[1] The striped bars denote results from the RC condition, and the solid bars denote results from the VC condition. The light bars denote completions following PD primes, and the dark bars denote completions following DO primes.

Figure 4.17: Percent of PD completions for RC and VC by Position*Lag of 3

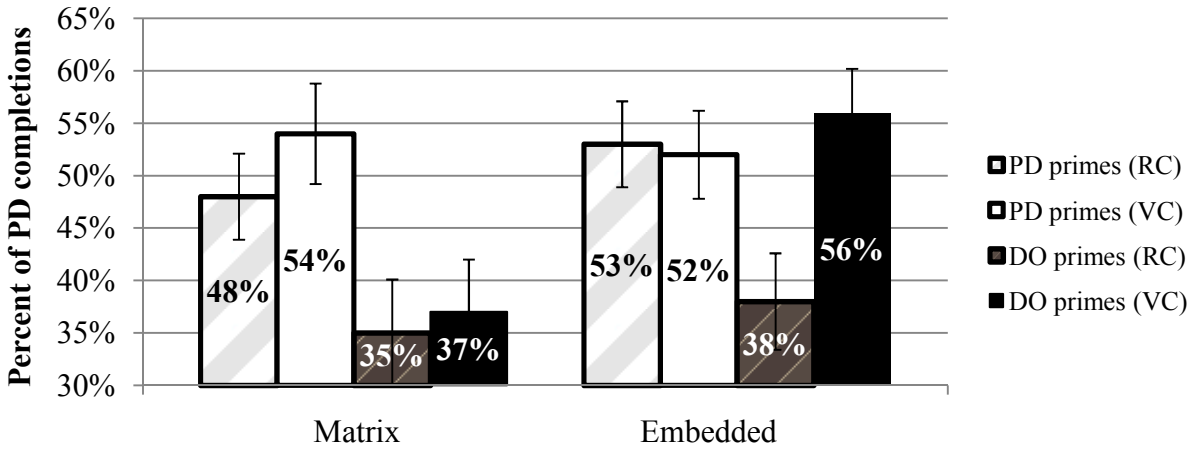


Table 4.13: Regression results from Position*Condition for RC and VC at lag of 3

	Estimate	Standard Error	z	P-value
Intercept	0.26	0.28	0.94	0.35
Baseline & PD	0.13	0.12	1.14	0.25
PD & DO	-0.32	0.08	-3.97	0.001***
Position	-0.35	0.36	-0.97	0.33
Condition	-0.38	0.19	-2.05	0.04*
Position*Condition	0.21	0.17	1.20	0.23

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

[1] The baseline productions for the RC and the VC conditions are not shown for simplicity's sake.

This regression found no significant difference between completions following PD primes across the two conditions and the baselines for the RC and VC sentences. However, there was a significant difference between the PD and DO primes (N(180) $z = -3.97$, $p < 0.001$). There was also a main effect of condition ($z = -2.05$, $p < 0.05$), suggesting that the structural contexts may indeed be significantly different. The data indicate that there were more PD completions in the VC condition (50%, stdev = 0.24) than in the RC condition (44%, stdev = 0.25). This effect is probably driven by the high percent of PD completions in the VC condition.

As Figure 4.30 suggests, the embedded forms in the two conditions differ. Although the interaction did not reach significance, post hoc analysis that compared only the DO primes to DO primes and PD primes to PD primes found that position mattered for DO primes. Depending on the position of the DO prime different patterns of priming behavior surfaced. The difference between the two conditions in embedded positions is significant for DO primes ($t(29) = 2.93$, $p = 0.01$) but not PD primes ($t(29) = -0.17$, $p = 0.87$). However, the difference between them in the matrix position is not significant for either prime type (PD prime $t(29) = 1.08$, df = 29, p-value = 0.29; DO prime $t(29) = 0.68$, $p = 0.50$).

RC and VC at lag of 1 and lag of 3: The final regression compared all the data from Experiment 1 and 2. Both the VC and RC conditions at the short lag and the long lag were combined to see if there are any general trends that arise when the phenomena is considered in its entirety. A comparison of models found that the best-fitting models allowed for an interaction between Condition (RC or VC) and Lag (1 or 3). The difference between the main effects model and the model with only one interaction (i.e. Condition*Lag) was significant ($\chi^2(1, N = 240) =$

3.92, $p < 0.05$; difference in log likelihood = 2.00). Additional interactions did not improve the fit of the models, so the results from the single interaction Condition*Lag are reported in Table 4.14.

Table 4.14: Regression results for RC and VC conditions at lag of 1 and 3

	Estimate	Standard Error	z	P-value
Intercept	-0.48	0.34	-1.44	0.15
Prime	0.54	0.08	7.19	0.00
Position	-0.21	0.35	-0.60	0.55
Condition	0.50	0.33	1.53	0.13
Lag	0.12	0.10	1.14	0.25
Condition*Lag	-0.29	0.15	-2.00	0.05

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

This regression found a main effect of prime (N(240) $z = 7.19$, $p < 0.00$). Overall, there were more PD completions following PD primes (52%) than following DO primes (42%). However, there was also a significant interaction between the condition (RC and VC) and lag (1 and 3 items) ($z = -2.00$, $p < 0.05$). This interaction suggests that the structural context in which a prime occurs interacts with priming behavior but only as a function of time. As was mentioned earlier, DO primes in the RC condition showed an effect of lag, in which all DO primes primed more at longer lags. This increase in priming occurred regardless of the prime's position (i.e. in a relative clause or in a sentence with a relative clause). Lag-based differences were also found for primes in verb complement clauses. However, instead of improving priming (as in the RC condition), the lag undermined priming behavior, leading to an absence of priming at longer lags for primes in verb complement clauses.

4.5 Discussion

The above results suggest that structural priming isn't as context insensitive as previously

assumed. In particular, primes that occur in verb complement clauses do not demonstrate the same pattern of behavior as those occurring in different structural configurations. After a long lag, priming from this position disappeared, though priming from the other positions persisted. SAP claims that all of the primes should have demonstrated similar patterns of priming. If priming disappears from one position, it should disappear from all. If it persists in one, it can persist in all. PRICE contends that primes occurring in different structural contexts should demonstrate different patterns of priming behavior. The above results offer support for the PRICE account of priming. PRICE claims that structural context affects priming behavior by affecting the estimates of a rule's utility (section 1.2). However, this effect is not apparent in the short-term (e.g. lag of 1) due to the boost in strength received from recent activation. We now turn to how the model of language processing presented in Chapter 2 accounts for the results from Experiment 1 and 2.

5. General Discussion

These two experiments indicate that structural priming is affected by the location of the prime within the larger structural context of its containing sentence. It is not the case that encountering a prime in one structural context is equivalent to encountering it in a different structural context. The story is more complex than that. The above results suggest that we need a more nuanced view of structural priming.

Contra to Branigan et al.'s (2006) findings for structural priming, my results indicate that participants are sensitive to the structural context in which structural primes occur. Branigan et

al. found that speakers were just as likely to reuse structural alternates (e.g. DO completions) when the alternate occurred in matrix position or embedded in verb complement clauses. My results indicate that this reuse is sensitive to the amount of time between the prime and the target. As time increases (e.g. from no items between the prime and target to three items), the influence of the prime disappears when the prime is embedded in a verb complement clause. Thus, my results indicate that the processing of the prime and its larger structural context interact.

This supports the general claim made by Scheepers (2003) and Desmet and Declercq (2006) that the processor tracks broader syntactic constructions and is sensitive to differences in 'global structure.'[1] However, my results demonstrate a different form of sensitivity than the one explored by Scheepers (2003) and Desmet and Declercq (2006). In their studies, they found that participants were primed by the attachment levels of relative clauses. If the prime sentence had a low-attaching relative clause, the participant was more likely to produce a low-attaching relative clause than if the prime sentence had a high-attaching relative clause (see section 1.2 for discussion). These findings suggest that the processor tracks the relation among structural units and that subsequent processing of a similar form is facilitated by exposure to a particular relation. My results indicate that the processor is also inhibited by structural contexts and configurations. Reusing particular structural patterns (e.g. DOs) is less likely over time when the structural pattern is associated with structural contexts such as verb complement clauses. Combined Scheepers's, Desmet and Declercq's, and my findings all indicate that the processor is sensitive to the entire structural context of a prime sentence and that the processing of all the

[1] Branigan et al. (2006) use the term 'global structure' to refer to the larger syntactic context in which a prime occurs and the ordering of various phrases and clauses associated with a prime sentence.

forms associated with the prime sentence interact to facilitate some aspects of processing (e.g. the attachment levels of modificational material) but not others (e.g. the reuse of DO or PD forms). In what follows, I clarify how why this interaction occurs.

To begin our discussion, I first review structural priming and its place within the language processing model presented in Chapter 2. I then revisit the processing differences predicted by the model, namely the processing of argument and adjunct clauses. This is followed by a demonstration of how processing affects representations in memory and how this, in turn, affects structural priming behavior.

5.1 Structural priming revisited

In standard accounts, structural priming is explained as a result of node activation (e.g. Pickering & Branigan 1998; Reitter 2008) or as a result of implicit long-term learning (e.g. Bock 1986b, Bock & Kroch 1989, and Ferreira & Bock 2006). In both of these approaches, having processed a structural prime, such as a prepositional dative (as in "gave the book to Sandy"), increases the likelihood of reusing or responding more quickly to another instance of same linguistic form as the prime. However, the reason for this tendency differs.

The Pickering and Branigan (1998) approach treats combinatorial information in the same way as they treat lexical or semantic information. This type of approach focuses more fully on the short-term effects of processing. The implicit learning approach is similar in that it assumes that processing affects the activation weight of a form, but it treats this processing as more substantial than simple transitory, lexical-like priming. Structural priming affects the long-term representations and base-level activation weight of a prime. Kaschak (2007) contends that

these two approaches can account for some of the same data, although the implicit-learning account may be more tenable than the Pickering and Branigan (1998) account as currently stated. Kaschak proposes that there is a long-term effect of processing that affects the overall base rate of a form's use by adjusting, or 'tuning', the weights associated with the form. There is also a separate contribution of the most recent tuning event in that the most recent event that biases the processor toward one completion over the other. This bias may reflect the short-term effects of priming. I return to this distinction in Chapter 5.

As discussed in section 1.2, I contend that structural priming is best understood as priming for procedural knowledge (structure building). Procedural knowledge is represented by a set of production rules. Structural priming is the reuse of recently-used production rules. For example, to decide which pattern (double object or prepositional dative) to use following a dative verb, the processor compares the strength and utility of different production rules (see also Chapter 2, section 3.3.2).

A production rule's strength is similar to a declarative chunk's activation weight, in that it is sensitive to the number and recency of retrievals. Each time a rule is deployed, its strength increases, regardless of whether it led to the successful completion of the goal. Thus, rule strength gives rise to potential recency effects. Rules used more recently have an additional boost to their strengths that—like the activation weight boost for chunks—wanes over time.

A production rule's utility is sensitive to both (i) the probability that the rule successfully achieves its effects and leads to the completion of the goal and (ii) the cost associated with firing the rule and all subsequent rules necessary for the completion of the goal (Chapter 2, section

3.3.2 and section 1.2 above).

To determine a rule's utility, the processor must access previous instances of the rule's use. Upon retrieving an instance, the processor evaluates the rule's success (e.g. if the rule successfully completed its particular goal and if the rule led to the successful completion of the larger goal) and the cost associated with applying the rule. The more processing involved in a particular instance of the rule's use, the more costly the rule is assumed to be. Thus, to determine whether a particular application of a production rule is likely to raise or lower the utility of a rule, we must consider the amount of processing involved during the processing of the prime's containing sentence, in particular the structural context in which the prime occurs. Processing here refers to the number of chunks, rules, and unification cycles (Chapter 2, section 3.4.1) necessary for the processing of the prime's structural context. The structural context is associated with a unification chain generated during processing, i.e. the series of unification cycles used to resolve a single goal (Chapter 2, section 3.4.3). For example, during the processing of a sentence's subject DP, the processor must retrieve a DP-chunk and an NP-chunk, pop them, and unify them. The unification of the NP-chunk unifies with the open =NP value of the DP-chunk counts as a unification cycle. The unification of the resulting DP-chunk with the open =DP value of the S-chunk counts as another unification cycle. Because these two were part of the same subgoal structure, they are part of the same unification chain (see Chapter 2, section 3.3.1 and section 3.4 for depictions of the formation of unification chains).

In the model of language processing presented in Chapter 2, when the processor retrieves sentences from memory, it retrieves the unification chain(s) associated with the production of the

sentence. As the processor evaluates a rule's utility, it considers only the number of rules involved in the formation of the unification chain associated with the rule of interest. Thus, features of the chain, such as length, affect the utility score the processor gives particular rules associated with the chain.

Because structural priming is based on the reuse of production rules, both strength and utility must be considered. Strength is affected by activation and decay. Utility is affected by the cost of applying a rule and the probability of the rule's success. When there are more rules necessary for the completion of a subgoal structure, the cost of applying a rule increases and the likelihood of success decreases. Unification chains reflect subgoal structures, and these subgoal structures reflect differences in structural contexts. When the unification chain is longer due to larger subgoal structures, the individual rules associated with the chain have lower utility scores.

Differences in the strength or utility of a rule can lead to differences in priming. In Experiments 1 and 2 reported in this chapter, I controlled for recency and varied structural context. I found an effect of structural context. Before we can interpret these results, we must understand how the structural contexts differ and how these differences affect the memory traces for primes.

5.2 Determining structural contexts and setting predictions

The primary way in which the prime sentences' structural contexts differ is in the length of the unification chains that they are associated with. Of particular interest to us are the differences between processing primes that appear in adjunct clauses and primes that appear in argument clauses. In Chapter 2, section 3.4.3, I presented a model of processing in which

arguments and adjuncts were processed differently. Specifically, arguments associated with the same unification chain as their selectors whereas adjuncts associated with separate unification chains. Recall that there are three types of production rules (Chapter 2, section 3.3.1):

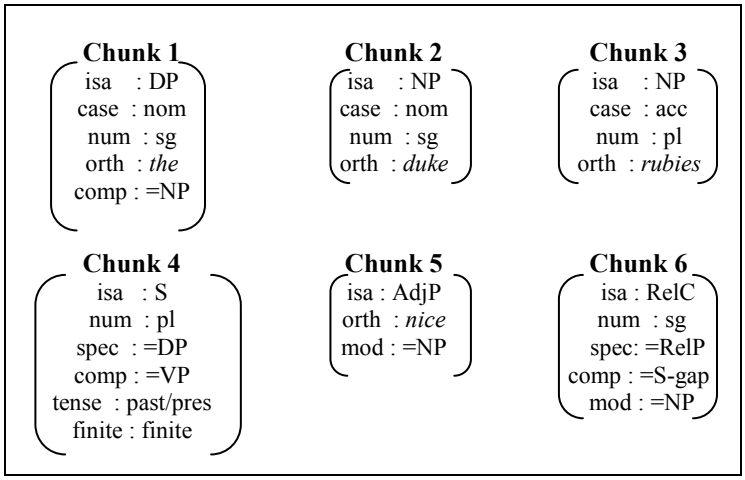
(i) RETRIEVAL RULES find chunks in long-term memory and place them into the retrieval buffer

(ii) PUSH RULES change the problem state by adding new subgoals

(iii) POP RULES remove chunks from the retrieval buffer or subgoals from the problem state

These rules fire based on the state of the buffers. Rule firing is also affected by the accessibility of the rules, as reflected by rule strength and utility. Different patterns of retrieve, push, and pop rules are associated with the processing of different structural contexts. After a 'pop' rule fires, a chunk that was being held in the retrieval buffer becomes available for unification with the chunk associated with the problem state buffer (i.e. the chunk associate with the next subgoal) (see Chapter 2, section 3.2 for a description of the buffers). When the popped chunk's type (e.g. NP) satisfies an open value in the chunk associated with the next subgoal (e.g. the open =NP value of a DP-chunk), the popped chunk and the open value in the subgoal's chunk unify. The product of their unification then pops from the retrieval buffer and becomes available for unification with the next subgoal's chunk (see Chapter 2, section 3.4 and Chapter 3, section 5.2 for a more detailed discussion of this process). As long as the product of unification satisfies an open value in the next subgoal's chunk, the unification cycles continue to add links to the unification chain. As soon as there are no open values in the problem state buffer that a popped chunk can satisfy, the chain ends and is sent to long-term memory (LTM).

The significance of this is that adjunct clauses form distinct chains, whereas argument clauses are part of the same chain as their selectors. The reason that adjuncts are distinct is that when an adjunct is fully processed and popped from the retrieval buffer, its values are not required by the next subgoal's chunk. Consider again the chunks presented in Chapter 2, section 3.3.1.



Note the open values (i.e. anything with the form =XP). Chunk 6 (i.e. the RelC-chunk) requires an NP to satisfy its 'mod : =NP.' Similarly, Chunk 1 (i.e. the DP-chunk) requires an NP to satisfy its 'comp : =NP.' However, nothing has an open =RelC chunk, whereas the S-chunk requires a DP-chunk. Because nothing requires a RelC, it cannot be unified with any particular chunk. On the other hand, because a DP-chunk is required by other chunks' open values (e.g. the open =DP value of the S-chunk), the DP-chunk can be unified with other chunks after its open values are resolved. In contrast, once the processing of a relative clause is complete, the unification chain associated with the relative clause ends and is sent to LTM. However, the unification chain associated with the processing of a verb complement clause such as the

underlined clause in "The king knew that the duke promised the duchess the rubies" is part of the unification chain associated with the verb complement clause's containing sentence. Recall that PRICE contends that long unification chains lead to weaker priming than short unification chains. If this I correct, then there should typically be weaker priming from primes associated with sentences with argument clauses (e.g. those with verb complement clauses) than those contained in sentences with adjunct clauses (e.g. relative clauses).

5.3 The effects of processing structural contexts on structural priming

The model of language processing presented in Chapter 2 states that a rule's retrieval is sensitive to its strength and its utility (Chapter 2, section 3.3.2). The 'structural context' of a prime is taken here to be reflected by the unification chain that the prime is associated with. The length of these chains is determined by the differences in the subgoal patterns associated with the processing of different sentences. These differences map onto differences such as the argument/adjunct distinction. When the processor estimates strength, it retrieves the unification chains and their associated production rules and determines the amount of decay between the creation of the chain and the current processing event. To determine the utility of a rule, the processor retrieves the unification chains associated with the rule and determines the cost and probability of success for the particular rule given the number of other rules also associated with the unification chain.

The size and number of elements in the unification chains are relevant. The model of language processing presented in Chapter 2 predicts that primes that are associated with longer unification chains exert less influence over subsequent behavior than those in shorter chains.

Consequently, structural primes in configurations such as verb complement clauses are expected to not exert as much influence as primes in configurations such as the primes in relative clauses or matrix clauses. The reason is that the unification chains associated with sentences containing argument clauses, e.g. verb complement clauses, are typically longer than those generated by sentences containing matrix clauses or adjunct clauses.

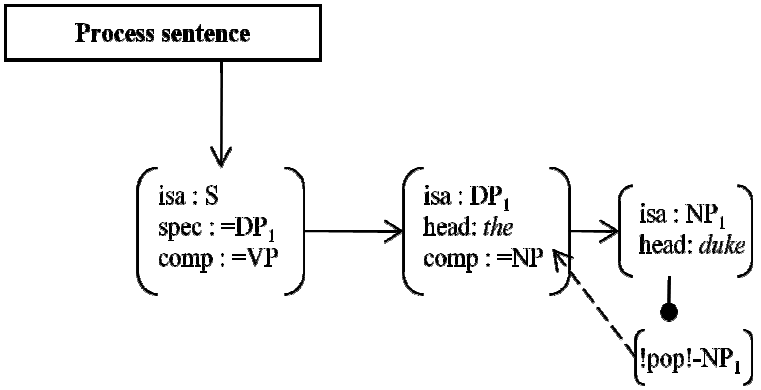
To demonstrate the role of unification chains in explaining the sensitivity of structural priming to syntactic context, let us start with how a chain for a matrix clause containing a double object prime (underlined) develops.

(29) Prime in matrix clause

The duke promised the duchess the rubies.

Multiple chunks must be retrieved, pushed, and popped in order to process this sentence. Each of these actions is associated with a rule. These rules form a pattern that is ultimately represented in memory along with the specific chunks used during the sentence's processing.

In the demonstrations below, as in the demonstrations in Chapter 3, section 5.2, I adopt the following notational conventions. In this and all subsequent demonstrations, the processing goals are shown in a box, whereas all the retrieved chunks appear in brackets. Solid arrows (→) denote the application of a production rules that retrieves a chunk, terminal buttons (↓) denote the application of a production rules that pops chunks, and dashed arrows (↖) represent unification operations. The diagram below shows the steps used to retrieve an S-chunk, then a DP-chunk (→), then an NP-chunk (→), followed by the popping (↓) of the NP, and its unification with the open =NP value of the DP-chunk (↖).



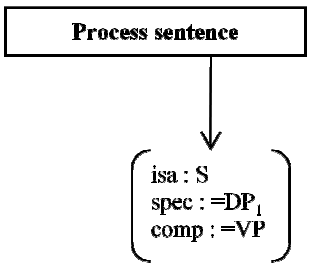
The right-hand column keeps track of all the rules used thus far. Unification operations are shown in bold italics. Rules are in normal font. The syntactic constituents (NP, VP, etc.) are numbered strictly for the purpose of explaining the processing of the different sentences that we consider in this section. For example, the NP contained in the subject DP is $NP_{1,}$ and the NP contained in the first DP unified with the VP is NP_2. After I step through the processing of each sentence type considered in Experiments 1 and 2, I present a table with all the production rules for the unification chains generated during a sentence's processing. The number of columns reflects the number of chains associated with a sentence.

Retrieved rules
retrieve S-chunk
push S-chunk
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP with =NP of the DP

For the sentence in (29) above (i.e. "The duke promised the duchess the rubies"), the processor begins with a goal, i.e. 'process sentence,' which stays in the control buffer until the

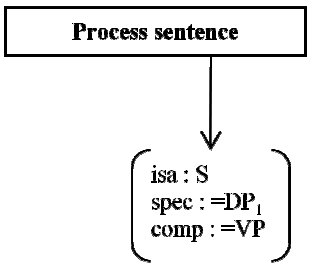
sentence is fully processed:

The processor notes this goal, checks the buffers, finds them to be empty, and selects a 'retrieve S-chunk' rule:



retrieve S-chunk

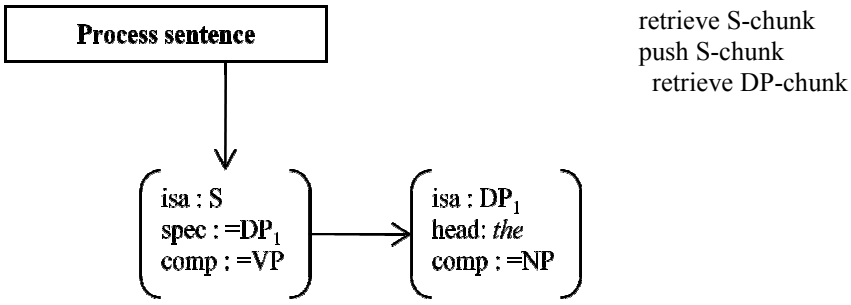
The S-chunk has two open values: =DP and =VP. Because there are open values, the processor determines that the chunk is incomplete, so it selects the rule 'push S-chunk' rule, which moves the DP-chunk from the retrieval buffer into the problem state buffer (Chapter 2, section 3.3.1).



retrieve S-chunk
push S-chunk

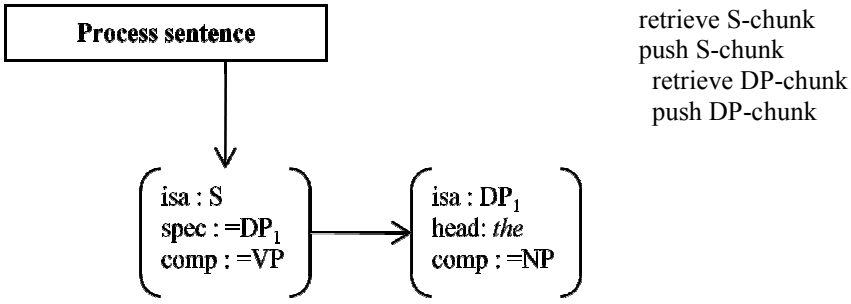
The processor notes the state of the buffers. The problem state has a chunk with two open values (i.e. =DP and =VP), which generate two subgoals ('process DP' and 'process VP'). There is not currently any chunk in the retrieval buffer, so the processor selects a rule that should ultimately

lead to the successful completion of one of the two subgoals. In this case, it selects a 'retrieve DP' rule.



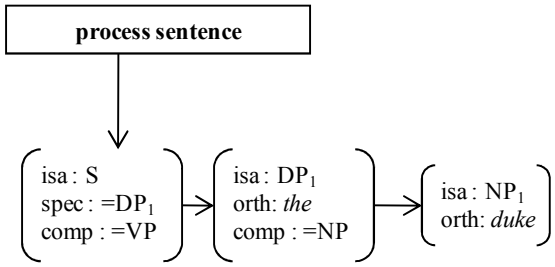
Note that the 'retrieve DP' rule in the right-hand column is indented. This means that the 'retrieve DP' rule was selected in an attempt to resolve an aspect of the chunk in the problem state buffer. In this case, the 'retrieve DP' rule resolves the open =DP value of the S-chunk. However, before it the DP-chunk can unify with the open =DP value of the S-chunk, the processor must verify that the DP-chunk does not have any open values.

The DP-chunk, like the S-chunk before it, has an open value, so the processor pushes the chunk into the problem state buffer.



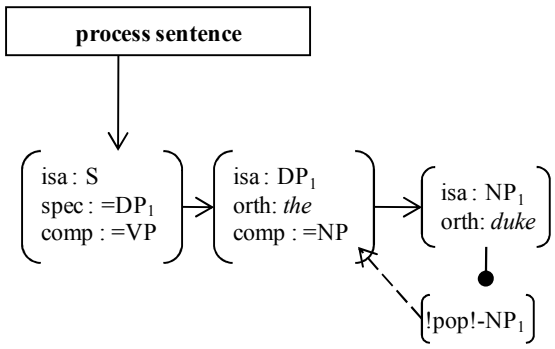
The processor again checks the contents of the buffers, notes the open =NP in the problem state's DP-chunk (i.e. the new subgoal to 'process NP') and the empty retrieval buffer. Given this, the

processor selects a ‘retrieve NP’ rule, which retrieves the NP-*duke*-chunk.



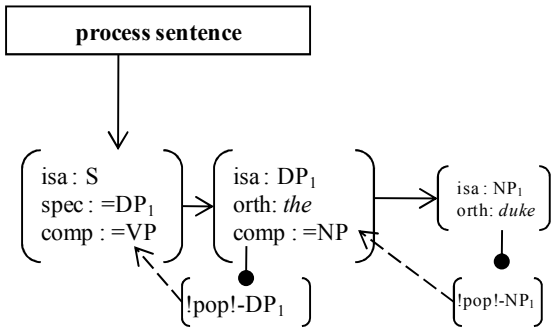
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk

The NP has no open values, so the processor selects a ‘pop NP’ rule. At this point, the NP-chunk becomes available for unification. The chunk that was associated with the next subgoal (i.e. the DP-chunk whose subgoal was ‘process NP’) has an open =NP value, which matches the type of the popped chunk, so the NP-chunk and the open =NP value of the DP-chunk can unify.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP$_1$ with =NP of DP$_1$

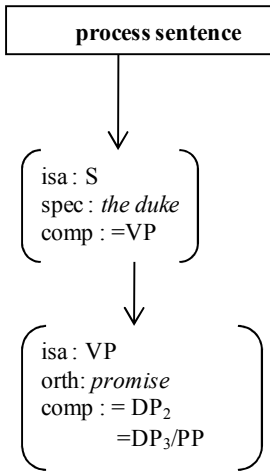
The values of the DP-chunk are filled, so the processor selects the ‘pop DP’ rule, making the DP available for unification. The DP-chunk matches the open values in the chunk associated with the next subgoal (i.e. the S-chunk’s open =DP value and ‘process DP’ subgoal), so the DP and the open =DP value in the S-chunk unify.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP$_1$ with =NP of DP$_1$
  pop DP-chunk
unify DP$_1$ with =DP of S

In the depiction above, the reduction of the font size of the NP-*duke*-chunk denotes the onset of the chunk's decay. Henceforth, I do not show chunks that have been popped and unified.

The processor has satisfied the DP subgoal, so it turns to the next subgoal ('process VP'). The processor notes this goal in the problem state and the empty retrieval buffer, so it selects a rule likely to help resolve this subgoal. In this case, it selects a 'retrieve VP' rule and retrieves the VP-*promise*-chunk and places it in the retrieval buffer.

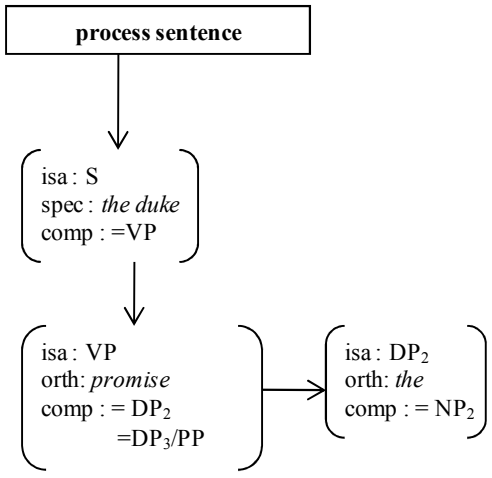


retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP$_1$ with =NP of DP$_1$
  pop DP-chunk
unify DP$_1$ with =DP of S
  retrieve VP-chunk

The VP-*promise*-chunk has two open 'comp' values. One selects for a DP. The other selects for either a DP or PP chunk. Rather than having two independent VP-*promise*-chunks (one that

selects a DP and PP for the prepositional dative form and one that selects two DPs for the double object form), I use the denotation =DP/PP above to indicate optionality. Pragmatic and semantic information that might bias toward one completion or the other is not included in the current example. Here, I focus strictly on syntactic processing though leave open the possibility for separate levels of pragmatic or semantic processing. The processor keeps track of which rules are retrieved and fired and whether the pattern of firings led to the creation of two DPs (the double object form of the dative alternation) or a DP and a PP (the prepositional forms of the dative alternation).

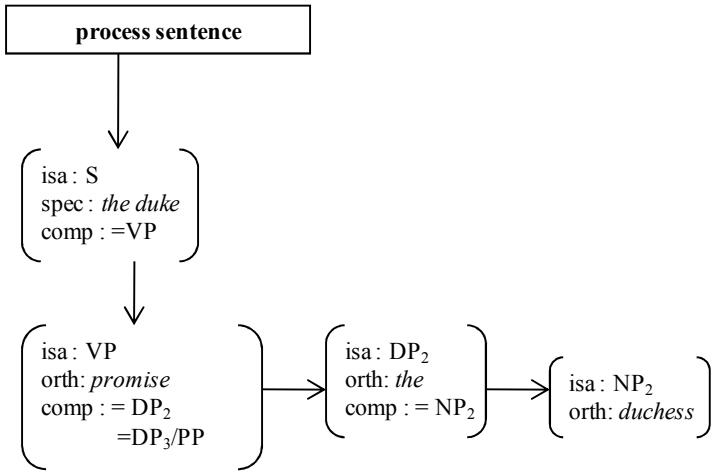
Returning to the processing at hand, the processor notes the open values in the VP, so it selects a 'push VP' rule to put it in the problem state buffer. The processor again checks the buffers, notes the open values in the VP-chunk currently in the problem state buffer and the empty retrieval buffer and selects a 'retrieve DP' rule.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk

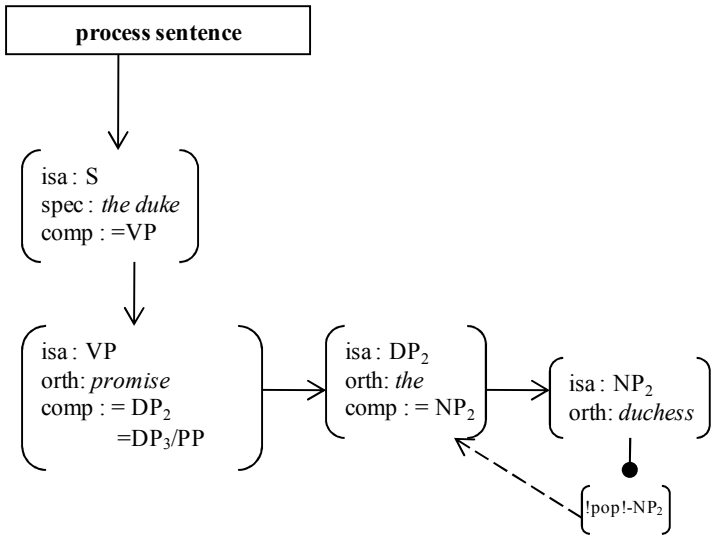
Next, the processor notes the open value in the DP-chunk, so it selects a 'push DP' rule to move

it into the problem state buffer. The processor then selects a 'retrieve NP' rule in an attempt to help resolve the open values in the VP-chunk.



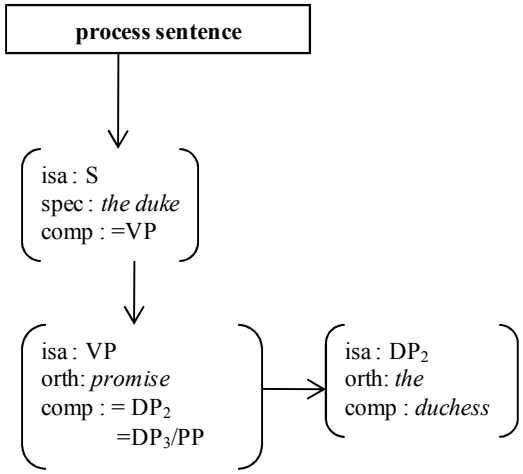
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP$_1$ with =NP of DP$_1$
  pop DP-chunk
unify DP$_1$ with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk

There are no open values in the NP-*duchess*-chunk, so the processor fires a 'pop NP' rule, making the NP available for unification. The values of this chunk match the open values of the DP-chunk associated with the problem state buffer. The popped NP unifies with the open =NP value of the DP-chunk.



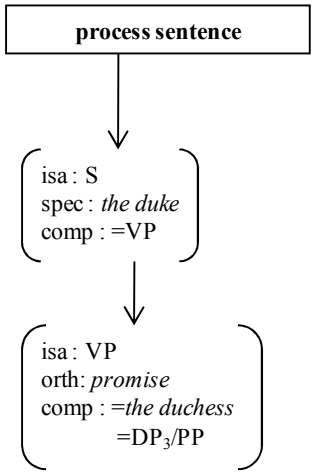
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP$_1$ with =NP of DP$_1$
  pop DP-chunk
unify DP$_1$ with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP$_2$ with =NP$_2$ of DP$_2$

The open value in the DP is now resolved, so the DP-chunk is popped from the buffer and becomes available for unification. Its values match the open =DP value in the VP-chunk, so they unify.



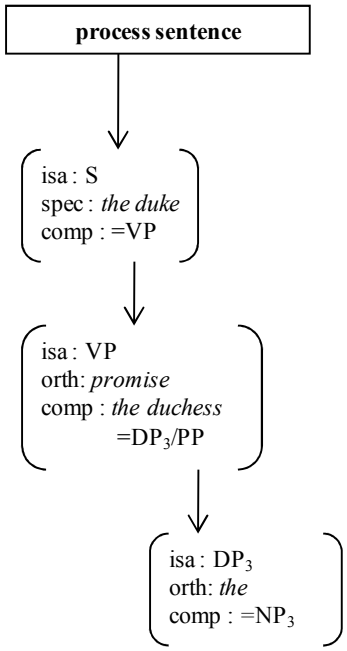
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP

The first argument of the VP-*promise*-chunk is now filled. However, because it still has an open value, the VP chunk remains in the problem state buffer.



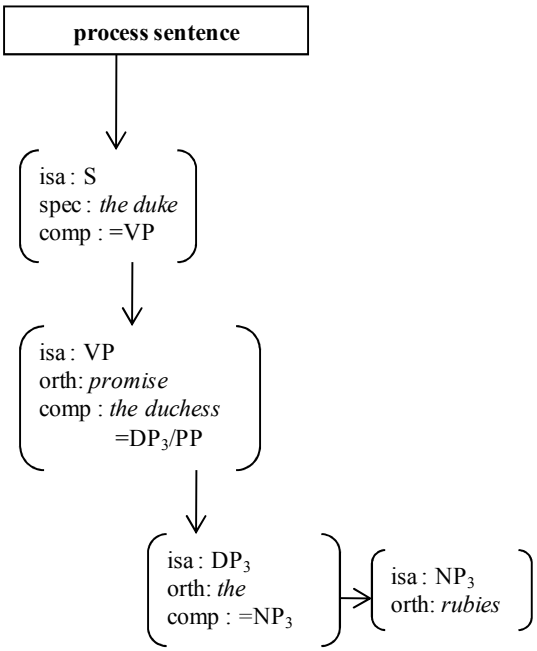
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP

The processor notes the open value in the VP-chunk (i.e. the =DP/PP) and the empty retrieval buffer. It selects among the production rules and fires a 'retrieve DP' chunk, which returns the 'DP-*the*-chunk' and places it in the retrieval buffer.



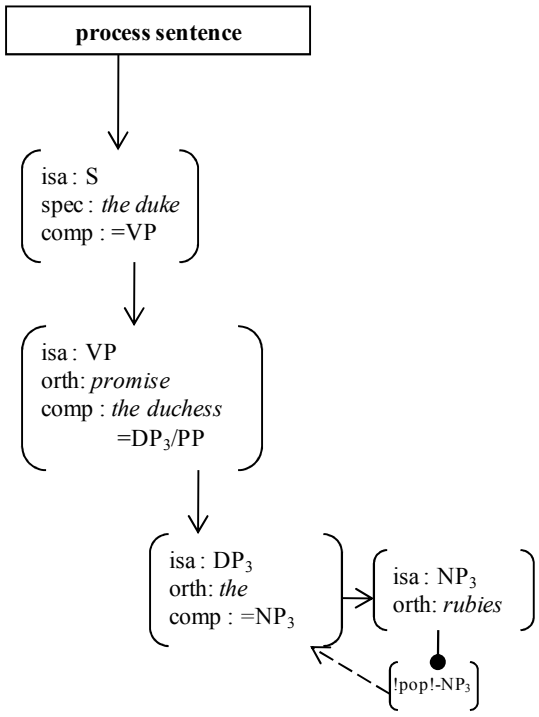
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk

The DP-chunk also has an open value, so the processor chooses a 'push DP' rule. Then it begins work on processing an NP to resolve this open value, firing a 'retrieve NP' rule, which returns the NP-*rubies*-chunk.



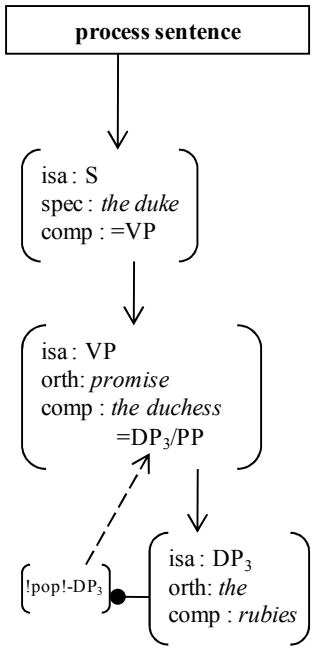
retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk

The NP-chunk has no open values, so the processor selects a 'pop NP' rule, making the NP available for unification. Because its value (NP) matches the open value in the DP-chunk, the popped NP unifies with the open =NP value of the DP-chunk.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_3 with =NP_3 of DP_3

The DP has no open values, so the processor selects a 'pop DP' rule. The DP unifies with the open =DP value in the VP-chunk.

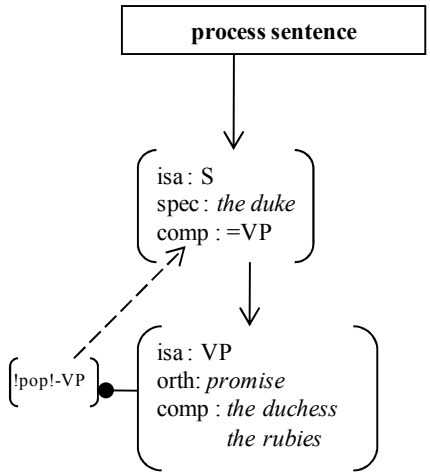


retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_3 with =NP_3 of DP_3
    pop DP-chunk
unify DP_3 with =DP_3 of VP

All of the rules used to process the open values of the VP-*promise*-chunk ultimately worked to resolve the subgoals of that chunk. Because they all fired to resolve the VP-*promise*-chunk's subgoals, the rules are associated with the same unification chain. All the rules used to process the first DP_2 and the second DP_3 are part of the same subgoal structure (i.e. the structure necessary to satisfy the 'process VP' subgoal). Each of the unifications that followed one of the 'pop' rules resolved an open value of its larger subgoal, making them part of the same unification chain and, hence, part of the same memory trace.

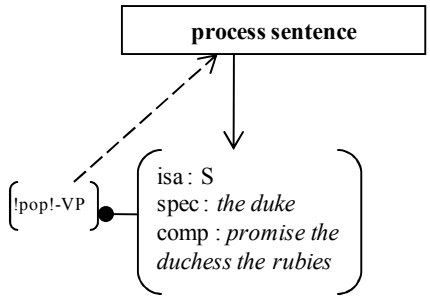
Now that the VP's open values are resolved, the processor selects a rule to pop it from the buffer system. It is available for unification. The final chunk associated with the problem state (i.e. the S-chunk) has an open =VP value. The popped VP-chunk and the open =VP value of the

S-chunk unify.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_3 with =NP_3 of DP_3
    pop DP-chunk
unify DP_3 with =DP_3 of VP
  pop VP-chunk
unify VP with =VP of S

The S-chunk has no open values, so the processor pops it from the buffer system, and it unifies with the main goal ‘process sentence’ in the control state.



retrieve S-chunk
push S-chunk
  retrieve DP-chunk
  push DP-chunk
    retrieve NP-chunk
    pop NP-chunk
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP-chunk
  push VP-chunk
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_2 with =NP_2 of DP_2
    pop DP-chunk
unify DP_2 with =DP_2 of VP
    retrieve DP-chunk
    push DP-chunk
      retrieve NP-chunk
      pop NP-chunk
unify NP_3 with =NP_3 of DP_3
    pop DP-chunk
unify DP_3 with =DP_3 of VP
  pop VP-chunk
unify VP with =VP of S
pop S-chunk
unify S with control state

As suggested by the right-hand column, each rule fired in pursuit of the same goal (i.e. the processing of the sentence "The duke promised the duchess the rubies"), and as such are associated with the same unification chain. All the rules that were involved in the processing of the subject DP and the predicate VP satisfied the subgoals of the same S-chunk, so they are linked (see Chapter 2, section 3.4.1 for discussion). Because they are linked, they are represented together in LTM. When the processor retrieves the sentence, it retrieves this series of firings.

In this example, the prime (i.e. the DP form of the dative alternation) occurred in the matrix clause. The unification chain in which it occurred was associated with 21 rule

applications, as shown in Table 4.15:

Table 4.15: List of rules for the processing of the matrix DO alternation

retrieve S-chunk
push S-chunk
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_1 with =NP of DP_1
pop DP-chunk
unify DP_1 with =DP of S
retrieve VP-chunk
push VP-chunk
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_2 with =NP_2 of DP_2
pop DP-chunk
unify DP_2 with =DP_2 of VP
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_3 with =NP_3 of DP_3
pop DP-chunk
unify DP_3 with =DP_3 of VP
pop VP-chunk
unify VP with =VP of S
pop S-chunk
unify S with control state

In what follows, I show how the other sentence types examined in Experiments 1 and 2 are processed by the model presented in Chapter 2. Whenever possible, I refer the example just discussed rather than repeat the process in full. Furthermore, rather than demonstrating the retrieval of each chunk, its popping, and its unification as in the previous example, I show only the rules used for the processing. Appendix 4E provides further detail. PRICE contends that different patterns of priming arise due to the way memory represents the processing of sentences, and this representation is described as the unification chain, or chains, generated during the

processing event. I begin by comparing the chain generated during the processing of the sentences used in the Verb Complement (VC) condition. Recall that these sentence type mirror Branigan *et al.*'s (2006) sentence types and included primes in matrix sentences preceded by adverbial clauses and primes embedded in verb complement clauses. Examples are given below with the alternation underlined and the relevant context in brackets.

(30) Prime in matrix clause with introductory adverbial clause
As the report declared, [the duke promised the duchess the rubies].

(31) Prime in verb complement clause
The report declared [that the duke promised the duchess the rubies].

I begin with the processing of sentence (30), "As the reported declared, the duke promised the duchess the rubies." As in the matrix example above, the processor has the main goal 'process sentence' in the control buffer. This goal initiates the firing of a 'retrieve S-chunk' rule. The S-chunk is placed in the retrieval buffer. However, because it has two open values (=DP and =VP),[1] it cannot be popped and is, instead, pushed into the problem state via a 'push S' rule:

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk	The processor retrieves an S-chunk and pushes it into the
push S-chunk	problem state because of its open values.

At this point, the processor retrieves an AdvC , which also contains open values (i.e. 'spec : =Adv', 'comp : =S', and 'mod : =S'). This retrieval does not follow directly from the needs of the chunk in the problem state buffer. That is, the selection an AdvC does not necessarily assist in resolving the open values in the S-chunk. I denote this by separating the AdvC's retrieval from the S-chunk. This separation becomes more relevant as the processing continues and different

[1] See Appendix 2A for a list of the chunks including their open values.

unification chains are generated. Because the AdvC has open values, the processor selects a 'push AdvC' rule, moving it into the problem state buffer.

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk push S-chunk	The processor retrieves an S-chunk and pushes it into the problem state because of its open values.
retrieve AdvC push AdvC	The processor retrieves a AdvC-chunk and places it into the retrieval buffer and then pushes it into the problem state because of its open values ('spec :=Adv', 'comp : =S', and 'mod : =S'). [1]

The processor evaluates the problem state buffer and the retrieval buffer and selects a rule to retrieve an adverb. It selects an Adv-*as*-chunk and places it into the retrieval buffer. Because this chunk has no open values it is popped.

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk push S-chunk	The processor retrieves an S-chunk and pushes it into the problem state because of its open values
retrieve AdvC push AdvC	The processor retrieves a AdvC-chunk and places it into the retrieval buffer and then pushes it into the problem state because of its open values ('spec :=Adv', 'comp : =S', and 'mod : =S').
retrieve Adv pop Adv ***unify Adv with AdvC***	The processor retrieves an adverb and places it into the retrieval buffer. It has no open values, so it is popped and unified with the AdvC.

The Adv-*as*-chunk's values unify with the =Adv value in the AdvC-chunk. However, the processor is not finished with the AdvC chunk. The chunk still has two open values, i.e. 'comp : =S' and 'mod : =S.' The processor fires an S-chunk retrieval rule and places the S-chunk in the retrieval buffer. Because the S-chunk has open values, it too is pushed into the problem state buffer.

[1] The 'mod' value denotes the type of XP that the AdvC needs to modify. See Chapter 2, section 3.3.1 for further discussion.

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk push S-chunk	The processor retrieves an S-chunk and pushes it into the problem state because of its open values
retrieve AdvC push AdvC	The processor retrieves a AdvC-chunk and places it into the retrieval buffer and then pushes it into the problem state because of its open values ('spec :=Adv', 'comp : =S', and 'mod : =S').
retrieve Adv pop Adv ***unify Adv with =Adv in AdvC*** retrieve S_2-chunk push S_2-chunk	The processor retrieves an adverb and places it into the retrieval buffer. It has no open values, so it is popped and unified with the AdvC. The processor retrieves an S-chunk and pushes it into the problem state buffer.

The S-chunk has two open values: 'spec : =DP' and 'comp : =VP-gap.' The open value in the spec feature is the same as in the previous S-chunks. However, the VP-gap value is new. This VP-gap allows the processor to note that it is processing a structure that contains a gap (see Lewis & Vashisth 2005 for discussion). I return to the processing of the VP-gap-chunk below. The processor begins by firing the rules necessary to retrieve, push, and pop the chunks associated with the processing of the "the report" DP. The production rules are shown below.

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk push S-chunk	The processor retrieves an S-chunk and pushes it into the problem state because of its open values
retrieve AdvC push AdvC	The processor retrieves a AdvC-chunk and places it into the retrieval buffer and then pushes it into the problem state because of its open values (spec : =Adv, comp : =S, mod =S).
retrieve Adv pop Adv ***unify Adv with =Adv in AdvC***	The processor retrieves an adverb and places it into the retrieval buffer. It has no open values, so it is popped and unified with the AdvC.
retrieve S_2-chunk push S_2-chunk	The processor retrieves an S-chunk and pushes it into the problem state buffer.
retrieve DP_1-chunk push DP_1-chunk	The processor retrieves the DP-*the*-chunk, places it into the retrieval buffer.
retrieve NP_1-chunk pop NP_1 ***unify NP with =NP in DP-chunk***	It then retrieves the NP-*report*-chunk and pops it. It unifies with the open =NP value in the DP-chunk.
pop DP_1 ***unify DP with =DP in S-chunk***	The processor pops the DP, which then unifies with the open =DP value in the S-chunk.

At this point, the S-chunk in the problem state has one open value =VP-gap. The processor notes this and the empty retrieval buffer. It fires a 'retrieve VP-gap' rule and returns the VP-gap-*declare*-chunk. This chunk, as shown below,[1] has an empty '___' associated with its 'comp' feature and a 'gap' feature that needs a filler of the type S. These lists are ultimately saturated by the values of the open =S value in the AdvC-chunk in a manner similar to the way the filler-gap dependency is satisfied in relative clause constructions (see Chapter 3, section 5.5.2 for discussion).

VP-gap-*declare*-chunk

isa : VP-gap
orth: *declare*
comp : __
gap : =S

[1] See Appendix 2A for a complete list of chunks.

The VP-gap chunk is unified with the open =VP-gap-chunk value of the S-chunk, passing the gap up. The S-chunk's values are now all resolved because its VP-gap value is filled, so the S-chunk is popped and unified with the open =S value in the AdvC-chunk. The AdvC-chunk has no open values, so it is popped from the buffer systems. It is available for unification, but the chunk in the problem state (i.e. the S-chunk) does not have an open =AdvC. The popped-AdvC cannot, therefore, unify with anything in the current problem state, so its syntactic representation proceeds to long-term memory (LTM).

'process sentence'	**The state of the control buffer is to process a sentence**
retrieve S-chunk push S-chunk	The processor retrieves an S-chunk and pushes it into the problem state because of its open values.
retrieve AdvC push AdvC	The processor retrieves a AdvC-chunk and places it into the retrieval buffer and then pushes it into the problem state because of its open values (spec : =Adv, comp : =S, mod =S).
retrieve Adv pop Adv ***unify Adv with =Adv in AdvC***	The processor retrieves an adverb and places it into the retrieval buffer. It has no open values, so it is popped and unified with the AdvC.
retrieve S_2-chunk push S_2-chunk	The processor retrieves an S-chunk and pushes it into the problem state buffer.
retrieve DP_1-chunk push DP_1-chunk	The processor retrieves the DP-*the*-chunk, places it into the retrieval buffer.
retrieve NP_1-chunk pop NP_1 ***unify NP with =NP in DP-chunk***	It then retrieves the NP-*report*-chunk and pops it. It unifies with the open =NP value in the DP-chunk.
pop DP_1 ***unify DP with =DP in S-chunk***	The processor pops the DP, which then unifies with the open =DP value in the S-chunk.
retrieve VP-gap-chunk pop VP ***unify VP with =VP-gap in S-chunk***	The processor retrieves the VP-*declare*-gap-chunk, pops it from the retrieval buffer and unifies it with the open =VP value in the S-chunk.
pop S ***unify S with AdvC-chunk***	The processor pops the S-chunk and unifies it with the AdvC-chunk.
pop AdvC ***→send to LTM***	The processor pops the AdvC. It has nothing to unify with, so it is sent to long-term memory.

The processor checks the buffer states, notes that the retrieval buffer is empty and that the problem state still has the S-chunk with its open =DP and =VP, so it returns to the subgoals associated with these open values. The processing of the =DP subject and =VP predicate of the matrix clause proceeds just as it did in the matrix example above. Rather than repeat the processing steps here, I simply add them to the list of rules used thus far.

Combining the process of this matrix clause with the processing of the AdvC, we end up the list of production rule below:

'process sentence'
retrieve S-chunk
push S-chunk
retrieve AdvC
push AdvC
retrieve Adv
pop Adv
unify Adv with =Adv in AdvC
retrieve S_2-chunk
push S_2-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in DP-chunk
pop DP_1
unify DP with =DP in S-chunk
retrieve VP-gap-chunk
pop VP
unify VP with =VP-gap in S-chunk
pop S
unify S with AdvC-chunk
pop AdvC
→send to LTM
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_1 with =NP of DP_1
pop DP-chunk
unify DP_1 with =DP of S
retrieve VP-chunk
push VP-chunk
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_2 with =NP_2 of DP_2
pop DP-chunk
unify DP_2 with =DP_2 of VP
retrieve DP-chunk
push DP-chunk
retrieve NP-chunk
pop NP-chunk
unify NP_3 with =NP_3 of DP_3
pop DP-chunk
unify DP_3 with =DP_3 of VP
pop VP-chunk
unify VP with =VP of S
pop S-chunk
unify S with control state

In the above depiction, the AdvC production rules are separated (as indicated by the horizontal

lines) from the rules used to process the matrix clause.

During subsequent retrieval of the sentence "As the report declared, the duke promised the duchess the rubies," the processor retrieves the unification chain associated with the processing of the adverbial clause and the chain associated with the processing of the matrix clause, as shown in Table 4.16 below.

Table 4.16: Unification chains for matrix prime with adverbial clause

"as the report declared"	**"the duke promised the duchess the rubies"**
retrieve AdvC	retrieve S-chunk
push AdvC	push S-chunk
retrieve Adv	retrieve DP-chunk
pop Adv	push DP-chunk
unify Adv with =Adv in AdvC	retrieve NP-chunk
retrieve S_2-chunk	pop NP-chunk
push S_2-chunk	***unify NP_1 with =NP of DP_1***
retrieve DP_1-chunk	pop DP-chunk
push DP_1-chunk	***unify DP_1 with =DP of S***
retrieve NP_1-chunk	retrieve VP-chunk
pop NP_1	push VP-chunk
unify NP with =NP in DP-chunk	retrieve DP-chunk
pop DP_1	push DP-chunk
unify DP with =DP in S-chunk	retrieve NP-chunk
retrieve VP-gap-chunk	pop NP-chunk
pop VP	***unify NP_2 with =NP_2 of DP_2***
unify VP with =VP-gap in S-chunk	pop DP-chunk
pop S	***unify DP_2 with =DP_2 of VP***
unify S with AdvC-chunk	retrieve DP-chunk
pop AdvC	push DP-chunk
	retrieve NP-chunk
	pop NP-chunk
	unify NP_3 with =NP_3 of DP_3
	pop DP-chunk
	unify DP_3 with =DP_3 of VP
	pop VP-chunk
	unify VP with =VP of S
	pop S-chunk
	unify S with control state

When the processor determines the utility of a rule, it evaluates each rule according to the number of other rules that are associated with the rule's unification chain. The more rules associated with chain, the lower the utility of each rule.

In the current example, the prime is associated with the matrix chain (right-hand column). When the processor needs to evaluate the utility of using the primed form (DO) or the alternate

(PD), it estimates the utility partly on the basis of how costly the rule was when applied in the processing of the prime sentence. That is, it estimates the utility of the prime form based, in part, on how many other rules need to fire in order for the main goal to be successfully resolved.

To illustrate, consider the difference between processing the adverbial clause sentence above and the processing of the verb complement clause sentence repeated below.

(31) Prime in verb complement clause
The report declared [that the duke promised the duchess the rubies].

The processing proceeds in a manner similar to that as in the previous example up to the retrieval of the first verb.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in DP-chunk
pop DP_1
unify DP with =DP in S-chunk
retrieve VP_1-chunk
push VP_1-chunk

The retrieved verb (i.e. VP-*declare*-chunk) contains an open value for either a DP or CP argument. The processor pushes the VP chunk into the problem state buffer. The processor then chooses to build a CP structure to due to pressures coming from, for example, the demands of the intended message (e.g. an act of declaring that an act of promising between a duke and duchess involving rubies occurred). Because this choice of a CP can lead to the resolution of a subgoal currently in the problem state, the CP is represented as occurring in the same processing line as

the VP (i.e. there is no line separating them). The table below contains all the rules up to the retrieval of the CP and the rules that fire to satisfy the CP-chunk's open =Comp value.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in DP-chunk
pop DP_1
unify DP with =DP in S-chunk
retrieve VP_1-chunk
push VP_1-chunk
retrieve CP-chunk
push CP-chunk
retrieve Comp-chunk
pop Comp-chunk
unify Comp with CP

At this point, the processor follows the same pattern of rule firings as shown in the two matrix examples shown above, leading to the processing of a DO form.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in DP-chunk
pop DP_1
unify DP with =DP in S-chunk
retrieve VP_1-chunk
push VP_1-chunk
retrieve CP-chunk
push CP-chunk
retrieve Comp-chunk
pop Comp-chunk
unify Comp with CP
retrieve S_2-chunk
push S_2-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_2
unify NP with =NP in DP-chunk
pop DP_2
unify DP with =DP in S-chunk
retrieve VP-chunk
push VP-chunk
retrieve DP_3-chunk
push DP_3-chunk
retrieve NP_3-chunk
pop NP_3
unify NP with =NP in DP-chunk
pop DP_3
unify DP with =DP in VP-chunk
retrieve DP_4-chunk
push DP_4-chunk
retrieve NP_4-chunk
pop NP_4
unify NP with =NP in DP-chunk
pop DP_4
unify DP with =DP in VP-chunk
pop VP
unify VP with =VP in S-chunk
pop S

Now, the S-chunk unifies with the open =S value in the CP-chunk. The CP-chunk is popped and

unified with the open =CP value in the VP-chunk. The VP chunk pops and unifies with the open =VP value in the S chunk, thereby resolving the last subgoal in the problem state.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in DP-chunk
pop DP_1
unify DP with =DP in S-chunk
retrieve VP_1-chunk
push VP_1-chunk
retrieve CP-chunk
push CP-chunk
retrieve Comp-chunk
pop Comp-chunk
unify Comp with CP
retrieve S_2-chunk
push S_2-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_2
unify NP with =NP in DP-chunk
pop DP_2
unify DP with =DP in S-chunk
retrieve VP-chunk
push VP-chunk
retrieve DP_3-chunk
push DP_3-chunk
retrieve NP_3-chunk
pop NP_3
unify NP with =NP in DP-chunk
pop DP_3
unify DP with =DP in VP-chunk
retrieve DP_4-chunk
push DP_4-chunk
retrieve NP_4-chunk
pop NP_4
unify NP with =NP in DP-chunk
pop DP_4
unify DP with =DP in VP-chunk
pop VP
unify VP with =VP in S-chunk
pop S
unify S with CP-chunk
pop CP
unify CP with VP_1-chunk
pop VP_1
unify VP with S_1-chunk
pop S

The popped S unifies with the main goal in the control state, and the processing of the sentence is complete. Unlike the previous example for sentence (30) (i.e. the matrix prime with an introductory adverbial clause), all of the rules fired in this example are associated with the same unification chain. Each retrieval, pushing, and popping worked to satisfy the same subgoal structure. Thus, when the processor retrieves the memory trace for the sentence "The report declared that the duke promised the duchess the rubies," it retrieves the entire sequence of rules that are associated with the unification chain that was built during the processing of the sentence. This chain, in which the prime occurs, is noticeably longer than the matrix chain, in which the previous example's prime occurred. Compare the number of rules in Table 4.17 below.

Table 4.17: Comparison of unification chains for matrix and verb complement clause

Matrix clause	Verb complement clause
retrieve S-chunk	retrieve S-chunk
push S-chunk	push S-chunk
retrieve DP-chunk	retrieve DP_1-chunk
push DP-chunk	push DP_1-chunk
retrieve NP-chunk	retrieve NP_1-chunk
pop NP-chunk	pop NP_1
unify NP_1 with =NP of DP_1	***unify NP with =NP in DP-chunk***
pop DP-chunk	pop DP_1
unify DP_1 with =DP of S	***unify DP with =DP in S-chunk***
retrieve VP-chunk	retrieve VP_1-chunk
push VP-chunk	push VP_1-chunk
retrieve DP-chunk	retrieve CP-chunk
push DP-chunk	push CP-chunk
retrieve NP-chunk	retrieve Comp-chunk
pop NP-chunk	pop Comp-chunk
unify NP_2 with =NP_2 of DP_2	***unify Comp with CP***
pop DP-chunk	retrieve S_2-chunk
unify DP_2 with =DP_2 of VP	push S_2-chunk
retrieve DP-chunk	retrieve DP_1-chunk
push DP-chunk	push DP_1-chunk
retrieve NP-chunk	retrieve NP_1-chunk
pop NP-chunk	pop NP_2
unify NP_3 with =NP_3 of DP_3	***unify NP with =NP in DP-chunk***
pop DP-chunk	pop DP_2
unify DP_3 with =DP_3 of VP	***unify DP with =DP in S-chunk***
pop VP-chunk	retrieve VP-chunk
unify VP with =VP of S	push VP-chunk
pop S-chunk	retrieve DP_3-chunk
unify S with control state	push DP_3-chunk
	retrieve NP_3-chunk
	pop NP_3
	unify NP with =NP in DP-chunk
	pop DP_3
	unify DP with =DP in VP-chunk
	retrieve DP_4-chunk
	push DP_4-chunk
	retrieve NP_4-chunk
	pop NP_4
	unify NP with =NP in DP-chunk
	pop DP_4
	unify DP with =DP in VP-chunk
	pop VP
	unify VP with =VP in S-chunk
	pop S
	unify S with CP-chunk
	pop CP
	unify CP with VP_1-chunk
	pop VP_1
	unify VP with S_1-chunk
	pop S
21 rules	40 rules

Here we see that the prime occurs in a much longer chain of rules. For the verb complement

clause prime sentence, 40 rule firings occurred during the processing of the sentence in which the prime occurred. For the matrix clause prime in the sentence with an introductory adverbial clause, only 21 rule firings are associated with the prime's unification chain. According to PRICE, the higher number of rules decreases the utility of the rules. This decreased utility should lead to weaker priming effects.

At the short lags (i.e. Experiment 1), there was no difference between primes in these two sentence types. However, at long lags, significant differences did arise. Specifically, priming from verb complement clauses disappeared. This behavior is in keeping with PRICE. Recall that there are two main factors affecting the likelihood of a rule's (or rule pattern's) use: strength and utility. Utility is determined by the number of rules necessary for processing and the likelihood of success minus the cost of using the rules. Thus, utility is affected by structural context. The larger the chain, the weaker the utility. However, this is not the only factor the processor considers. It also considers the strength of the rules. This factor is affected by the recency of a rule's use and may be sufficient to make its retrieval more likely. PRICE contends that as the strength wanes and returns to the baseline, utility becomes more relevant. Due to the low utility score of the prime in verb complement clauses, the prime is less likely to demonstrate priming than primes with higher utility scores (e.g. those occurring in matrix position) when its strength drops below some threshold.

Another of PRICE's predictions was that primes occurring in argument clauses differ from primes occurring in adjunct clauses. In the current experiments, this would mean that priming from verb complement clauses and relative clauses should differ. Before testing this

prediction, let us consider the processing of the two prime sentence types in the Relative Clause condition. (32) and (33) contain examples in which the prime alternate is underlined and the relative clause is in brackets.

(32) Prime in matrix position of sentence with subject-modifying relative clause
The duke [who loved the king] promised the duchess the rubies.

(33) Prime embedded in object-modifying relative clause
The king loved the duke [who promised the duchess the rubies].

We begin with the relative clause sentence in which the dative alternation occurs in matrix position (i.e. (32)). In this sentence type, the processor begins as it normally would, by retrieving an S-chunk, placing it into the problem state buffer, and initiating the series of retrievals, pushings, and poppings necessary to build the subject DP. This process is shown up to the popping of the NP.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1

At this point, rather than unifying the popped NP with the open =NP value in the DP-chunk associated with the next subgoal, the processor retrieves a RelC from long-term memory and places it in the retrieval buffer.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC

The retrieval of the RelC is a departure from the subgoal stack currently in the problem state. Unlike the retrieval of an NP-chunk, which can ultimately lead to the resolution of a current subgoal in the stack, the retrieval of a RelC-chunk initiates a new subgoal chain, as denoted by the horizontal line between the 'pop NP' and the 'retrieve RelC' rules. Although rules that specifically address the needs of the problem state may be more likely (e.g. because their utility for resolving the subgoal may be high), the processor may select other rules in order to satisfy other pressures, such as pragmatic or semantic concerns.

Now that the RelC-chunk is in the retrieval buffer, the processor begins work on the 'process RelC' subgoal. The RelC-chunk has open values (i.e. 'spec : =RelP,' 'comp : =S-gap,' and 'mod : =NP'), so the processor pushes the chunk into the problem state and begins work on the 'process RelP' subgoal. This subgoal leads to the retrieval of the RelP-*who*-chunk. This chunk has no open values, so it is popped and unifies with the =RelP value of the RelC-chunk. Below are the rules associated with the processing up to this point.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC
retrieve RelP
pop RelP
unify RelP with =RelP in RelC

The processor now moves on to satisfy the next subgoal (i.e. 'comp : =S-gap'). It fires a 'retrieve

S-gap' rule and places the S-gap-chunk in the retrieval buffer. A rule then pushes it into the problem state buffer.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC
retrieve RelP
pop RelP
unify RelP with =RelP in RelC
retrieve S-gap-chunk
push S-gap-chunk

The S-gap-chunk contains an open gap feature, as denoted by the gap : ___ (see Chapter 2, Appendix 2A for a complete listing of the chunks and Chapter 3, section 5 for a discussion of how chunks with gap features unify). It also contains a 'gap' feature that requires a filler of the type NP, as denoted by the =NP. These two features and their values indicate that the subject of the relative clause is 'extracted' and that information about this extracted element is ultimately provided by the filler (i.e. the values of the NP-chunk that unify with the open =NP in the 'mod' of the RelC-chunk) via a subject extraction lexical rule (Sag, Wasow, & Bender 2003).

Now, we move onto the processing of the S-gap-chunk's 'process VP' subgoal. This subgoal leads to the retrieval of the VP-*love*-chunk, which is then pushed into the problem state buffer so that its open =DP value can be resolved. Below are the rules and unifications necessary for the processing of the rest of the RelC up to the point where the S-gap-chunk unifies with the open =S-gap value in the RelC-chunk.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC
retrieve RelP
pop RelP
unify RelP with =RelP in RelC
retrieve S_2-chunk
push S_2-chunk
retrieve VP_2-chunk
push VP_2-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2
unify NP with =NP in DP-chunk
pop DP_2
unify DP with =DP in VP-chunk
pop VP
unify VP with =VP in S-chunk
pop S
unify S with =S in RelC-chunk

Now all but one of the RelC-chunk's open values are resolved. The only remaining open value is the =NP value in the 'mod' feature. Recall that the 'mod' feature denotes the type of XP the chunk must modify (Chapter 2, section 3.3.1), in this case an NP. The processor fires a 'retrieve NP' rule. Due to its recent processing, the processor retrieves the NP-*duke*-chunk and places it in the retrieval buffer. Because this NP-chunk is the same chunk that the processor previously retrieved, I use the same numbering for it (i.e. NP_1). The processor then pops the NP-chunk. It unifies with the open =NP value in the RelC-chunk, thereby resolving all of the RelC's open values.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC
retrieve RelP
pop RelP
unify RelP with =RelP in RelC
retrieve S-gap-chunk
push S-gap-chunk
retrieve VP_2-chunk
push VP_2-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2
unify NP with =NP in DP-chunk
pop DP_2
unify DP with =DP in VP-chunk
pop VP
unify VP with =VP in S-chunk
pop S
unify S with =S in RelC-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in RelC-chunk
pop RelC

The processor pops the RelC, and it becomes available for unification. However, the subgoal currently in the problem state buffer (i.e. the 'process NP' associated with the open =NP value in the DP-*the*-chunk in the problem stated) does not have an open =RelC value. Because the RelC-chunk cannot unify with an open value in the next subgoal, it proceeds to LTM, and the processor begins work on the most active subgoal: 'process NP.'

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
retrieve RelC
retrieve RelP
pop RelP
unify RelP with =RelP in RelC
retrieve S-gap-chunk
push S-gap-chunk
retrieve VP_2-chunk
push VP_2-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2
unify NP with =NP in DP-chunk
pop DP_2
unify DP with =DP in VP-chunk
pop VP
unify VP with =VP in S-chunk
pop S
unify S with =S in RelC-chunk
retrieve NP_1-chunk
pop NP_1
unify NP with =NP in RelC-chunk
pop RelC
➔ ***send to LTM***

The subgoal 'process NP' must be resolved to satisfy the DP-chunk's open =NP value. The processor must again fire a 'retrieve NP' rule that, again, returns the NP-*duke*-chunk. The NP-chunk is placed into and then popped from the retrieval buffer. It unifies with the open =NP value in the DP-chunk.

'process sentence'	
retrieve S-chunk	
push S-chunk	
retrieve DP_1-chunk	
push DP_1-chunk	
retrieve NP_1-chunk	
pop NP_1	
retrieve RelC	
retrieve RelP	
pop RelP	
unify RelP with =RelP in RelC	
retrieve S-gap-chunk	
push S-gap-chunk	
retrieve VP_2-chunk	
push VP_2-chunk	
retrieve DP_2-chunk	
push DP_2-chunk	
retrieve NP_2-chunk	
pop NP_2	
unify NP with =NP in DP-chunk	
pop DP_2	
unify DP with =DP in VP-chunk	
pop VP	
unify VP with =VP in S-chunk	
pop S	
unify S with =S in RelC-chunk	
retrieve NP_1-chunk	
pop NP_1	
unify NP with =NP in RelC-chunk	
pop RelC	→ ***send to LTM***
retrieve NP_1-chunk	
pop NP_1	
unify NP_1 with =NP of DP_1	
pop DP-chunk	
unify DP_1 with =DP of S	
retrieve VP-chunk	
push VP-chunk	
retrieve DP-chunk	
push DP-chunk	
retrieve NP-chunk	
pop NP-chunk	
unify NP_2 with =NP_2 of DP_2	
pop DP-chunk	
unify DP_2 with =DP_2 of VP	
retrieve DP-chunk	
push DP-chunk	
retrieve NP-chunk	
pop NP-chunk	
unify NP_3 with =NP_3 of DP_3	
pop DP-chunk	
unify DP_3 with =DP_3 of VP	
pop VP-chunk	
unify VP with =VP of S	
pop S-chunk	
unify S with control state	

All of the rules that fired during the processing of the matrix clause, i.e. the clause with the dative verb, are associated with the same unification chain. The rules associated with processing the matrix clause are associated with a single unification chain, whereas the rules associated with the relative clause are associated with a separate chain. The processor considers only the rules associated with the unification chain built during the processing of the matrix clause in the left-hand column below.

Table 4.18: Unification chains for matrix prime with relative clause

"the duke promised the duchess the rubies"	**"who loved the king"**
retrieve S-chunk	retrieve RelC
push S-chunk	retrieve RelP
retrieve DP_1-chunk	pop RelP
push DP_1-chunk	***unify RelP with =RelP in RelC***
retrieve NP_1-chunk	retrieve S-gap-chunk
pop NP_1	push S-gap-chunk
retrieve NP_1-chunk	retrieve VP_2-chunk
pop NP_1	push VP_2-chunk
unify NP_1 with =NP of DP_1	retrieve DP_2-chunk
pop DP-chunk	push DP_2-chunk
unify DP_1 with =DP of S	retrieve NP_2-chunk
retrieve VP-chunk	pop NP_2
push VP-chunk	***unify NP with =NP in DP-chunk***
retrieve DP-chunk	pop DP_2
push DP-chunk	***unify DP with =DP in VP-chunk***
retrieve NP-chunk	pop VP
pop NP-chunk	***unify VP with =VP in S-chunk***
unify NP_2 with =NP_2 of DP_2	pop S
pop DP-chunk	***unify S with =S in RelC-chunk***
unify DP_2 with =DP_2 of VP	retrieve NP_1-chunk
retrieve DP-chunk	pop NP_1
push DP-chunk	***unify NP with =NP in RelC-chunk***
retrieve NP-chunk	pop RelC
pop NP-chunk	➔ ***send to LTM***
unify NP_3 with =NP_3 of DP_3	
pop DP-chunk	
unify DP_3 with =DP_3 of VP	
pop VP-chunk	
unify VP with =VP of S	
pop S-chunk	
unify S with control state	

Compare the processing of this prime with the processing of a prime embedded within a

relative clause, e.g. (33) repeated below:

(33) Prime embedded in object-modifying relative clause
The king loved the duke [who promised the duchess the rubies].

The processor begins by retrieving an S-chunk and following the pattern of rules necessary for building a subject DP and a predicate VP.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP_1 with =NP of DP_1
pop DP-chunk
unify DP_1 with =DP of S
retrieve VP_1-chunk
push VP_1-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2-chunk

Here we see the pattern of retrievals, pushings, and poppings that takes us up to the point where the NP is popped and the relative clause begins. Just as in the previous relative clause example, the processor selects a 'retrieve RelC' rule, which generates a new subgoal structure ('process RelC').

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP_1 with =NP of DP_1
pop DP-chunk
unify DP_1 with =DP of S
retrieve VP_1-chunk
push VP_1-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2-chunk
retrieve RelC
push RelC
retrieve RelP
pop RelP
unify RelP with =Rel P of RelC
retrieve S_2-chunk
push S_2-chunk
retrieve VP_2-chunk
push VP_2-chunk
retrieve DP_3-chunk
push DP_3-chunk
retrieve NP_3-chunk
pop NP_3
unify NP_3 with =NP of DP-chunk
pop DP_3
unify DP with =DP of VP-chunk
retrieve DP_4-chunk
push DP_4-chunk
retrieve NP_4-chunk
pop NP_4
unify NP with =NP of DP-chunk
pop DP_4
unify DP with =DP of VP-chunk
pop VP_2
unify VP with =VP of S-chunk
pop S-gap-chunk
unify S with RelC-chunk

The above pattern of rules tracks the processing of a RelC containing a DO prime. Upon completing this, the processor pops the RelC, retrieves the NP, and places it in the retrieval buffer. After it is popped, it unifies with the open =NP value in the RelC-chunk's 'mod' feature.

All the open values of the RelC are now resolved, and it can be popped from the buffer system.

'process sentence'
retrieve S-chunk
push S-chunk
retrieve DP_1-chunk
push DP_1-chunk
retrieve NP_1-chunk
pop NP_1
unify NP_1 with =NP of DP_1
pop DP-chunk
unify DP_1 with =DP of S
retrieve VP_1-chunk
push VP_1-chunk
retrieve DP_2-chunk
push DP_2-chunk
retrieve NP_2-chunk
pop NP_2-chunk
retrieve RelC
push RelC
retrieve RelP
pop RelP
unify RelP with =Rel P of RelC
retrieve S_2-chunk
push S_2-chunk
retrieve VP_2-chunk
push VP_2-chunk
retrieve DP_3-chunk
push DP_3-chunk
retrieve NP_3-chunk
pop NP_3
unify NP_3 with =NP of DP-chunk
pop DP_3
unify DP with =DP of VP-chunk
retrieve DP_4-chunk
push DP_4-chunk
retrieve NP_4-chunk
pop NP_4
unify NP with =NP of DP-chunk
pop DP_4
unify DP with =DP of VP-chunk
pop VP_2
unify VP with =VP of S-chunk
pop S-gap-chunk
unify S with RelC-chunk
retrieve NP_2
pop NP_2
unify NP with =NP in RelC-chunk
pop RelC → ***send to LTM***

The processor picks up the processing of the next subgoal in the problem state, in this case the 'process NP' subgoal associated with the DP-chunk's open =NP value.

'process sentence'	
retrieve S-chunk	
push S-chunk	
retrieve DP_1-chunk	
push DP_1-chunk	
retrieve NP_1-chunk	
pop NP_1	
unify NP_1 with =NP of DP_1	
pop DP-chunk	
unify DP_1 with =DP of S	
retrieve VP_1-chunk	
push VP_1-chunk	
retrieve DP_2-chunk	
push DP_2-chunk	
retrieve NP_2-chunk	
pop NP_2-chunk	
retrieve RelC	
push RelC	
retrieve RelP	
pop RelP	
unify RelP with =Rel P of RelC	
retrieve S-gap-chunk	
push S-gap-chunk	
retrieve VP_2-chunk	
push VP_2-chunk	
retrieve DP_3-chunk	
push DP_3-chunk	
retrieve NP_3-chunk	
pop NP_3	
unify NP_3 with =NP of DP-chunk	
pop DP_3	
unify DP with =DP of VP-chunk	
retrieve DP_4-chunk	
push DP_4-chunk	
retrieve NP_4-chunk	
pop NP_4	
unify NP with =NP of DP-chunk	
pop DP_4	
unify DP with =DP of VP-chunk	
pop VP_2	
unify VP with =VP of S-chunk	
pop S-gap-chunk	
unify S with RelC-chunk	
retrieve NP_2	
pop NP_2	
unify NP with =NP in RelC-chunk	
pop RelC	→ ***send to LTM***
retrieve NP_2	
pop NP_2	
unify NP with =NP in DP-chunk	
pop DP_2	
unify DP with =DP in VP-chunk	
pop VP_1	
unify VP with=VP in S	
pop S_1	
unify S with control state	

Just as in the previous relative clause sentence, the processor formed two unification chains

during the processing of this sentence. Although the two chains are associated with the NP-*duke*-chunk, they are distinct. Table 4.19 contains all the rules associated with the two unification chains.

Table 4.19: Unification chains for prime embedded in relative clause

"the king loved the duke"	**"who promised the duchess the rubies"**
retrieve S-chunk	retrieve RelC
push S-chunk	retrieve RelP
retrieve DP_1-chunk	pop RelP
push DP_1-chunk	***unify RelP with =RelP in RelC***
retrieve NP_1-chunk	retrieve S-gap-chunk
pop NP_1	push S-gap-chunk
unify NP_1 with =NP of DP_1	retrieve VP_2-chunk
pop DP-chunk	push VP_2-chunk
unify DP_1 with =DP of S	retrieve DP_3-chunk
retrieve VP_1-chunk	push DP_3-chunk
push VP_1-chunk	retrieve NP_3-chunk
retrieve DP_2-chunk	pop NP_3-chunk
push DP_2-chunk	***unify NP_3 with =$NP_4$of DP_3***
retrieve NP_2-chunk	pop DP_3-chunk
pop NP_2-chunk	***unify DP_3 with =DP_3 of VP***
retrieve NP_2-chunk	retrieve DP_4-chunk
pop NP_2-chunk	push DP_4-chunk
unify NP_2 with =NP_2 of DP_2	retrieve NP_4-chunk
pop DP_2-chunk	pop NP_4-chunk
unify DP_2 with =DP_2 of VP_1	***unify NP_4 with =NP_4 of DP_4***
pop VP_1-chunk	pop DP_4-chunk
unify VP with =VP of S_1	***unify DP_4 with =DP_4 of VP_2***
pop S_1-chunk	pop VP_2-chunk
unify S with control state	***unify VP_2 with =VP of S_1***
	pop S-gap-chunk
	unify S with =S in RelC-chunk
	retrieve NP_2-chunk
	pop NP_2
	unify NP with =NP in RelC-chunk
	pop RelC
	➔ ***send to LTM***

Recall that the results from the Relative Clause (RC) condition for both Experiment 1 and Experiment 2 found no significant difference between the embedded and matrix primes. Priming was equally possible from either position. Both SAP and PRICE claim that there should not be any difference between these two cases but for different reasons. SAP contends that structural

context never affects priming. The results from the VC condition in Experiment 2 challenge this contention. PRICE suggests that there should be similar patterns of priming because the number of rule firings associated with the processing of the structural contexts in which the primes occurred are almost equal, as demonstrated in Table 4.20 below.

Table 4.20: Comparison of unification chains for relative clause sentences

Matrix clause	Relative clause
retrieve S-chunk	retrieve RelC
push S-chunk	retrieve RelP
retrieve DP_1-chunk	pop RelP
push DP_1-chunk	***unify RelP with =RelP in RelC***
retrieve NP_1-chunk	retrieve S-gap-chunk
pop NP_1	push S-gap-chunk
retrieve NP_1-chunk	retrieve VP_2-chunk
pop NP_1	push VP_2-chunk
unify NP_1 with =NP of DP_1	retrieve DP_3-chunk
pop DP-chunk	push DP_3-chunk
unify DP_1 with =DP of S	retrieve NP_3-chunk
retrieve VP-chunk	pop NP_3-chunk
push VP-chunk	***unify NP_3 with =$NP_4$of DP_3***
retrieve DP-chunk	pop DP_3-chunk
push DP-chunk	***unify DP_3 with =DP_3 of VP***
retrieve NP-chunk	retrieve DP_4-chunk
pop NP-chunk	push DP_4-chunk
unify NP_2 with =NP_2 of DP_2	retrieve NP_4-chunk
pop DP-chunk	pop NP_4-chunk
unify DP_2 with =DP_2 of VP	***unify NP_4 with =NP_4 of DP_4***
retrieve DP-chunk	pop DP_4-chunk
push DP-chunk	***unify DP_4 with =DP_4 of VP_2***
retrieve NP-chunk	pop VP_2-chunk
pop NP-chunk	***unify VP_2 with =VP of S_1***
unify NP_3 with =NP_3 of DP_3	pop S-gap-chunk
pop DP-chunk	***unify S with =S in RelC-chunk***
unify DP_3 with =DP_3 of VP	retrieve NP_2-chunk
pop VP-chunk	pop NP_2
unify VP with =VP of S	***unify NP with =NP in RelC-chunk***
pop S-chunk	pop RelC
23 rules	22 rules

As discussed in previous sections, utility is affected by the number of rules necessary for the processing of a particular structure. The more rules, the lower the utility of each single rule. Because the unification chains in the RC conditions are associated with approximately the same number of rule firings (23 for the case in which prime is in matrix position with relative clause sentence and 22 for the case in which the prime is embedded in the relative clause sentence),

they also had similar utility scores.

Although there was no difference between the matrix and embedded primes in the RC condition, there was an effect of lag such that priming was stronger from both positions at longer lags. Previous experiments have found a slight increase in priming after an initial drop following one filler item, but in general, priming remains stable over time (Bock & Griffin 2000, Hartsuiker *et al.* 2008, Ferreira *et al.* 2005). The increase found here may be part of the normal increase found in these previous studies. Another possibility is that there is initially less priming due to the fact that the processor must saturate the gap list and thereby the empty '___' associated with the 'spec' feature of the S-gap-chunks with the information from the filler element (i.e. the NP-*duke*-chunk that unifies with the open =NP value of the RelC-chunk. As the memories consolidate, this additional processing burden wanes, and the priming from these sentences returns to normal, mirroring priming from other clauses that are associated with unification chains of similar lengths. I explore this idea again in Chapter 5.

Table 4.21 contains a list of all the rules necessary for processing the four different structural contexts tested in the VC and RC conditions.

Table 4.21: Comparison of rule firings

In matrix clause of sentence with adverbial clause	In verb complement clause	In matrix clause of relative clause sentence	In relative clause
retrieve S-chunk	retrieve S-chunk	retrieve S-chunk	retrieve RelC
push S-chunk	push S-chunk	push S-chunk	retrieve RelP
retrieve DP-chunk	retrieve DP_1-chunk	retrieve DP_1-chunk	pop RelP
push DP-chunk	push DP_1-chunk	push DP_1-chunk	***unify RelP with =RelP in RelC***
retrieve NP-chunk	retrieve NP_1-chunk	retrieve NP_1-chunk	retrieve S-gap-chunk
pop NP-chunk	pop NP_1	pop NP_1	push S-gap-chunk
unify NP_1 with =NP of DP_1	***unify NP with =NP in DP-chunk***	retrieve NP_1-chunk	retrieve VP_2-chunk
pop DP-chunk	pop DP_1	pop NP_1	push VP_2-chunk
unify DP_1 with =DP of S	***unify DP with =DP in S-chunk***	***unify NP_1 with =NP of DP_1***	retrieve DP_3-chunk
retrieve VP-chunk	retrieve VP_1-chunk	pop DP-chunk	push DP_3-chunk
push VP-chunk	push VP_1-chunk	***unify DP_1 with =DP of S***	retrieve NP_3-chunk
retrieve DP-chunk	retrieve CP-chunk	retrieve VP-chunk	pop NP_3-chunk
push DP-chunk	push CP-chunk	push VP-chunk	***unify NP_3 with =$NP_4$of DP_3***
retrieve NP-chunk	retrieve Comp-chunk	retrieve DP-chunk	pop DP_3-chunk
pop NP-chunk	pop Comp-chunk	push DP-chunk	***unify DP_3 with =DP_3 of VP***
unify NP_2 with =NP_2 of DP_2	***unify Comp with CP***	retrieve NP-chunk	retrieve DP_4-chunk
pop DP-chunk	retrieve S_2-chunk	pop NP-chunk	push DP_4-chunk
unify DP_2 with =DP_2 of VP	push S_2-chunk	***unify NP_2 with =NP_2 of DP_2***	retrieve NP_4-chunk
retrieve DP-chunk	retrieve DP_1-chunk	pop DP-chunk	pop NP_4-chunk
push DP-chunk	push DP_1-chunk	***unify DP_2 with =DP_2 of VP***	***unify NP_4 with =NP_4 of DP_4***
retrieve NP-chunk	retrieve NP_1-chunk	retrieve DP-chunk	pop DP_4-chunk
pop NP-chunk	pop NP_2	push DP-chunk	***unify DP_4 with =DP_4 of VP_2***
unify NP_3 with =NP_3 of DP_3	***unify NP with =NP in DP-chunk***	retrieve NP-chunk	pop VP_2-chunk
pop DP-chunk	pop DP_2	pop NP-chunk	***unify VP_2 with =VP of S_1***
unify DP_3 with =DP_3 of VP	***unify DP with =DP in S-chunk***	***unify NP_3 with =NP_3 of DP_3***	pop S-gap-chunk
pop VP-chunk	retrieve VP-chunk	pop DP-chunk	***unify S with =S in RelC-chunk***
unify VP with =VP of S	push VP-chunk	***unify DP_3 with =DP_3 of VP***	retrieve NP_2-chunk
pop S-chunk	retrieve DP_3-chunk	pop VP-chunk	pop NP_2
	push DP_3-chunk	***unify VP with =VP of S***	***unify NP with =NP in RelC-chunk***
	retrieve NP_3-chunk	pop S-chunk	pop RelC
	pop NP_3		
	unify NP with =NP in DP-chunk		
	pop DP_3		
	unify DP with =DP in VP-chunk		
	retrieve DP_4-chunk		
	push DP_4-chunk		
	retrieve NP_4-chunk		
	pop NP_4		
	unify NP with =NP in DP-chunk		
	pop DP_4		
	unify DP with =DP in VP-chunk		
	pop VP		
	unify VP with =VP in S-chunk		
	pop S		
	unify S with CP-chunk		
	pop CP		
	unify CP with VP_1-chunk		
	pop VP_1		
	unify VP with S_1-chunk		
	pop S		
21 rules	40 rules	23 rules	22 rules

What this table should make abundantly clear is the significant difference in the rules necessary

for the processing of each of the prime's structural contexts. The prime occurring in a verb complement clauses is associated with 40 rule firings, whereas the primes in the other three contexts are associated with approximately 22 rule firings (matrix with adverbial clause: 21; matrix with relative clause: 23; embedded in relative clause: 22). Consider again how utility is calculated:

$$\mathbf{U = PG - C} \qquad \text{Utility}$$

Utility is determined by the probability *P* that a rule achieves its intended effect and that it leads to the successful completion of the goal *G* minus the cost *C* associated with firing the rule. As the number of rules increases, the probability of success decreases and the cost increases, thereby lowering the utility of each rule. For example, cost is calculated by adding the cost of firing the particular rule *a* and the estimated cost of all subsequent rules *b*.

$$\mathbf{C = a + b} \qquad \text{Associated cost}$$

Assuming that *a* remains constant (i.e. firing a 'retrieve VP' rule always costs the same amount), any increase in *b* leads to lower utility. *b* increases as the number of rules in the context increases. For instance, consider the number of rules associated with the processing of the single-clause sentence "The duke promised the duchess the rubies" (21 rules) versus the sentence with the verb complement clause "The report declared that the duke promised the duchess the rubies" (40 rules). If we consider the firing of the first 'retrieve DP' rule, there are 20 other firings that must occur to process the single-clause sentence, whereas there are 39 that must fire to process the verb complement clause sentence. Thus, the firing of the same rule is more costly in the verb

complement clause due to the number of other rules that must fire in order to make a grammatical sentence.

The fact that primes in verb complement clauses have lower utility scores may lead one to predict that primes in verb complement clauses always prime less than primes in other positions. However, utility is not the only factor affecting priming behavior. Rule strength also determines how likely a rule is to be retrieved. At the shorter lag, the production rule strength is sufficient to lead to priming for primes in verb complement clauses as found in Experiment 1. Priming waned only after a delay, when the activation (hence strength) had decayed and utility became the primary decision-making factor.

6. Conclusion

In this chapter, I explored the PRICE claim that structural context mediates the effects of recency on structural priming. The SAP account claimed that all structural primes should demonstrate the same pattern of priming behavior regardless of the structural context in which the prime occurred. PRICE predicted differences among the structural contexts. The results from Experiments 1 and 2 support PRICE.

The results from these studies indicate that structural primes in verb complement clauses do not lead to stable structural priming over time. Rather, priming disappeared at longer lags. PRICE claimed that differences among the different structural contexts are due to the way memory represents the application of production rules. The model of language processing presented in Chapter 2 claimed that the argument/adjunct distinction is a crucial factor in

determining which structural contexts are relevant. This model of language processing contends that the retrieval of production rules depends on the strength and utility of a given rule. Determining a rule's strength depends on both the overall history of use and the recency of use. Determining a rule's utility depends on estimating the probability of success and cost associated with the rule. The utility of rules associated with long unification chains is more difficult for the processor to assess than it would be if the chain were shorter. This difficulty leads to lower utility scores relative to those associated with rules occurring in short unification chains. When a particular rule's strength has waned due to natural activation decay, the effects of lower utility scores manifest and weaker structural priming surfaces. That is, the amount of structural priming from long unification chains is lower than the amount from short unification chains. We return to the effects of production rule strength and utility and their impact on structural priming in Chapter 5.

Appendix 4A: Experimental items for the structural priming experiments

The materials for the two embedded conditions are given first, starting with the relative clause condition and then the verb complement clause condition. This is followed by the two matrix clause conditions. All the primes are shown with the double object dative form.

Embedded Primes

Primes and targets for relative clause condition	NP1	NP2	VERB
The dean spoke with the graduates who baked the professors the cookies.			
Target: The organist gossiped about the neighbor who . . .	**Brownies**	**pastor**	**BAKE**
The auditor knew the manager who handed the customer the cash.			
Target: The valet talked with the customer who . . .	**waiter**	**menu**	**HAND**
The neighbors knew the mother who promised the girl the ring.			
Target: The king befriended the lord who . . .	**rubies**	**duchess**	**PROMISE**
The gang paid the inspector who offered the bar owner the bribe.			
Target: The judge called the attorney who . . .	**thief**	**deal**	**OFFER**
The social worker met the tenant who owed the landlord the rent.			
Target: The news quoted the captain who . . .	**complement**	**lieutenant**	**OWE**
The bank called the salesman who sold the couple the Jeep.			
Target: The butcher carpooled with the grocer who . . .	**shopper**	**walnuts**	**SELL**
The cop talked to the waitress who served the executive the martini.			
Target: The expert interviewed the hostess who . . .	**quilters**	**pastries**	**SERVE**
The psychologist interviewed the child who showed the officer the coloring book.			
Target: The principal confided in the teacher who . . .	**raters**	**answers**	**SHOW**

Primes and targets for verb complement clause condition	NP1	NP2	VERB
The letter alleged that the graduates baked the professors the cookies.			
Target: The paper stated that the neighbor . . .	**brownies**	**pastor**	**BAKE**
The report disclosed that the manager handed the customer the cash.			
Target: The film revealed that the customer . . .	**waiter**	**menus**	**HAND**

The headline alleged that the mother promised the girl the ring.

Target:	The photograph disclosed that the lord . . .	**rubies**	**duchess**	**PROMISE**

The video revealed that the inspector offered the bar owner the bribe.

Target:	The paper claimed that the attorney . . .	**thief**	**deal**	**OFFER**

The report stated that the tenant owed the landlord the rent.

Target:	The headline revealed that the captain . . .	**complement**	**lieutenant**	**OWE**

The headline declared that the salesman sold the couple the Jeep.

Target:	The paper revealed that the grocer . . .	**shopper**	**walnuts**	**SELL**

The rumors alleged that the waitress served the executive the martini.

Target:	The report claimed that the hostess . . .	**quilters**	**pastries**	**SERVE**

The report stated that the child showed the officer the coloring book.

Target:	The documents alleged that the teacher . . .	**raters**	**answers**	**SHOW**

Matrix primes

Primes and targets for relative clause condition		**NP1**	**NP2**	**VERB**
The mayor who cited the newspaper awarded the fireman the medal.				
Target:	The man who sat next to the parents . . .	**winner**	**trophy**	**AWARD**
The pilot who recommended the company bought the flight crew the drinks.				
Target:	The sailor who married the teacher . . .	**children**	**candies**	**BUY**
The toddler who kissed the aunt fed the rabbit the carrot.				
Target:	The nanny who scolded the visitor . . .	**twins**	**cake**	**FEED**
The clerk who emailed the temp issued the typist the key.				
Target:	The trooper who contacted the station . . .	**ticket**	**driver**	**ISSUE**
The swimmer who questioned the coach lent the diver the towel.				
Target:	The seamstress who met the groom . . .	**dress**	**bride**	**LEND**
The teenager who saw the teacher passed the student the note.				
Target:	The fan who kissed the guitarist . . .	**drummer**	**cigars**	**PASS**
The boy who knew the magician taught the girl the trick.				
Target:	The social worker who phoned the activists . . .	**migrants**	**english**	**TEACH**

The lifeguard who warned the crowd threw the surfer the life vest.				
Target:	The pitcher who loved the fans . . .	**coach**	**ball**	**THROW**

Primes and targets for verb complement clause condition		**NP1**	**NP2**	**VERB**
As the newspaper claimed, the mayor awarded the fireman the medal.				
Target:	As the rumors alleged, the man . . .	**winner**	**trophy**	**AWARD**
As the paper reported, the pilot bought the flight crew the drinks.				
Target:	As the video revealed, the sailor . . .	**children**	**candies**	**BUY**
As the report claimed, the toddler fed the rabbit the carrot.				
Target:	As the document alleged, the nanny . . .	**twins**	**cake**	**FEED**
As the film revealed, the temp issued the typist the key.				
Target:	As the video alleged, the trooper . . .	**ticket**	**driver**	**ISSUE**
As the photograph revealed, the swimmer lent the diver the towel.				
Target:	As the program claimed, the seamstress . . .	**dress**	**bride**	**LEND**
As the rumors claimed, the teenager passed the student the note.				
Target:	As the report declared, the fan . . .	**drummer**	**cigars**	**PASS**
As the documents revealed, the boy taught the girl the trick.				
Target:	As the paper stated, the social worker . . .	**migrants**	**english**	**TEACH**
As the program alleged, the lifeguard threw the surfer the life vest.				
Target:	As the headline disclosed, the pitcher . . .	**coach**	**ball**	**THROW**

Appendix 4B: Filler items for the structural priming experiments

Two-place predicates full and fragment sentences	**Word 1**	**Word 2**	**Verb**
The careless delivery boy put the pizza in the trunk.			
The flight attendant put the coat in the compartment.			
The senior librarian put the book on the shelf.			
The considerate biologist put the sample in the refrigerator.			
The siblings put their parents' anniversary picture on the refrigerator.			
The florist placed the lilies in the bottle.			
The cautious dentist placed the tools on the tray.			
The overly sentimental aunt placed the card on the mantel.			
The junior high lunch lady placed the mashed potatoes on the student's tray.			
The guitarist and the pianist placed their differences aside.			
The best friends each put the others' picture in a frame.			
The law student . . .	article	folder	PUT
The trainer . . .	weights	rack	PUT
The overwhelmed and underpaid secretary . . .	document	shredder	PUT
The mime and street musicians both . . .	names	list	PUT
Both the acrobat and the clown put . . .	makeup	faces	PUT
The travel agent . . .	ticket	envelope	PLACE
The well-liked mailman happily . . .	package	doorstep	PLACE
The conservative satirical columnist . . .	receipt	purse	PLACE
The insurance adjustor accidentally . . .	claim	briefcase	PLACE
The happy but careless newlyweds . . .	presents	closet	PLACE
The stressed surgeon intentionally . . .	x-ray	file	PLACE
***Where*-clause full and fragment sentences**			
The repairman fixed the hole where the crack was.			
The teenager parked where the cool kids smoked.			
The girl looked where the kittens were.			
The boys visited the zoo where the giraffes live.			
The children yelled up the tree where the boy waited.			
The cowboy saw where the steer waited.			
The man knew where the mouse hid.			
The teacher placed the answer sheet where the quiz was.			
The girl looked where the doll and the lamp stood.			
The maid discovered where the children hid.			
The second grader found where the cookies were kept.			
The children and parents all knew where . . .	babysitter	magazine	HIDE
The gardener dug where . . .	bulbs	spring	GROW

The ranger placed the sign where . . .	hunters	deer	GATHER
The chef chopped the vegetables where . . .	spice	jars	BE
The couple walked toward the stand where . . .	driver	cab	WAIT
The divorcee and the lawyer both knew where . . .	corporate	investments	BE
The author wrote where . . .	poet	sister	DIE
The ballerina practiced where . . .	musicians	blues	MEET
The reporter ran to where . . .	accident	train	HAPPEN
The artist remembered where . . .	collector	painting	HANG
The acrobat walked where . . .	rope	taunt	BE

Finite clause complement full and fragment sentences

The expert certified that the broken antique was genuine.
The marketing students discovered that the new ad campaign was plagiarized.
The review board confirmed that the results were valid.
Even the NRA considers that legalizing grenades is dangerous.
The physician diagnosed that the soprano's tumor was benign.
The woman discovered that her fiancée was completely untrustworthy.
The professor expected that the worst student was an athlete.
The young couple felt that the condominium was too small for the family.
The suspicious husband guessed that his wife's excuse was false.
The plumber hypothesized that the problem was internal.
The representative imagined that the other party's candidate was an idiot.

The expert affirmed . . .	document	forged	BE
The critic confessed . . .	favorite	jazz	BE
The airline employee confirmed . . .	jet	late	BE
The dog show judge considered . . .	beagles	superior	BE
The visiting professor conceded. .	research	questionable	BE
The administrator discovered . . .	manager	incompetent	BE
The OBGYN acknowledged . . .	couple	pregnant	BE
The UN translator felt . . .	diplomat	fair	BE
The uncle guessed . . .	nephew	honest	BE
The humanitarian organization hypothesized . . .	reporter	helpful	BE
The whistle blower foresaw . . .	inspector	corrupt	BE

Object-control full and fragment sentences

The father persuaded the girl to be a careful skier.
The woman forced her husband to be neater around the house.
The librarian somehow convinced the researcher to be quiet.
The boss encouraged his servant to be faster.
The child convinced the clown to make a balloon dog.

The babysitter encouraged the employer to be a better parent.
The son allowed the father to be the teacher for a day.
The roommates encouraged each other to be more studious.
The dog begged the owner to be more generous with the food.
The crew persuaded the captain to stop drinking whiskey.
The company forced the CEO to resign his post.

The nun allowed . . .	priest	late	BE
The activist encouraged . . .	community	proactive	BE
The neighbor begged . . .	gardener	Early	BE
The new employee begged . . .	mentors	available	BE
The student asked . . .	tutor	specific	BE
The reporter persuaded . . .	informant	honest	BE
The sergeant encouraged . . .	recruits	active	BE
The new rules forced . . .	applicants	competitive	BE
The calm friend convinced . . .	president	patient	BE
The tyrant forced . . .	editor	flattering	BE
The fraternity's president allowed . . .	members	diverse	BE

Appendix 4C: Instructions used in the structural priming experiment 1 and 2

Welcome and thank you for participating.

In the first phase of this study, you will be asked to do two tasks. In one task, you will be asked to read sentences out loud, while in another task, you will be asked to finish incomplete sentences using a set of specific words. Following these tasks, you will be shown another series of sentences, and you will be asked whether you remember seeing them or not.

For the first phase, you will see some slides with the word "READ" at the top. When you see this prompt, you should first read the sentence presented on the screen silently to yourself, and then read it out loud. Be sure to read the sentence carefully and say it aloud clearly and accurately. When you are done, press the space bar for the next slide.

On other slides, you will see the word "COMPLETE". Underneath this word will be an incomplete sentence.

Read the incomplete sentence silently to yourself. Then, press the SPACE BAR, and 3 words will appear.

The bottom word, in all CAPS, will be a verb, and the other two words will be nouns or adjectives. These are the words you should use to complete the sentence.

Read the 3 words silently, and then read the sentence fragment aloud and finish it using the additional words. You may need to change the form of the verb or the order of the words, or add prepositions or articles ("a"/"the") in order to make a complete sentence. Consider the following example:

"Yesterday, Robert . . ."

cafeteria
kiwi
EAT

POSSIBLE RESPONSE:
"Yesterday, Robert ate a kiwi in the cafeteria."

As you can see in this example, the verb "EAT" needed to be in the past tense "ATE", and both of the nouns needed an article, and "cafeteria" needed a preposition to make sense.

Now you will try a few practice slides.

Remember to just read aloud the "READ" slides as accurately as possible.

And for the "COMPLETE" slides, you may add words as necessary, but avoid adding too many.

The most important thing is that YOU MUST USE ALL THE WORDS that appear.

Try to produce your responses as quickly as possible, but do not spend too much time with any single response.

Practice 1:
READ
Looking around for thirty minutes, the lab tech searched the cafeteria for his friends.

Practice 2:
READ
The philosopher whose book was recently published danced with the sailor all night.

Practice 3:
COMPLETE
The musician whose guitar is on auction. . .
hall
waltz
PERFORM

Practice 4:
READ
Hoping to avoid the police, the smugglers jumped over the fence into the backyard.

Practice 5:
COMPLETE
Rubbing her forehead, the artist . . .
canvas
picture
PAINT

Practice 6:
READ
The sleeping child was woken up by the loud music.

Practice 7:
READ
Knowing that it would cost him his job, the whistle-blower confronted his boss.

Practice 8:
COMPLETE
The old church tower was . . .
bolt
lightning
STRIKE

Practice 9:
COMPLETE
The ballerina knew the dancer whose . . .
friend
girl
GRADUATE

Now you are ready for the experiment.

As you move through the experiment, remember to read the "READ" slides carefully and to give your first response to the "COMPLETE" slides smoothly.

When you are done with this phase, you will be given a memory test to see if you recognize any of the sentences from the experiment. Therefore, you should read them carefully.

If you have any questions, please ask the experimenter now. Otherwise, proceed….

Appendix 4D: Regression models

1. Experiment 1: Short Lag of 1

1.1 Relative clause condition (RC)

A) RC main effects at lag of 1: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.95
Verb (16)	0.70

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.08	0.28	0.27	0.78
Baseline & PD	0.06	0.17	0.36	0.72
PD & DO	-0.22	0.11	-2.00	0.05*
Position	-0.10	0.37	-0.28	0.78

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) RC interaction at lag of 1: Parameter values for the fixed effect prime and position and their interaction in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.97
Verb (16)	0.70

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.08	0.28	0.27	0.78
Baseline & PD	0.05	0.18	0.27	0.79
PD & DO	-0.18	0.18	-1.00	0.32
Position	-0.10	0.37	-0.28	0.78
Baseline & PD*Position	0.02	0.17	0.12	0.91
PD & DO*Position	-0.08	0.30	-0.26	0.79

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check of main effects and interaction for RC at lag of 1: There was no significant difference between the main effects and interaction model $\chi^2(2, N = 90) = 0.07$, $p = 0.97$.

1.2 Verb complement clause condition (VC)

A) VC main effects at lag of 1: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.94
Verb (16)	0.64

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.26	0.00	0.99
Baseline & PD	0.18	0.17	1.08	0.28
PD & DO	-0.43	0.11	-3.82	0.001***
Position	-0.20	0.34	-0.57	0.57

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) VC interaction at lag of 1: Parameter values for the fixed effect prime and position and their interaction in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.94
Verb (16)	0.64

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.26	0.00	0.99
Baseline & PD	0.17	0.18	0.93	0.35
PD & DO	-0.48	0.19	-2.58	0.01**
Position	-0.19	0.34	-0.56	0.57
Baseline & PD*Position	0.02	0.17	0.10	0.92
PD & DO*Position	0.10	0.30	0.33	0.75

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) **Model check of main effects and interaction for VC at lag of 1:** There was no significant difference between the main effects and interaction model $\chi^2(2, N = 90) = 0.15$, $p = 0.93$.

1.3 Comparing the relative clause and the verb complement clause condition at lag 1

A) **Interaction between Condition (RC/VC) and Position at lag of 1:** Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, matrix clause primes were compared to embedded clause primes, and the verb complement clause version was compared to the relative clause version.

Random Effects

	Standard deviation
Participants (180)	0.95
Verb (16)	0.66

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.27	-0.01	0.99
Baseline & PD	0.12	0.12	1.02	0.31
PD & DO	-0.32	0.08	-4.11	0.001***
Position	-0.21	0.35	-0.59	0.56
Condition	0.08	0.18	0.44	0.66
Position*Condition	0.11	0.17	0.67	0.51

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) All possible interactions among condition (RC/VC) and position at lag of 1: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, matrix clause primes were compared to embedded clause primes, and the verb complement clause version was compared to the relative clause version.

Random Effects

	Standard deviation
Participants (180)	0.95
Verb (16)	0.66

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.00	0.27	-0.01	0.99
Baseline & PD	0.17	0.18	0.94	0.35
PD & DO	-0.48	0.18	-2.59	0.01**
Position	-0.21	0.35	-0.58	0.56
Condition	0.08	0.18	0.44	0.66
Baseline & PD*Position	0.02	0.17	0.11	0.92
PD & DO*Position	0.10	0.30	0.33	0.74
Baseline & PD*Condition	-0.12	0.26	-0.48	0.63
PD & DO*Condition	0.29	0.26	1.13	0.26
Position*Condition	0.11	0.17	0.67	0.51
Baseline & PD* Position*Condition	0.00	0.24	0.001	1.00
PD & DO*Position*Condition	-0.17	0.42	-0.41	0.68

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check of position and version interaction and all the possible interactions models for RC and VC lag 1: There was no significant difference between the two models $\chi^2(6, N = 180) = 2.31$, $p = 0.89$.

2. Experiment 2: Long Lag of 3

2.1 Relative clause condition lag 3

A) RC main effects at lag of 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO

completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	1.06
Verb (16)	0.71

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	-0.14	0.29	-0.47	0.64
Baseline & PD	0.29	0.18	1.63	0.10
PD & DO	-0.54	0.12	-4.47	0.001***
Position	-0.14	0.38	-0.37	0.72

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) RC interaction at lag of 3: Parameter values for the fixed effect prime and position and their interaction in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (60)	1.15
Verb (16)	0.70

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	-0.13	0.29	-0.47	0.64
Baseline & PD	0.27	0.20	1.34	0.18
PD & DO	-0.51	0.20	-2.55	0.01*
Position	-0.14	0.38	-0.37	0.71
Baseline & PD*Position	0.07	0.17	0.38	-0.71
PD & DO*Position	-0.05	0.33	-0.15	0.88

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check of main effects and interaction for RC at lag of 3: There was no significant difference between the main effects and interaction model $\chi^2(2, N = 90) = 0.14$, $p = 0.93$.

2.2 Relative clause condition at lag 1 and lag 3

A) RC main effects at lag of 1 and 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

	Standard deviation
Participants (120)	1.07
Verb (16)	0.72

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.04	0.36	-0.14	0.89
Prime	0.58	0.11	5.38	0.001***
Position	-0.14	0.37	-0.38	0.70
Lag	-0.17	0.11	-1.56	0.12

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) RC interaction of Position*Lag at lag of 1 and 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared to DO baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

	Standard deviation
Participants (120)	1.07
Verb (16)	0.72

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.08	0.37	-0.22	-0.82
Prime	0.59	0.11	5.37	0.001***
Position	-0.07	0.43	-0.16	0.87
Lag	-0.16	0.12	-1.28	0.20
Position*Lag	-0.04	0.11	-0.34	0.73

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check for RC lag 1 and lag 3 main effects and Position*Lag interaction: There was no significant difference between the main effects and interaction model $\chi^2(1, N = 120) = 0.12$, $p = 0.74$.

D) RC all possible interactions at lag of 1 and 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared to DO

baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

	Standard Deviation
Participant (120)	1.07
Verb (16)	0.72

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	0.15	0.46	0.32	0.75
Prime	0.12	0.55	0.21	0.84
Position	-0.16	0.66	-0.24	0.81
Lag	-0.27	0.18	-1.52	0.13
Prime*Position	0.18	1.00	0.18	0.86
Prime*Lag	0.22	0.25	0.88	0.38
Position*Lag	-0.02	0.25	-0.04	0.97
Prime*Position*Lag	-0.05	0.45	-0.12	0.91

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

E) Model check for RC lag 1 and lag 3 main effects and full interaction: There was no significant difference between the main effects and interaction model $\chi^2(4, N = 120) = 3.10$, $p = 0.54$.

2.3 Verb clause version lag 3

A) VC main effects at lag of 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.85
Verb (16)	0.68

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.25	0.30	0.94	0.35
Baseline & PD	-0.01	0.15	-0.09	0.93
PD & DO	-0.11	0.11	-1.05	0.29
Position	-0.37	0.36	-1.02	0.31

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) VC interaction at lag of 3: Parameter values for the fixed effect prime and position and their interaction in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes.

Random Effects

	Standard deviation
Participants (90)	0.85
Verb (16)	0.68

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.25	0.27	0.94	0.35
Baseline & PD	-0.08	0.17	-0.47	0.64
PD & DO	0.22	0.17	1.30	0.19
Position	-0.36	0.36	-1.00	-0.32
Baseline & PD*Position	0.14	0.17	0.81	0.42
PD & DO*Position	-0.68	0.27	-2.51	0.01*

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check for VC main effects and interaction at lag of 3: There was a significant difference between the main effects and interaction model $\chi^2(2, N = 90) = 6.02$, $p < 0.05$, difference in log likelihood = 3.01.

2.4 Verb complement clause version at lag 1 and lag 3

A) VC lag 1 and lag 3 main effects: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

Groups	Standard Deviation
Participant (120)	0.85
Verb (16)	0.65

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.44	0.32	-1.40	0.16
Position	0.47	0.11	4.49	0.001***
Prime	-0.31	0.34	-0.91	0.36
Lag	0.15	0.09	1.62	0.11

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) VC lag 1 and lag 3 Position*Lag interaction: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

Groups	Standard Deviation
Participant (120)	0.88
Verb (16)	0.65

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.55	0.33	-1.67	0.10
Prime	0.47	0.11	4.48	0.001***
Position	-0.07	-0.40	-0.18	0.86
Lag	0.21	0.11	1.96	0.05.
Position*Lag	-0.12	0.11	-1.13	0.26

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check for RC lag 1 and lag 3 main effects and Position*Lag interaction: There was no significant difference between the main effects and interaction model $\chi^2(1, N = 120) = 1.21$ $p = 0.27$.

D) VC lag 1 and lag 3 Prime*Position*Lag interactions: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the

baselines, DO completions were compared to PD baselines, and matrix clause primes were compared to embedded clause primes, and lag 3 results were compared to lag 1.

Random Effects

Groups	Standard Deviation
Participant (120)	0.86
Verb (16)	0.65

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-1.04	0.40	-2.58	0.01**
PD Prime	1.44	0.47	3.08	0.001**
Position	0.45	0.58	-0.79	0.43
Lag	0.51	0.14	3.42	0.001***
Prime*Position	-1.04	0.82	-1.27	0.21
Prime* Lag	-0.60	0.21	-2.86	0.00**
Position*Lag	-0.50	0.21	-2.33	0.02*
Prime*Position*Lag	0.75	0.37	2.05	0.04

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

E) Model check for VC lag 1 and lag 3 main effects and all possible interactions: There was no significant difference between the main effects and interaction model $\chi^2(4, N = 120) = 11.33$, $p = 0.02$, difference in log odds is 5.6.

2.5 Verb complement clause version at lag 1 and lag 3 with DO primes only

A) Interaction between Position and Lag for DO primes only: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. Matrix clause primes were compared to embedded clause primes, and lag 1 was compared to lag 2.

Random Effects

	Standard deviation
Participants (120)	0.95
Verb (16)	0.56

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	-0.96	0.41	-2.32	0.02*
Position	0.35	0.58	0.60	0.55
Lag	0.47	0.16	2.89	0.01**
Position*Lag	-0.46	0.23	-2.00	0.05*

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

2.6 Relative clause and verb complement clause versions at lag 3

A) Interaction between Condition (RC/VC) and Position at lag of 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, matrix clause primes were compared to embedded clause primes, and the verb complement clause version was compared to the relative clause version.

Random Effects

	Standard deviation
Participants (180)	0.95
Verb (16)	0.68

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.26	0.28	0.94	0.35
Baseline & PD	0.13	0.12	1.14	0.25
PD & DO	-0.32	0.08	-3.97	0.001***
Position	-0.35	0.36	-0.97	0.33
Condition	-0.38	0.19	-2.05	0.04*
Position*Condition	0.21	0.17	1.20	0.23

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

B) All possible interactions among condition (RC/VC) and position at lag of 3: Parameter values for the fixed effect prime and position in a generalized linear mixed model logistic regression of DO/PD responses, in log odds, and associated standard errors, z-scores, and probabilities. PD completions were compared against the baselines, DO completions were compared to PD baselines, matrix clause primes were compared to embedded clause primes, and the verb complement clause version was compared to the relative clause version.

Random Effects

	Standard deviation
Participants (180)	0.93
Verb (16)	0.69

Fixed Effects

	Estimate	Standard Error	z	P-value
Intercept	0.26	0.28	0.94	0.35
Baseline & PD	-0.08	0.18	-0.45	0.65
PD & DO	0.23	0.18	1.24	0.22
Position	-0.35	0.37	-0.96	0.34
Condition	-0.38	0.18	-2.08	0.04*
Baseline & PD*Position	0.14	0.17	0.81	0.42
PD & DO*Position	-0.70	0.30	-2.36	0.02*
Baseline & PD*Condition	0.34	0.26	1.30	0.19
PD & DO*Condition	-0.73	0.26	-2.79	0.01**
Position*Condition	0.21	0.17	1.18	0.24
Baseline & PD* Position*Condition	-0.08	0.24	-0.32	0.75
PD & DO*Position*Condition	0.66	0.42	1.56	0.12

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

C) Model check for RC and VC at lag 3 single interaction and all possible interactions model:
There was no significant difference between the main effects and interaction model $\chi^2(6, N = 180) = 12.53$, $p = 0.051$.

2.7 Comparing RC and VC at lag of 1 and 3 (just the PD and DO completions, no baseline)

A) Comparison of selected models for the compiled data organized by degrees of freedom and log likelihood

	Model	Df	AIC	BIC	Loglik
a)	Main effects	7	4360.9	4404.1	2173.5
b)	Interaction Lag*Condition	8	4359.0	4408.4	2171.5
c)	Interaction Position*Condition	8	4362.0	4411.4	2173.0
d)	Interaction Prime*Position	8	4362.2	4411.5	2173.1
e)	Interaction Lag*Position	8	4362.5	4411.8	2173.2
f)	Interaction Lag*Prime	8	4362.9	4412.3	2173.4
g)	Interaction Prime*Condition	8	4362.9	4412.2	2173.4
h)	Interaction Condition(Prime, Position, Lag)	10	4362.1	4423.7	2171.0
i)	Interaction Lag(Prime, Position, Condition)	10	4362.5	4424.2	2171.3
j)	Interaction Prime*Position*Condition	11	4366.8	4434.7	2172.4
k)	Interaction Lag*Condition(Prime, Position)	14	4364.5	4450.9	2168.3
l)	All possible interactions	18	4368.9	4479.9	2166.4

B) The regression results for the models organized by degrees of freedom and log likelihood

a) Main Effects

Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.19	0.30	-0.64	0.53
Prime	0.54	0.08	7.19	0.00
Position	-0.21	0.35	-0.60	0.55
Condition	-0.08	0.15	-0.57	0.57
Lag	-0.03	0.07	-0.38	0.71

b) Interaction Condition*Lag Random Effects

	Standard Deviation
Participants (240)	0.97
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	Z	p-value
Intercept	-0.48	0.34	-1.44	0.15
Prime	0.54	0.08	7.19	0.00
Position	-0.21	0.35	-0.60	0.55
Condition	0.50	0.33	1.53	0.13
Lag	0.12	0.10	1.14	0.25
Condition*Lag	-0.29	0.15	-2.00	0.05

c) Interaction Condition*Position Random Effects
Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Error	Z	p-value
Intercept	-0.16	0.31	-0.52	0.60
Lag	-0.03	0.07	-0.38	0.71
Prime	0.54	0.08	7.19	0.00
Position	-0.28	0.35	-0.80	0.43
Condition	-0.15	0.16	-0.93	0.35
Condition*Position	0.15	0.15	0.97	0.33

d) Interaction Prime*Position
Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Error	z	p-value
Intercept	-0.13	0.31	-0.42	0.68
Prime	0.42	0.16	2.57	0.01
Position	-0.34	0.38	-0.89	0.37
Condition	-0.08	0.15	-0.57	0.57
Lag	-0.03	0.07	-0.38	0.71
Prime*Position	0.26	0.29	0.87	0.39

e) Interaction Lag*Position
Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.24	0.31	-0.78	0.44
Prime	0.54	0.08	7.19	0.00
Condition	-0.08	0.15	-0.57	0.57
Position	-0.10	0.38	-0.28	0.78
Lag	0.00	0.08	-0.04	0.97
Lag*Position	-0.05	0.08	-0.70	0.49

f) Interaction Prime*Lag
Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.18	0.31	-0.57	0.57
Condition	-0.08	0.15	-0.57	0.57
Position	-0.21	0.35	-0.60	0.55
Prime	0.52	0.17	3.08	0.00
Lag	-0.03	0.08	-0.42	0.67
Prime*Lag	0.01	0.08	0.18	0.86

g) Interaction Prime*Condition

Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Error	z	p-value
Intercept	-0.18	0.31	-0.60	0.55
Lag	-0.03	0.07	-0.38	0.70
Position	-0.21	0.35	-0.60	0.55
Prime	0.52	0.11	4.92	0.00
Condition	-0.10	0.17	-0.63	0.53
Prime*Condition	0.04	0.15	0.26	0.80

h) Interaction Condition(Prime, Position, Lag)

Random Effects

	Standard Deviation
Participants (240)	0.97
Verb (16)	0.67

Fixed Effects

	Estimate	Error	z	p-value
Intercept	-0.44	0.34	-1.30	0.19
Lag	0.12	0.10	1.14	0.25
Prime	0.52	0.11	4.91	0.00
Position	-0.28	0.35	-0.79	0.43
Condition	0.41	0.34	1.20	0.23
Lag*Condition	-0.29	0.15	-2.00	0.05
Prime*Condition	0.04	0.15	0.26	0.80
Position*Condition	0.15	0.15	0.96	0.34

i) Interaction Lag (Condition, Prime, Position)

Random Effects

	Standard Deviation
Participants (240)	0.97
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.52	0.35	-1.47	0.14
Condition	0.50	0.33	1.53	0.13
Position	-0.11	0.38	-0.28	0.78
Prime	0.52	0.17	3.07	0.00
Lag	0.13	0.12	1.17	0.24
Condition*Lag	-0.29	0.15	-2.00	0.05
Position*Lag	-0.05	0.08	-0.68	0.50
Prime*Lag	0.01	0.08	0.19	0.85

j) Interaction Prime*Position*Condition

Random Effects

	Standard Deviation
Participants (240)	0.98
Verb (16)	0.67

Fixed Effects

	Estimate	Error	z	p-value
Intercept	-0.04	0.32	-0.13	0.90
Lag	-0.03	0.07	-0.38	0.70
Condition	-0.26	0.23	-1.14	0.25
Position	-0.50	0.41	-1.22	0.22
Prime	0.31	0.23	1.34	0.18
Condition*Position	0.34	0.33	1.01	0.31
Condition*Prime	0.22	0.33	0.69	0.49
Position*Prime	0.44	0.42	1.06	0.29
Condition*Prime*Position	-0.38	0.59	-0.64	0.52

k) Interaction Lag*Condition(Positions, Prime)

Random Effects

	Standard Deviation
Participants (240)	0.97
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.65	0.37	-1.77	0.08
Position	-0.17	0.41	-0.40	0.69
Prime	0.84	0.24	3.53	0.00
Condition	0.77	0.40	1.93	0.05
Lag	0.22	0.13	1.77	0.08
Position*Condition	0.12	0.34	0.36	0.72
Prime*Condition	-0.66	0.34	-1.95	0.05
Position*Lag	-0.06	0.11	-0.54	0.59
Prime*Lag	-0.16	0.11	-1.50	0.13
Condition*Lag	-0.47	0.18	-2.64	0.01
Position*Condition*Lag	0.01	0.15	0.09	0.93
Prime*Condition*Lag	0.35	0.15	2.31	0.02

l) All possible interactions

Random Effects

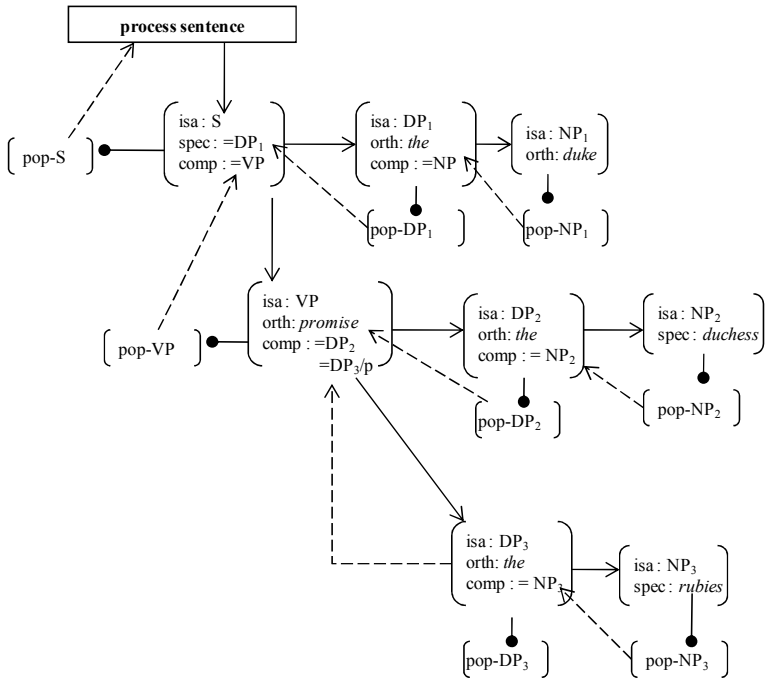
	Standard Deviation
Participants (240)	0.96
Verb (16)	0.67

Fixed Effects

	Estimate	Standard Error	z	p-value
Intercept	-0.87	0.43	-2.00	0.05
Position	0.27	0.62	0.44	0.66
Prime	1.27	0.51	2.49	0.01
Condition	1.02	0.51	2.00	0.05
Lag	0.39	0.16	2.38	0.02
Position*Prime	-0.88	0.92	-0.96	0.34
Position*Condition	-0.39	0.73	-0.53	0.60
Prime*Condition	-1.15	0.72	-1.60	0.11
Position*Lag	-0.39	0.23	-1.67	0.10
Prime*Lag	-0.48	0.23	-2.11	0.03
Condition*Lag	-0.64	0.23	-2.81	0.01
Position*Prime*Condition	1.02	1.30	0.78	0.43
Position*Prime*Lag	0.66	0.41	1.60	0.11
Position*Condition*Lag	0.36	0.33	1.09	0.27
Prime*Condition*Lag	0.69	0.32	2.13	0.03
Position*Prime*Condition*Lag	-0.70	0.58	-1.19	0.23

Appendix 4E: Diagrams of processing and production rules

Processing chain for matrix dative clause

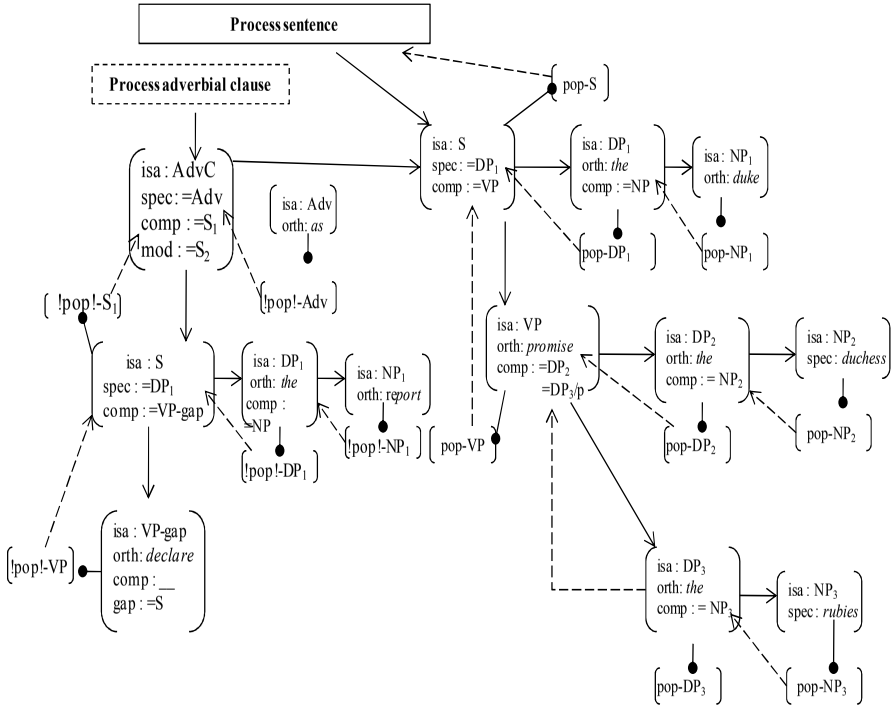


Chain of production rules & unification operations

retrieve S-chunk
push S-chunk
retrieve DP-chunk
 push DP-chunk
  retrieve NP-chunk
  pop NP-chunk
unify NP1 with =NP of DP1
 pop DP-chunk
unify DP1 with =DP of S
 retrieve VP-chunk
 push VP-chunk
  retrieve DP-chunk
  push DP-chunk
   retrieve NP-chunk
   pop NP-chunk
unify NP2 with =NP2 of DP2
  pop DP-chunk
unify DP2 with =DP2 of VP
  retrieve DP-chunk
  push DP-chunk
   retrieve NP-chunk
   pop NP-chunk
unify NP3 with =NP3 of DP3
  pop DP-chunk
unify DP3 with =DP3 of VP
 pop VP-chunk
unify VP with =VP of S
pop S-chunk

SENTENCE: "The duke promised the duchess the rubies."

Processing chain for a sentence with an introductory adverbial clause with a dative verb in matrix clauses



SENTENCE: "As the report declared , the lord promised the duchess the rubies."

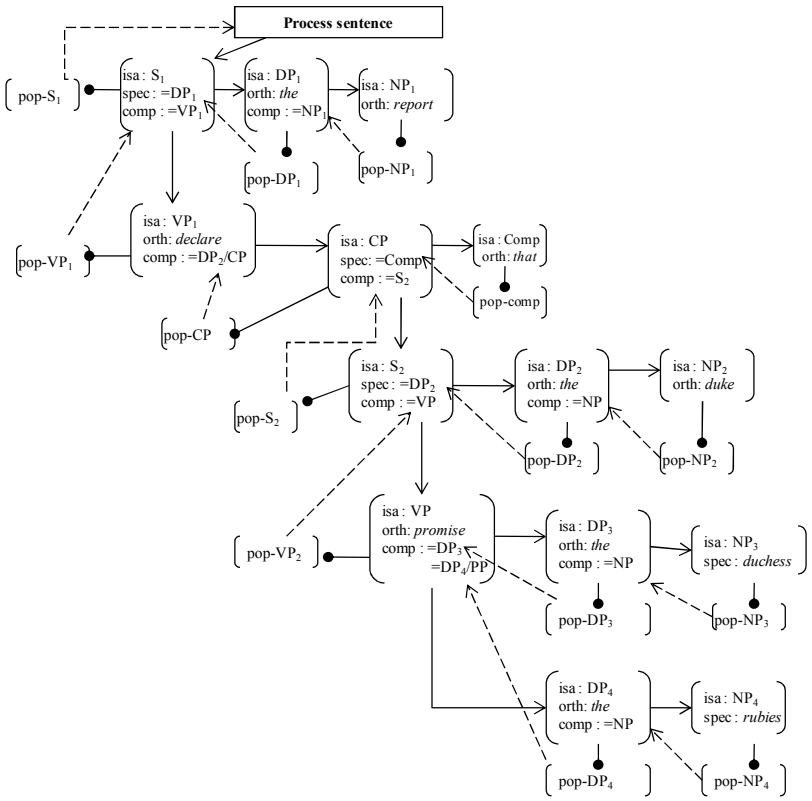
Chain of production rules & unification operations

retrieve S_1-chunk
push S_1-chunk

retrieve AdvC
push AdvC
 retrieve Adv
 !pop!-Adv
unify pop-Adv with =Adv in AdvC
retrieve S_2-chunk
push S_2-chunk
 retrieve DP_1-chunk
 push DP_1-chunk
  retrieve NP_1-chunk
  push NP_1-chunk
  !pop!-NP_1
unify pop-NP with =NP in DP-chunk
  !pop!-DP_1
unify pop-DP with =DP in S_1-chunk
 retrieve VP-gap-chunk
 !pop!-VP
unify pop-VP with S_2-chunk
  !pop!-S_2
unify pop-S_2 with =S in AdvC-chunk
  !pop!-AdvC

retrieve DP_2-chunk
 push DP_2-chunk
  retrieve NP_2-chunk
  pop NP_2-chunk
unify NP_2 with =NP of DP_2
 pop DP_2-chunk
unify DP_2 with =DP of S
 retrieve VP_2-chunk
 push VP_2-chunk
  retrieve DP_3-chunk
  push DP_3-chunk
   retrieve NP_3-chunk
   pop NP_3-chunk
unify NP_3 with =NP of DP_3
  pop DP_3-chunk
unify DP_3 with =DP of VP_2
  retrieve DP_4-chunk
  push DP_4-chunk
   retrieve NP_4-chunk
   pop NP_4-chunk
unify NP_4 with =NP of DP_4
  pop DP_4-chunk
unify DP_4 with =DP of VP
 pop VP_2-chunk
unify VP_2 with =VP of S_1
pop S_1-chunk

Processing chain for a sentence with a dative verb in the internal complement of a verb

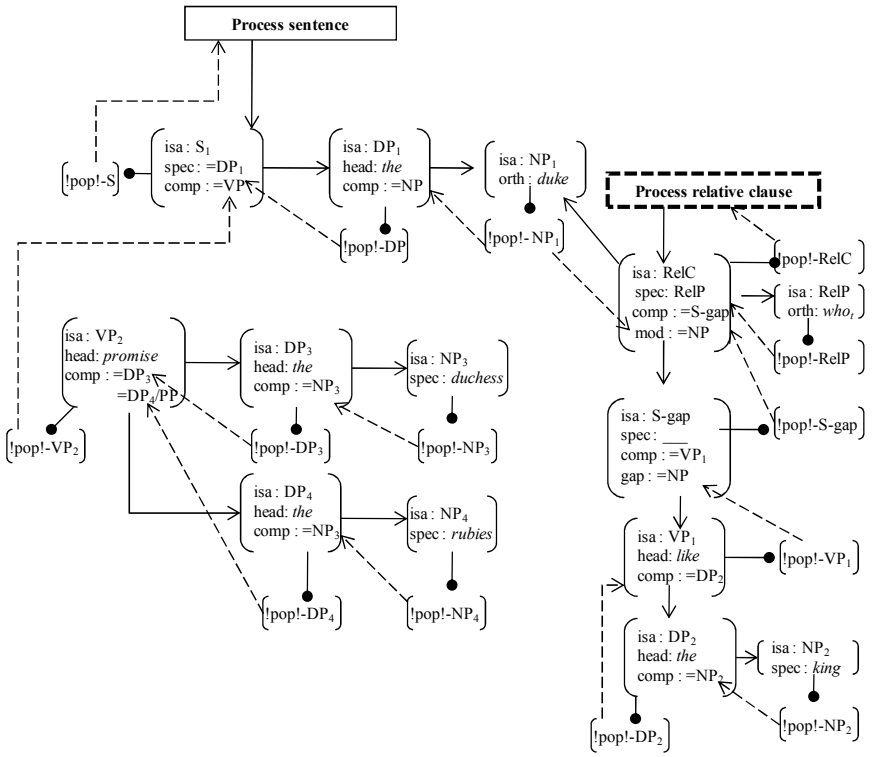


SENTENCE: "The report declared that the lord promised the duchess the rubies."

Chain of production rules & unification operations

retrieve S-chunk
push S-chunk
  retrieve DP_1-chunk
  push DP_1-chunk
    retrieve NP_1-chunk
    pop NP_1
unify NP with =NP in DP-chunk
    pop DP_1
unify DP with =DP in S-chunk
  retrieve VP_1-chunk
  push VP_1-chunk
    retrieve CP-chunk
    push CP-chunk
      retrieve Comp-chunk
      pop Comp-chunk
unify Comp with CP
 retrieve S_2-chunk
 push S_2-chunk
  retrieve DP_1-chunk
    push DP_1-chunk
      retrieve NP_1-chunk
      pop NP_2
unify NP with =NP in DP-chunk
      pop DP_2
unify DP with =DP in S-chunk
  retrieve VP-chunk
  push VP-chunk
    retrieve DP_3-chunk
    push DP_3-chunk
      retrieve NP_3-chunk
      pop NP_3
unify NP with =NP in DP-chunk
      pop DP_3
unify DP with =DP in VP-chunk
    retrieve DP_4-chunk
    push DP_4-chunk
      retrieve NP_4-chunk
      pop NP_4
unify NP with =NP in DP-chunk
      pop DP_4
unify DP with =DP in VP-chunk
    pop VP
unify VP with =VP in S-chunk
    pop S
unify S with CP-chunk
    pop CP
unify CP with VP_1-chunk
    pop VP_1
unify VP with S_1-chunk
    pop S

Processing chain for a with dative verb with subject-modifying relative clause



SENTENCE: "The duke who likes the king promised the duchess the rubies."

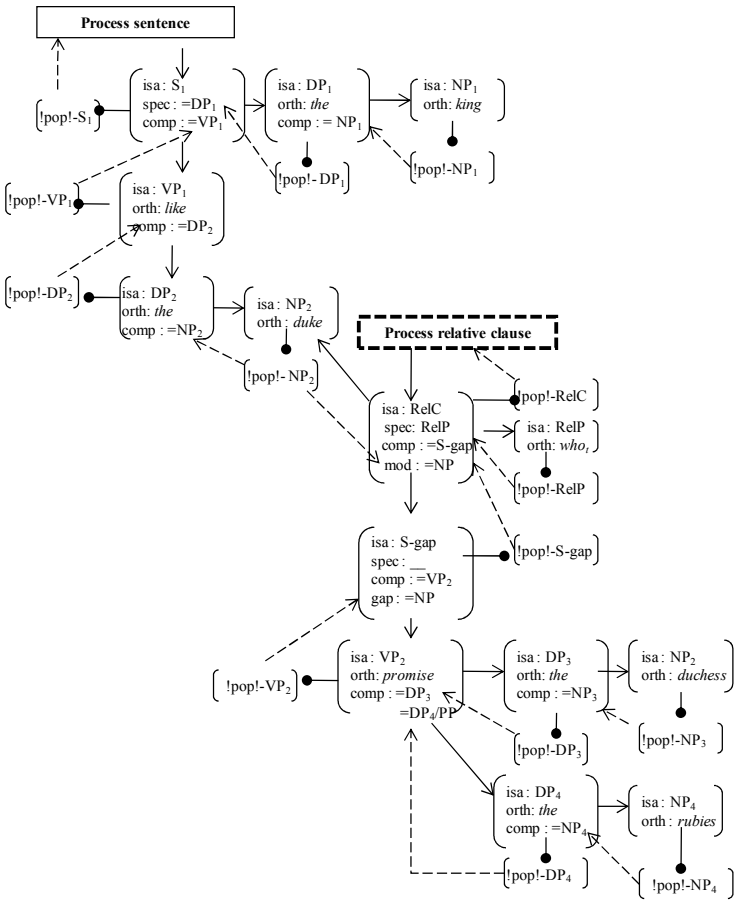
Chain of production rules & unification operations

retrieve S-chunk
push S-chunk
  retrieve DP_1-chunk
  push DP_1-chunk
   retrieve NP_1-chunk
   pop NP_1

retrieve RelC
  retrieve RelP
  pop RelP
unify RelP with =RelP in RelC
  retrieve S-gap-chunk
  push S-gap-chunk
   retrieve VP_2-chunk
   push VP_2-chunk
     retrieve DP_2-chunk
     push DP_2-chunk
       retrieve NP_2-chunk
       pop NP_2
unify NP with =NP in DP-chunk
       pop DP_2
unify DP with =DP in VP-chunk
    pop VP
unify VP with =VP in S-chunk
  pop S
unify S with =S in RelC-chunk
  retrieve NP_1-chunk
  pop NP_1
unify NP with =NP in RelC-chunk
pop RelC
send to LTM

  retrieve NP_1-chunk
  pop NP_1
unify NP_1 with =NP of DP_1
 pop DP-chunk
unify DP_1 with =DP of S
 retrieve VP-chunk
 push VP-chunk
  retrieve DP-chunk
  push DP-chunk
   retrieve NP-chunk
   pop NP-chunk
unify NP_2 with =NP_2 of DP_2
  pop DP-chunk
unify DP_2 with =DP_2 of VP
  retrieve DP-chunk
  push DP-chunk
   retrieve NP-chunk
   pop NP-chunk
unify NP_3 with =NP_3 of DP_3
  pop DP-chunk
unify DP_3 with =DP_3 of VP
 pop VP-chunk
unify VP with =VP of S
pop S-chunk
unify S with control state

Processing chain for a sentence with an object-modifying relative clause with a dative verb



SENTENCE: "The king likes the duke who promised the duchess the rubies."

Chain of production rules & unification operations

retrieve S-chunk
push S-chunk
  retrieve DP_1-chunk
  push DP_1-chunk
    retrieve NP_1-chunk
    pop NP_1
unify NP_1 with =NP of DP_1
  pop DP-chunk
unify DP_1 with =DP of S
  retrieve VP_1-chunk
  push VP_1-chunk
    retrieve DP_2-chunk
    push DP_2-chunk
      retrieve NP_2-chunk
      pop NP_2-chunk

retrieve RelC
  push RelC
    retrieve RelP
    pop RelP
unify RelP with =Rel P of RelC
  retrieve S-gap-chunk
  push S-gap-chunk
retrieve VP_2-chunk
    push VP_2-chunk
      retrieve DP_3-chunk
      push DP_3-chunk
        retrieve NP_3-chunk
        pop NP_3
unify NP_3 with =NP of DP-chunk
        pop DP_3
unify DP with =DP of VP-chunk
      retrieve DP_4-chunk
      push DP_4-chunk
        retrieve NP_4-chunk
        pop NP_4
unify NP with =NP of DP-chunk
        pop DP_4
unify DP with =DP of VP-chunk
      pop VP_2
unify VP with =VP of S-chunk
    pop S-gap-chunk
unify S with RelC-chunk
    retrieve NP_2
    pop NP_2
unify NP with =NP in RelC-chunk
  pop RelC
➔ ***send to LTM***

5 CHAPTER

Conclusion

> Perplexity is the beginning of knowledge. ~ *Khalil Gibran*

We began this dissertation with the observation that linguistic behavior can be affected by two factors: recent use and structural context. The amount of time that has passed since a speaker last encountered a linguistic form along with the structural context in which the form occurred can affect how likely reuse of the form is. In Chapter 1, I proposed RICE:

> **Recent Interaction with Context Effect (RICE)**
> The effect of a recently-encountered linguistic form on subsequent behavior is mediated by the way its structural context was processed.

From RICE came two basic questions: (1) does structural context affect the retrieval of both lexical and structural forms, and (2) what should count as the relevant context for exploring these effects? With respect to question (1), the research presented in Chapters 3 and 4 suggests that structural context affects the retrieval of both lexical forms and structural patterns. With respect to question (2), I claimed that the notion of a unification chain (discussed in detail in Chapter 2) can help account for the sensitivity of lexical and structural priming to structural context.

In what follows, I first review the basis of the RICE hypothesis and the basic findings of the lexical priming study in Chapter 3 and the structural priming studies in Chapter 4. Following

this, I discuss some of the implications and possible limitations of the current studies and suggest future avenues of research. The chapter ends with some speculations about RICE and language processing.

1. What we know about RICE

RICE proposes that the processing of a linguistic form and its structural context affect subsequent behavior with the form. The model of language processing presented in Chapter 2 claims that (i) recently used declarative chunks have higher activation weights and recently used production rules have higher strengths, and that (ii) the structural context influences the way declarative chunks and production rules are retrieved and integrated, thus affecting the way memories for the processing event are represented in long-term memory. This language processing model assumes that lexical knowledge and structure-building knowledge are represented differently in memory. Lexical knowledge maps onto declarative knowledge (modeled here by declarative chunks), and structure-building knowledge maps onto procedural knowledge (modeled here by production rules). These two types of knowledge are often considered distinct (Anderson 2005, Anderson & Lebiere 1998, Bock 1986b inter alia) and may even require the use of different areas of the brain (Ullman 2001; Ullman, Corkin, Coppola, Hickok, Growdon, Koroshetz, & Pinker 1997). The priming studies reported in Chapters 3 and 4 suggest that both the declarative knowledge and the procedural knowledge associated with language processing are sensitive to the larger structural contexts in which their associated prime forms occur.

In Chapter 3, I presented results from a lexical priming study where response times for primed words were slower for primes occurring in the internal complements of nouns (henceforth *noun complement clauses*) as compared to those occurring in matrix clauses, relative clauses, and the internal complements of verbs (henceforth *verb complement clauses*). In Chapter 4, I presented results from two studies that tested the effects of structural context on structural priming at short and long intervals (i.e. with one filler item between the prime and target and with three filler items between them). These results also indicate that priming behavior was sensitive to the different structural contexts with which the primes were associated. Primes occurring in verb complement clauses did not demonstrate priming at longer intervals (lags), unlike those occurring in matrix clauses of sentences with introductory adverbial clauses, matrix clauses of sentences with relative clauses, or in relative clauses. At the same time, priming from matrix clauses of sentences with relative clauses and from relative clauses themselves increased at longer lags. That is, the amount of priming relative to the baseline was significant at the long lag but not at the short lag, though there was always a difference between completions following PD primes and those following DO primes. The results from the structural priming studies indicate that both structural context and recency are relevant for structural priming. Taken together, the results from the lexical and structural priming studies support the RICE hypothesis: structural context mediates the effects of recent processing.

The reason structural context mediates priming behavior is that structural context affects the features of memory traces for processing events. Structural contexts that include argument clauses have more chunks and rules associated with them. For example, in the sentence “The

king knew that the duke promised the duchess the rubies," there is one unification chain containing all the chunks and rules associated with the processing of the sentence, and the prime is associated with this chain. However, in the sentence "The king knew the duke who promised the duchess the rubies," there are two unification chains: one for processing the matrix clause "the king knew the duke" and one for processing the relative clause "who promised the duchess the rubies." The prime is associated with the second chain, which has fewer chunks and rules than the unification chain associated with the verb complement clause sentence above. The more declarative chunks and production rules associated with the processing of the structural context of a prime, the less reliable the priming. After processing a sentence, the processor can access the memory traces formed during that processing event. Features of these traces, such as the number of production rules fired and declarative chunks utilized, affect how the processor evaluates the individual elements within the trace. For example, in Chapter 2, section 3.3.2, I argued that the more declarative chunks associated with the processing of a structural context, the less cognitive resources each form that is being processed receives. This, in turn, affects the total activation of a prime.[1] The more chunks that occur in the same structural context as the prime chunk, the less activation the prime chunk receives and, hence, the less priming. The presence of numerous other chunks leads to interference for locating the prime chunk during subsequent accessing of the processing event.

This 'interference' for chunks is similar to what happens with production rules. The language processing model in Chapter 2 contends that the more rules that are fired during the processing of a given structure, the less likely priming is for a particular rule is. The reason for

[1] This spread of cognitive resources is similar to the Fan Effect (Anderson 1974) discussed in Chapter 2, section 5.2.

this is that with each additional rule application, the cost of using a particular rule increases and the likelihood of successfully achieving the goal decreases. The higher cost and the decrease in possible success lead to lower utility scores for production rules. Consequently, the structural primes in some structural contexts affect subsequent behavior less than structural primes associated other structural contexts.

The processing model presented in Chapter 2 claims that differences between structural contexts are reflected by differences in unification chains, which are formed as a consequence of the goal structures generated during the processing of sentences. The model of processing presented in Chapter 2 treats language processing as a series of coordinated goals that involves the retrieval and manipulation of chunks and rules. A successful unification of two forms constitutes a unification cycle (Chapter 2, section 3.4). When the product of one cycle unifies with the next form in the problem state, the two unification cycles form a chain. This chain continues to grow until such point as the product of a unification cycle can no longer unify with an element in the problem state. At this point, the chain is sent to long-term memory. During subsequent processing, the processor accesses these chains, and priming behavior is affected by the number of elements (chunks or rules) associated with the chain.

In Chapter 2, section 3.4, I argue that these chains reflect the argument/adjunct distinction. Arguments are required by their selectors, and, as such, an argument is associated with the same unification chain as its selector. Adjuncts, on the other hand, are associated with independent chains that are associated with the element the adjuncts modify but that are distinct from the other chains associated with the processing of the sentence.

Previous research in both production and comprehension have found differences between the processing of argument and adjunct clauses, in particular noun/verb complement clauses and relative clauses (Gayraud & Martinie 2008; Gibson 1998, 2000, 2003; Gibson, Desmet, Grodner, Watson & Ko 2005; Grodner & Gibson 2005; Hudgins & Cullinan 1978; Kennison 2002; Shaprio, Oster, Garcia, Massey, & Thompson 1992; Tanenhaus, Spivey-Knowlton, Eberhard, & Sedivy 1995; Trueswell, Tanenhaus, & Garnsey 1994; van Gompel, Pickering, & Traxler 2001; Watson, Breen, & Gibson 2006; Watson & Gibson 2004). My results indicate that the argument/adjunct distinction may also be useful in predicting priming effects.

2. Limitations of lexical and structural priming studies

The lexical and structural priming studies presented in Chapters 3 and 4 offer support for RICE. In what follows, I discuss some of the limitations of those studies and then focus on lingering issues that must be addressed by future work.

2.1 General limitations

In each of the lexical and structural priming experiments presented in this dissertation, the prime occurred in the same linear position, namely in the final clause of a two clause sentence. Furthermore, the primes in these studies were either verbs or VP alternations. In the lexical priming study presented in Chapter 3, the target was the same verb in the same form as its prime. In the structural priming studies discussed in Chapter 4, the primed alternate was for a VP-level alternation (i.e. the dative alternation). And in all the studies, the primes occurred late in the sentence. Future research should manipulate both the linear position of the prime and the

prime's syntactic category.

To begin with, future work should explore the effects of priming from the first clause of two clause sentences or from subject positions of sentences. In the experiments discussed in Chapters 3 and 4, the prime occurred late in the sentence, as in the sentences below.

1a) Prime in a noun complement clause at the end of the sentence
The king knew the fact that the duke promised the duchess the rubies.

b) Prime in matrix/second clause of a sentence with adverbial clause
As the king hoped, the duke promised the duchess the rubies.

c) Prime in object-modifying relative clause
The king called the duke who promised the duchess the rubies.

In (1a) the prime occurs in a noun complement clause at the end of the sentence. In (1b), it occurs in the matrix clause of a sentence with an introductory adverbial clause ("as the king hoped"), and in (1c), the prime occurs in an object-modifying relative clause. Future research should explore the effects of moving these primes to earlier positions in similar sentence types. For example, in (2a), the prime *promise* occurs in a noun complement clause in subject position. In (2b) the prime occurs in the matrix and first clause of a sentence with an adverbial clause. In (2c), the prime occurs in a subject-modifying relative clause.

(2a) Prime in noun complement clause in subject position
The fact that the duke promised the duchess the rubies relieved the king.

b) Prime in matrix and first clause of a sentence with adverbial clause
The duke promised the duchess the rubies as the king hoped.

c) Prime in subject-modifying relative clause
The duke who promised the duchess the rubies called the king.

The model of language processing presented in Chapter 2 contends that the differences we found between primes in the second clauses and predicate positions should be the same as what we would find for primes in the first clauses and subject positions. In other words, if we found a difference between primes in noun complement clauses (e.g. (1a)) and relative clauses (e.g. (1c)) when they occurred in sentence final position, we should also find a difference between them when they occur in early in the sentence (as in (2a) and (2c)). At the same time, the processing model claims that primes occurring in the same structural context in either the first or final clause should demonstrate equal amounts of interference of facilitation in priming from the structural context. If recency and structural context contribute to priming independently, then we might find a slight difference in priming from elements earlier in the sentence due to recency. Once recency has waned and only the long-term effects of priming remain (e.g. after a lag of three items for structural priming), then the contributions of structural context on priming can be seen. At this point, there should be no difference between primes that occur early in the sentence than those that occur late in the sentence if they are associated with the same unification chain. The reason is that cognitive resources are shared among all forms of a unification chain. Thus, a particular form's position within the chain is not relevant, only that it is associated with the chain is relevant. In other words, priming from noun complement clauses late in the sentence (e.g. (1a)) should show the same amount of structural context facilitation (or lack thereof) as from noun complements early in the sentence (e.g. 2a)).

Another factor future experiments should explore is the effects of having a prime in a sentence with an argument clause in which the prime does not occur in the argument clause.

For example, in Chapter 3, I argued that response times for primes such as *promise* in (3a) were slower than response time for primes such as *promise* in (3b) because there were more chunks associated with the unification chain that resulted from the processing of the noun complement clause. In (3a) the forms that were used during the processing of the unification chain associated with the noun complement clause are denoted by the brackets. In (3b) the forms that were used during the processing of the unification chain associated with the relative clause are denoted by the brackets

.

(3a) Verb prime embedded in noun complement clause sentence
[The king knew the fact that the duke promised the duchess the rubies.]

b) Verb prime embedded in object-modifying relative clause
The king called the [duke who promised the duchess the rubies.]

If the number of chunks in associated with a chain is the factor that affects priming behavior, then whether the prime occurs in the noun complement clause, for example, or just in a sentence with a noun complement clause should not matter. To illustrate, consider sentences (4a) and (4b).

(4a) Prime in a sentence with a noun complement clause sentence
[The king knew the fact that the duke promised the duchess the rubies.]

b) Prime in a sentence with an object-modifying relative clause
[The king called the duke] who promised the duchess the rubies.

Here, the prime *king* occurs in a sentence with a noun complement clause (4a) and a sentence with an object-modifying clause (4b). In both cases, the prime is in the matrix clause and early in the sentence. However, in (4a), the unification chain associated with the prime is much longer

(i.e. 13 chunks) than the unification chain associated with the prime in (4b) (i.e. 5 chunks). PRICE claims that there should be less priming from (4a) than from (4b) because of the differences in the number of chunks associated with the primes' unification chains just as there was less priming from (3a) than from (3b) above. Likewise, as the above example suggests, the grammatical category of the prime should not matter. The reason that the grammatical category should be irrelevant is that within the model of language processing presented in Chapter 2 nouns and verbs are both chunks and are both susceptible to the same effects of recency and structural context. However, this assumption should be tested because previous research has found differences between nouns and verbs (e.g. Collina, Garbin & Tabossi 2007; Khader, Scherag, Streb, & Rösler 2003). Likewise, research should test the assumption mentioned earlier that being early the sentence versus late in the sentence is irrelevant. The assumption stems from the fact that all forms associated with a chain receive the same amount of goal-resource facilitation or the same rate of utility regardless of their linear order. However, previous research has found differences between early-mentioned and late-mentioned items (Gernsbacher 1990), so the RICE-based assumption should be tested.

2.2 Future work in lexical priming

The results from the lexical priming study found a sharp difference between primes in noun complements and those in all other structural configurations. However, the model of processing presented in Chapter 2 cannot as such account for the differences between noun and verb complement clauses discussed in Chapter 3. Both the noun and verb complement clauses should lead to greater interference and less priming than priming from adjunct clauses and matrix

clauses because the unification chains associated with sentences with noun and verb complement clauses contain more elements than the unification chains that are generated during the processing of single clause sentences and sentences with an adjunct clause. In Chapter 3, I proposed two possible explanations for the observed differences between noun complement clauses and verb complement clauses. The first possibility was inspired by syntactic work exploring the island effects associated with complex noun phrases. Previous research has found that extraction from noun complement clauses is not grammatical in English whereas extraction from verb complement clauses is (Ross 1967, Haegeman 1991, Lasnik 1999). For example, (5b) is ill-formed whereas (6b) is not.

(5) Extraction from a noun complement clause
a) Mark knew the fact that Yaron likes crepes.
b) *What did Mark know the fact that Yaron likes __?

(6) Extraction from a verb complement clause
a) Mark knew that Yaron likes crepes.
b) What did Mark know that Yaron likes__?

It may be possible to connect the prohibition against extraction from noun complement clauses to weak priming from these positions. If so, then we would expect to find less priming from other configurations that lead to island effects, such as subjects (7).

(7) Extraction from a subject
a) The art of making crepes fascinated Yaron.
b) *What did the art of making ___ fascinate Yaron?

It is worth exploring these other structures to see if priming may be sensitive to the same factors that lead to island effects.

The second possible source of the difference is that the effects of each additional chunk associated with a chain is not linear but rather exponential. Rather than each chunk contributing equally to the processing slowdown, each chunk compounds the effect. Recall that the prime sentences with noun complement clauses had two words ("the fact") more than the sentences with verb complement clauses. In Chapter 3, section 5, I argue that these two additional words led to weaker priming because of the exponential nature of the fan effect. However, to clarify, according to this view, it is not the mere occurrence of more words in a sentence that makes the difference but rather the occurrence of additional words that are associated with same unification chain as the prime. The addition of adjuncts (e.g. adverbs and adjectives) should not lead to an exponential increase in response times. For example, consider (8) and (9) below.

(8) Example of a noun complement clause
Fatima knew the fact that Yousuf **fed** the yogurt to the baby.
Word count: 12

(9) Example of a verb complement clause
Fatima knew that Yousuf **fed** the yogurt to the baby.
Word count: 10

According to the exponential function explanation, the occurrence of two additional words ("the fact") in the noun complement clause causes enough additional interference to lead to significantly slower reaction times for the prime (bolded) in the noun complement clause sentence (8) than in the verb complement clause sentence (9). This perspective on the observed priming differences between verb and noun complement clauses assumes that additional chunks associated with a unification chain can inhibit priming. If this is true, then we should observe a difference between arguments and adjuncts in these structural contexts. Additional adjuncts

should not differentiate between noun complement clauses and verb complement clauses. For instance, adding additional adjunct chains such as those associated with the adjectives in (10) should not affect overall response times. However, the extended unification chain that is a result of processing the structure associated with "Wendy said that" in (11) should lead to slower response times.

(10) Adding new chains (i.e. adjunct chains)
Fatima knew that Yousuf **fed** the sour plain yogurt to the happy baby.
Word count: 12

(11) Adding to the same unification chain
Wendy said that Fatima knew that Yousuf **fed** the yogurt to the baby.
Word count: 13

The reason that additional adjectives should not affect response times is that they are not associated with the same unification chain as the prime. The additional "Wendy said that" in (12) is associated with the same unification chain as the prime and as such should affect response times.

2.3 Future work in structural priming

The most striking finding from the structural priming studies in Chapter 4 was the interaction between structural context and time. Priming from relative clauses appeared to improve over time whereas priming from verb complement clauses deteriorated over time. In Chapter 4, section 5, I speculated that rather than improving at longer lags, primes in relative clauses were initially inhibited. This inhibition stemmed from the additional semantic processing associated with relative clauses. The processor must saturate the gap's feature and, thus, the empty '___' associated with the 'spec' features of the S-gap-chunks with information from the

filler element (i.e. the NP-chunk that unified with the open =NP value of the RelC-chunk). This may lead to slightly weaker priming at shorter lags due to the additional processing. Currently, there is no way to tell whether priming improved for relative clause primes or whether it returned to normal after the semantic processing concluded and the memory traces had consolidated. Future research needs to explore the time course of priming from relative clauses more closely.

The second major finding from the structural priming study was the decrease in priming from verb complement clauses over time. I contend that the reason for less priming in the long-lag experiment relative to the short-lag experiment stemmed from lower utility scores for rules associated with relatively long unification chains. If this explanation is correct, then we should find improved priming behavior as the processor becomes more familiar with the prime sentence pattern.[1] Previous research has found that participants are sensitive to the frequency of particular structural primes in training blocks (Kaschak 2007; Kaschak & Borreggine 2008; Kaschak, Loney, & Borreggine 2006) and have used this to argue that participants are learning about the overall probability of different alternates. Participants may also demonstrate learning effects as they are exposed to the pairing of verb complement clauses and forms of the dative alternation. The more experience the processor has with a rule pattern (e.g. retrieve X, push X, retrieve Y, push Y…), the higher the pattern's utility is, regardless of the length or complexity of the rule pattern. As the pairing of the structural prime and the structural context become more frequent,

[1] I am not suggesting that verb complement clauses are infrequent. In fact, they are quite frequent (Roland, Dick, & Elman 2007). Low frequency structures may have low utility scores because the processor does not have a lot of experience with them and cannot, therefore, reliably predict their probability of success or cost. However, because verb complement clauses are frequent, we cannot assume that the clause type itself led to low utility scores and, hence, weak priming. What I am contending is that the ability of the processor to evaluate a particular rule (e.g. one associated with building a DO or PD) is more difficult in long unification chains, such as those produced by the processing of verb complement clauses.

the utility score should increase. This leads one to expect that priming from sentences with argument clauses can stabilize after sufficient exposure to the association of the particular structural prime with the particular structural context. Priming from verb complement clauses after a short lag (1 filler item) was possible because the strength of the rule was still high, even if the utility was low. But as the strength decreased due to longer lags (i.e. 3 filler items), the low utility score led to weaker priming. As the processor becomes more familiar with the pattern of rule firings associated with the long unification chains formed by the processing of verb complement clause sentences, the utility scores for this type of sentence should increase. The increase in utility scores should lead to more reliable priming regardless of the lag (e.g. even after 3 filler items). Thus, the disappearance of priming from verb complement clauses after a lag of 3 filler items can be corrected with enough exposure, leading to priming from verb complements at long lags comparable to priming from relative clauses at long lags.

2.4 Other avenues for exploration with RICE and PRICE

A final avenue to explore is naturally occurring speech. Previous research has found that speakers tend to repeat the linguistic forms they encounter in their natural environment (e.g. Bock *et al.* 2007, Levelt & Kelter 1982, Jaeger & Snider 2008, Szmrecsanyi 2005). This work has found that priming weakens as the distance between the prime and the target increases. However, none of this work to date has considered the structural contexts of the prime or the target. The studies presented in Chapter 3 and Chapter 4 suggest that the structural context of the prime is an additional factor that should be explored in corpus work.

3. Final comments on and implications of RICE

RICE claims that both recency and structural context affect the representation of linguistic forms in long-term memory and that differences in these representations lead to different patterns of linguistic behavior. The studies in Chapters 3 and 4 support RICE. The structural context in which linguistic forms occur mediates the faciliatory effects of recent processing.

At times, we study language and priming by trying to examine a specific word or structural pattern independent of its larger structural context as though the form is easily disentangled from its context. What this dissertation suggests is that priming is sensitive to structural context. Priming is affected by both when the prime was processed (as measured by recency) and how the prime was processed (as determined by its structural context). These two forces blend together and create intricate patterns of behavior. Only by exploring the larger, arabesque patterns of language and the way the pieces interact and influence one another can we understand the dynamic and interconnected weaving of linguistic forms and behavior.

WORKS CITED

Ahrens, Kathleen. 2003. Verbal integration: the interaction of participant roles and sentential argument structure. Journal of Psycholinguistic Research 32(5). 487-516.

Allen, Mark and William Badecker. 1999. Stem homograph inhibition and stem allomorphy: Representing and processing inflected forms in a multilevel lexical system. Journal of Memory and Language 41. 105–123.

Allen, Mark and William Badecker. 2002. Inflectional regularity: Probing the nature of lexical representation in a cross-modal priming task. Journal of Memory and Language 46. 705–722.

Allen, Richard J. and Alan D. Baddeley. 2009. Working memory and sentence recall. Interactions between Short-term and Long-term Memory in the Verbal Domain, ed. by Annable Thorn and Mike Page, 63-85. New York, NY: Psychology Press.

Almor, Amit. 1999. Noun-phrase anaphora and focus: The informational load hypothesis. Psychological Review 106(4). 748-765.

Almor, Amit and Peter D. Eimas. 2008. Focus and noun phrase anaphors in spoken language comprehension. Language and Cognitive Processes 23(2). 201-225.

Altman, Gerry T. M. 2001. The language machine: Psycholinguistics in review. British Journal of Psychology 92. 129-170.

Altman, Gerry T. M., Alan Garnham, and Yvett Dennis. 1992. Avoiding the garden path: Eye movements in context. Journal of Memory and Language 31. 685-712.

Anderson, John R. 1974. Retrieval of propositional information from long-term memory. Cognitive Psychology 6(4). 451-474.

Anderson, John. R. 1993. Rules of the Mind. Hillsdale, NJ: Erlbaum.

Anderson, John. R. 1995. Learning and Memory. New York, NY: Wiley.

Anderson, John R. 2005. Human Symbol Manipulation within an Integrated Cognitive Architecture. Cognitive Science: A Multidisciplinary Journal 29(3). 313-341.

Anderson, John R, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. 2004. An integrated theory of the mind. Psychological Review 111(4). 1036-1060.

Anderson John R., Raluca Budiu, and Lynne M. Reder. 2001. A theory of sentence memory as

part of a general theory of memory. Journal of Memory and Language 45. 337-367.

Anderson, John R. and Scott Douglass. 2001. Tower of Hanoi: Evidence for the cost of goal retrieval. Journal of Experimental Psychology: Learning, Memory, and Cognition 27(6). 1331-1346.

Anderson, John R. and Christian Lebiere. 1998. The Atomic Components of Thought: Mahwah, NJ: Erlbaum.

Anderson, John. R. and Peter L. Pirolli. 1984. Spread of activation. Journal of Experimental Psychology: Learning, Memory, & Cognition 10. 791-799.

Anderson, John R. and Lynn M. Reder. 1999. The fan effect: New results and new theories. Journal of Experimental Psychology: General 128(2). 186-197.

Anderson, John R., Lynne M. Reder, and Christian Lebiere. 1996. Working memory: Activation limitations on retrieval. Cognitive Psychology 30. 221–256.

Anderson, John R., Christian Lebiere, Marsha Lovett, and Lynne Reder. 1998 ACT-R: A higher-level account of processing capacity. Behavioral and Brain Sciences 21(6). 831-832.

Anderson, John. R. and Christian D. Schuun. 2000. Implications of the ACT-R learning theory: No magic bullets. Advances in Instructional Psychology: Educational Design and Cognitive Science (Volume 5), ed. by Robert Glaser, 1-34. Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson Michael C. and James H. Neely. 1996. Interference and inhibition in memory retrieval. Handbook of Perception and Memory (Volume 10), ed. by Elizabeth L. Bjork and Robert A. Bjork, 237-313. San Diego, CA: Academic Press.

Arnold, Jennifer E., Tomas Wasow, Anthony Losongco, and Ryan Ginstrom. 2000. Heaviness vs. newness: The effects of structural complexity and discourse status on constituent ordering. Language 76(1). 28-55.

Atkinson, Richard and Richard Shiffrin. 1971. The control of short-term memory. Scientific American 225. 82-90.

Baayen, R.Harold, D.J. Davidson, and D.M. Bates. 2008. Mixed-effects modeling with crossed random effects for subjects and items. Journal of Memory and Language 59. 390-412.

Baayen, R. Harald, Ton Dijkstra, and Robert Schreuder. 1997. Singulars and Plurals in Dutch: Evidence for a Parallel Dual-Route Model. Journal of Memory and Language 37(1). 94- 117.

Baayen, R.Harald, Richard Piepenbrock, Leon and Gulikers. 1995. The CELEX Lexical Database (Release 2) [CD-ROM]. Philadelphia: Linguistics Data Consortium (Distributor).

Baddeley, Alan D. 1986. Working Memory. Oxford, United Kingdom: Oxford University Press.

Baddeley, Alan D. 1992. Is working memory working? The fifteenth Bartlett lecture. Quarterly Journal of Educational Psychology 44A(1). 1-31.

Baddeley, Alan D. 2000. The episodic buffer: A new component of working memory? Trends in Cognitive Sciences 4(11). 417-423.

Baddeley, Alan D, and Graham J. Hitch. 1974. Working memory. The Psychology of Learning and Motivation: Advances in Research and Theory (Volume 8), ed. by GordonH. Bower, 47-90. New York, NY: Academic Press.

Baddeley, Alan D, Graham J. Hitch, and Richard J. Allen. 2009. Working memory and binding in sentence recall. Journal of Memory and Language 61. 438-456.

Baddeley, Alan D. and Robert H. Logie. 1999. Working memory: The multiple-component model. Models of Working Memory: Mechanisms of Active Maintenance and Executive Control, ed. by Akira Miyake and Priti Shah, 28-61. New York, NY: Cambridge University Press.

Becker, Tilman. 1994. Patterns in metarules. In Proceedings of the Third International Workshop on Tree Adjoining Grammars. 9-11.

Becker, Susan, Morris Moscovitch, Marlene Behrmann and Steve Joordens. 1997. Long-term semantic priming: A computational account and empirical evidence. Journal of Experimental Psychology: Learning, Memory, and Cognition 23. 1059–1082.

Bentin, Sholmo and Laura Feldman. 1990. The contribution of morphologic and semantic relatedness to repetition priming at short and long lags: Evidence from Hebrew. Quarterly Journal of Experimental Psychology: Human Experimental Psychology 42(A). 693-711.

Bentin, Sholmo and Morris Moseovitch. 1988. The time course of repetition effects for words and unfamiliar faces. Journal of Experimental Psychology: General 117. 148-160.

Birch, Stacy, Jason E. Albrecht, and Jerome L. Myers. 2000. Syntactic focusing structures influence discourse processing. Discourse Processes 30. 285-304.

Birch, Stacy and Susan M. Garnsey. 1995. The effect of focus on memory for words in sentences. Journal of Memory and Language 34. 232-267.

Bjork, Robert A. and William B. Whitten. 1974. Recency-sensitive retrieval processes in long-term free recall. Cognitive Psychology 6. 173-189.

Bock, Kay. 1986a. Meaning, sound, and syntax: Lexical priming in sentence production. Journal of Experimental Psychology: Learning, Memory, Learning, and Cognition 12(4). 557-586.

Bock, Kay. 1986b. Syntactic persistence in language production. Cognitive Psychology 18(3). 355-387.

Bock, Kay and J. Cooper Cutting. 1992. Regulating mental energy: Performance units in language production. Journal of Memory and Language 31. 99–127.

Bock, Kay, Gary Dell, Franklin Chang and Kristine H. Onishi. 2007. Persistent structural priming from language comprehension to language production. Cognition 104(3). 437-458.

Bock, Kay and Zennzi M. Griffin. 2000. The persistence of structural priming: Transient activation or implicit learning? Journal of Experimental Psychology: General 129(2). 177-192.

Bock, Kay and Anthony S. Kroch. 1989. The isolability of syntactic processing. Linguistic Structure in Language Processing, ed. by Greg N. Carlson and Michael K. Tanenhaus, 157-196. Dordrecht, Netherlands: Kluwer.

Bock, Kay and Willem J. M. Levelt. 1994. Language production: Grammatical encoding. Handbook of Psycholinguistics, ed. by Morton Gernsbacher, 945–984. San Diego, CA: Academic Press.

Bock, Kay and Loebell, Helga. 1990. Framing sentences. Cognition 35. 1-39.

Bock, Kay and Carol A. Miller. 1991. Broken agreement. Cognitive Psychology 23. 45–93.

Bock, Kay, Janet Nicol, and Cooper Cutting. 1999. The ties that bind: Creating number agreement in speech. Journal of Memory and Language 40. 330–346.

Boland, Julie E. 2005. Cognitive mechanisms and syntactic theory: Arguments against adjuncts in the lexicon. Twenty-First Century Psycholinguistics: Four Cornerstones, ed. by Anne Cutler, 23-42. Mahwah, NJ: Lawrence Erlbaum Associates.

Boland, Julie E., Michael K. Tanenhaus, and Susan M. Garnsey. 1990. Evidence for the immediate use of verb control information in sentence processing. Journal of Memory and Language 29. 413-423.

Boland, Julie, Michael K. Tanenhaus, Susan M. Garnsey, and Greg N. Carlson. 1995. Verb argument structure in parsing and interpretation: Evidence from wh-questions. Journal of Memory and Language 34. 774–806.

Boyland, J. T. and John R. Anderson. 1997. Comprehension and production as avenues of syntactic priming. Paper presented at the 19th Annual Conference of the Cognitive Science Society.

Bowers, Jeffrey S. 2000. In defense of abstractionist theories of repetition priming and word identification. Psychonomic Bulletin & Review 7(1). 83-99.

Branigan, Holly E, Martin J. Pickering, and Sandie A. Cleland. 1999. Syntactic priming in written production: Evidence for rapid decay. Psychonomic Bulletin and Review 6. 635-640.

Branigan, Holly P., Martin J. Pickering, Simon P. Liversedge, Andrew J. Stewart and Thomas P. Urbach. 1995. Syntactic priming: Investigating the mental representation of language. Journal of Psycholinguistic Research 24(6). 489-506.

Branigan, Holly E., Martin J. Pickering, and Janet F. McLean. 2005. Priming prepositional-phrase attachment during comprehension. Journal of Experimental Psychology: Learning, Memory, and Cognition 31(3). 468-481.

Branigan, Holly P., Martin J. Pickering, Janet F. McLean, and Andrew J. Stewart. 2006. The role of local and global syntactic structure in language production: Evidence from syntactic priming. Language and Cognitive Processes 21(7). 974 – 1010.

Brennan, Susan and Herbert H. Clark. 1996. Conceptual Pacts and Lexical Choice in Conversation. Journal of Experimental Psychology: Learning, Memory, and Cognition 22(6). 1482–1493.

Bresnan, Joan. 2007. Is syntactic knowledge probabilistic? Experiments with the English dative alternation. Roots: Linguistics in Search of Its Evidential Base. Series: Studies in Generative Grammar, ed. by Sam Featherston and Wolfgang Sternefeld, 77-96. Berlin, Germany: Mouton de Gruyter.

Bresnan, Joan, Anna Cueni, Tatiana Nikitina, and Harald Baayen. 2007. Predicting the dative alternation. Cognitive Foundations of Interpretation, ed. by Gerlof Boume, Irene Krämer, and Joost Zwarts, 69-94. Amsterdam, Netherlands: Royal Netherlands Academy of Science.

Bresnan, Joan and Tatiana Nikitina. 2009. The gradience of the dative alternation. Reality Exploration and Discovery: Pattern Interaction in Language and Life, ed. by Linda Uyechi and Lian Hee Wee, 161-184. Stanford, CA: CSLI Publications.

Britt, Mary A. 1994. The interaction of referential ambiguity and argument structure in the parsing of prepositional phrases. Journal of Memory and Language 33(2). 251-283.

Callahan, Sarah M, Lewis P. Shapiro, and Tracy Love. 2008. The activation of verbs in sentences involving verb phrase anaphors. Presented at the 16th Annual CUNY Conference on Sentence Processing. Chapel Hill, NC.

Caplan, David. 1972. Clause boundaries and recognition latencies for words in sentences. Perception and Psychophysics 12. 73-76.

Carpenter, Patricia A. and Marcel A. Just. 1988. The role of working memory in language comprehension. In D. Klahr & K. Kotovksy (Eds.), Complex information processing: The impact of Herbert A. Simon. Hillsdale, N.J.: Erlbaum.

Chambers, Craig G., Michael Tanenhaus, and James S. Magnuson. 2004. Actions and affordances in syntactic ambiguity resolution. Journal of Experimental Psychology: Learning, Memory, and Cognition 30. 687–96.

Chang, Franklin. 2008. Implicit learning as a mechanism of language change. Theoretical Linguistics 34(2). 115-122.

Chang, Franklin, Gary S. Dell, and Kay J. Bock. 2006. Becoming syntactic. Psychological Review 113. 234–272.

Chang, Franklin, Gary S. Dell, Kay J. Bock, and Zenzi M. Griffin. 2000. Structural priming as implicit learning: A comparison of models of sentence production. Journal of Psycholinguistic Research 29. 217-229.

Chomsky, Noam. 1981. Government and Binding. Dordrecht, Netherlands: Foris.

Clahsen, Harald and Sam Featherston. 1999. Antecedent priming at trace positions: Evidence from German scrambling. Journal of Psycholinguistic Research 28(4). 531-571.

Cleland, Sandie A. and Michael J. Pickering. 2003. The use of lexical and syntactic information in language production: Evidence from the priming of noun-phrase structure. Journal of Memory and Language 49. 214-230.

Clifton, Charles. Jr, Sheila M. Kennison, and Jason E. Albrecht. 1997. Reading the words *her*, *his*, and *him*. Journal of Memory and Language 36. 276-292.

Clifton, Charles Jr., Shari Speer and Steven P. Abney. 1991. Parsing arguments: Phrase structure

and argument structure as determinants of initial parsing decisions. Journal of Memory and Language 30(2). 251-271.

Collina, S., G. Garbin, and P. Tabossi. 2007. The role of argument structure in the processing of nouns and verbs: An f-MRI study. Brain and Language 103. 8-249

Collins, Andrew M. and Elizabeth F. Loftus. 1975. A spreading activation theory of semantic processing. Psychological Review 82. 407-428.

Connine, Cynthia M., Dawn G. Blasko, and Jian Wang. 1994. Vertical similarity in spoken word recognition: Multiple lexical activation, individual differences, and the role of sentence context. Perception and Psychophysics 56(6). 624-636.

Cowan, Nelson. 1999. An embedded-processes model of working memory: A reconsideration of mental storage capacity. Behavioral and Brain Sciences 24. 87-114.

Cowan, Nelson. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. Brain and Behavioral Sciences 24. 87–114.

Cowan, Nelson and Zhijian Chen. 2009. How chunks form in long-term memory and affect-short term memory limits. Interactions between Short-term and Long-term Memory in the Verbal Working Domain, ed. by Annabel Thorn and Mike Page, 86-107. New York, NY: Psychology Press.

Cowles H. Wind and A. Garnham. 1995. Antecedent focus and conceptual distance effects in category noun-phrase anaphora. Language and Cognitive Processes 20(6). 725-750.

Cowles, H. Wind, Matthew Walenski and Robert Kluender. 2007. Linguistic and cognitive prominence in anaphor resolution: topic, contrastive stress and pronouns. Topoi 26. 3-18.

Craik, Fergus I. M. and Endel Tulving. 1975. Depth of processing and the retention of words in episodic memory. Journal of Experimental Psychology: General 104. 268-294.

Davelaar, Eddy J., Yonatan Goshen-Gottstein, Henk J. Haarmann, Amir Ashkenazi, and Marius Usher. 2005. The demise of short-term memory revisited: Empirical and computational investigations of recency effects. Psychological Review 112(1). 3-42.

de Goede, Dieuwke. 2007. Verbs in spoken sentence processing: Unraveling the activation pattern of the matrix verb. Ph.D. Dissertation. Groningen, Netherlands: Groningen Dissertations in Linguistics 63.

Dell, Gary S. 1986. A spreading activation theory of retrieval in sentence production.

Psychological Review 93. 281-321.

Dell, Gary S., Franklin Chang, and Zenzi M. Griffin. 1999. Connectionist models of language production: Lexical access and grammatical encoding. Cognitive Science 23(4). 517-542.

Dell, Gary S. and Jean K. Gordon. 2003. Neighbors in the lexicon: Friends or foes? Phonetics and Phonology in Language Comprehension and Production: Differences and Similarities, ed. by Niels O. Schiller and Antje S. Meyer, 9-38. Berlin, Germany: Mouton de Gruyter.

Dell, Gary S. and Padraig F. O'Seaghdha. 1991. Mediated and convergent lexical priming in language production: A comment on Levelt *et al.* Psychological Review 98(4). 604-614.

Demberg, Vera and Frank Keller. 2008a. A psycholinguistically motivated version of TAG. In 45 Proceedings of the 9th International Workshop on Tree Adjoining Grammars and Related Formalisms. Tübingen, Germany.

Demberg, Vera and Frank Keller. 2008b. Data from eye-tracking corpora as evidence for theories of syntactic processing complexity. Cognition 109(2). 193-210.

Demberg, Vera and Frank Keller. 2009. A computational model of prediction in human parsing: Unifying locality and surprisal effects. In Proceedings of the 29th meeting of the Cognitive Science Society (CogSci-09), Amsterdam, Netherlands.

Demestre, Josep and Jose E. Garcia-Albea. 2004. The on-line resolution of the sentence complement/relative clause ambiguity: Evidence from Spanish. Experimental Psychology 51(1). 59-71.

Desmet, Timothy and Mieke Declercq. 2006. Cross-linguistic priming of syntactic hierarchical configuration information. Journal of Memory and Language 54. 610-632.

Deese, James and Roger A. Kaufman. 1957. Serial effects in recall of unorganized and sequentially organized verbal material. Experimental Psychology 54. 180-187.

Doyle, Gabriel and Roger Levy. 2008. Environment prototypicality in syntactic alternation. Proceedings of the 34th Annual Meeting of the Berkeley Linguistics Society (BLS). Berkeley, CA.

Dubey, Amit, Frank Keller and Patrick Sturt. 2008. A probabilistic corpus-based model of syntactic parallelism. Cognition 109. 326-344.

Elman, Jeffery L. 1990. Finding structure in time. Cognitive Science 14. 179-211.

Elman, Jeffery L. 1991. Distributed representations, simple recurrent networks, and grammatical structure. Machine Learning 7. 195-225.

Ericsson, K. Anders and Walter Kintsch. 1995. Long-term working memory. Psychological Review 102. 211-245.

Estival, Dominique. 1985. Syntactic priming of the passive in English. Text 5. 7-21.

Ferrand, Ludovic and Boris New. 2003. Semantic and associative priming in the mental Lexicon. Found on: boris.new.googlepages.com/Semantic-final-2003.pdf

Ferreira, Fernanda. 1991. Effects of length and syntactic complexity on initiation times for prepared utterances. Journal of Memory and Language 30. 210-233.

Ferreira, Fernanda. 2000. Syntax in language production: An approach using tree-adjoining grammars. Aspects of Language Production, ed. by Linda Wheeldon, 291–330. Cambridge, MA: MIT Press.

Ferreira, Fernanda and John M. Henderson. 1990. Use of verb information in syntactic parsing: evidence from eye movements and word-by-word self-paced reading. Journal of Experimental Psychology: Learning, Memory, and Cognition 16. 555–68.

Ferreira, Fernanda and Benjamin Swets. 2002. How incremental is language production? Evidence from the production of utterances requiring the computation of arithmetic sums. Journal of Memory and Language 46. 58-84.

Ferreira, Victor. 1996. Is it better to give than to donate? Syntactic flexibility in language production. Journal of Memory and Language 35. 724-755.

Ferreira, Victor and Kay Bock. 2006. The functions of structural priming. Language and Cognitive Processes 21(7-8). 1011-1029.

Ferreira, Victor S., Kay Bock, Michael P. Wilson and Neal J. Cohen. 2008. Memory for syntax despite amnesia. Psychological Science 19. 940-946.

Ferreira, Victor. S and L. Robert Slevc. 2007. Grammatical encoding. The Oxford Handbook of Psycholinguistics, ed. by M. Gareth Gaskell, 453–469. New York, NY: Oxford University Press.

Foraker, Stephani and Brian McElree. 2007. The role of prominence in pronoun resolution: Active versus passive representations. Journal of Memory and Language 56. 357-383.

Ford, Marilyn. 1982. Sentence planning units: implications for the speaker's representation of

meaningful relations underlying sentences. The Mental Representation of Grammatical Relations, ed. by Joan Bresnan, 797-827. Cambridge, MA: MIT Press.

Forbus, Kenneth D., Dedre Gentner, and Keith Law. 1995. MAC/FAC: A model of similarity-based retrieval. Cognitive Science: A Multidiscplinary Journal 19(2). 141-205.

Fox Tree, Jean E. and Paul J. A. Meijer. 1999. Building syntactic structure in speaking. Journal of Psycholinguistic Research 28. 71-92.

Franck, Julie, Gabriella Vigliocco and Janet Nicol. 2002. Subject-verb agreement in French and English: The role of syntactic hierarchy. Language & Cognitive Processes 17. 371-404.

Frank, Robert. 1992. Syntactic locality and tree adjoining grammar: Grammatical, acquisition, and processing perspectives. Ph.D. Dissertation. Philadelphia, PA: University of Pennsylvania.

Frank, Robert. 2004. Restricting grammatical complexity. Cognitive Science 28. 669–697.

Frank, Robert and William Badecker. 2001. Modeling Syntactic Encoding with Tree Adjoining Grammar: How grammar constrains production and production constrains grammar. CUNY Sentence Processing Conference Presentation. Philadelphia, PA.

Frazier, Lyn, Lori Taft, Tom Roeper, Charles Clifton, and Kate Ehrlich. 1984. Parallel structure: A source of facilitation in sentence comprehension. Memory & Cognition 12. 421-430.

Friederici, Angela D., Herbert Schriefers, and Ulman Lindenberger. 1998. Differential age effects on semantic and syntactic priming. International Journal of Behavioral Development 22(4). 813-845.

Friederici, Angela D., Karsten Steinhauer, and Stefan Frisch. 1999. Lexical integration: Sequential effects of syntactic and semantic information. Memory & Cognition 27(3). 438-453.

Forster, K.I. 1981. Priming and the effects of sentence and lexical contexts on naming time: Evidence for autonomous lexical processing. The Quarterly Journal of Experimental Psychology 33(4). 465 – 495.

Garrett, Merrill F. 1982. Production of speech: Observations from normal and pathological language use. Normality and Pathology in Cognitive Functions, ed. by A.W. Ellis, 19–76. London, United Kingdom: Academic Press.

Garrett, Merrill. F. 1988. Processes in language production. In F. J. Newmeyer (Ed.), Linguistics: The Cambridge survey, Vol. 3. Language: Psychological and biological aspects. 69–96. Cambridge, United Kingdom: Cambridge University Press.

Garrod, Simon and Anthony Anderson. 1987. Saying what you mean in dialogue: A study in conceptual and semantic co-ordination. Cognition 27(2). 181-218.

Gayraud, Frédérique and Bruno Martinie. 2008. Does structural complexity necessarily imply processing difficulty? Journal of Psycholinguistic Research 37. 21-31.

Gernsbacher, Morton A. 1989. Mechanisms that improve referential access. Cognition 32. 99-156.

Gernsbacher, Morton A. 1990. Language Comprehension as Structure Building. Hillsdale, NJ: Lawrence Erlbaum Associates Inc.

Gibson, Edward. 1998. Linguistic complexity: locality of syntactic dependencies. Cognition 68. 1–76.

Gibson, Edward. 2000. The dependency locality theory: A distance-based theory of linguistic complexity. Image, language, brain, ed. by Yasushi Miyashita, Alec Marantz and Wayne O'Neil, 95–126. Cambridge, MA: MIT Press.

Gibson, Edward. 2003. Linguistic complexity in sentence comprehension. Encyclopedia of Cognitive Science, ed. by Lynn Nadel, 1137-1141. New York, NY: MacMillian.

Gibson, Edward, Timothy Desmet, Daniel Grodner, Duane Watson, and Kara Ko. (2005). Reading relative clauses in English. Cognitive Linguistics 16. 313-354.

Glosser, Guila & Rhonda B. Friedman. 1991. Lexical but not semantic priming in Alzheimer's Disease. Psychology and Aging, 6(4). 522-527.

Green, Georgia. 1974. Semantics and Syntactic Regularity. Bloomington, IN: Indiana University Press.

Gries, Stefan. 2005. Syntactic Priming: A Corpus-based Approach. Journal of Psycholinguistic Research 34(4). 365–399.

Gries, Stefan Th and Anatol Stefanowitsch. 2004. Extending collostructional analysis: A corpus-based perspective on 'alternations.' International Journal of Corpus Linguistics 9(1). 97-129.

Griffin, Zenzi and Justin Weinstein-Tull. 2003. Conceptual structure modulates structural priming in the production of complex sentences. Journal of Memory and Language 49. 537-55.

Grimshaw, Jane. 1979. Complement selection and the lexicon. Linguistic Inquiry 10(2). 279-

326.

Grodner, Daniel J., and Edward Gibson. 2005. Consequences of the serial nature of linguistic input for sentential complexity. Cognitive science 29(2). 261-291.

Gropen, Jess, Steven Pinker, Michelle Hollander, Richard Goldberg, and Ronald Wilson. 1989. The learnability and acquisition of the dative alternation in English. Language 65(2). 203-257

Halford, Graeme S., William H. Wilson, and Steven Phillips. 1998. Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology. Behavioral and Brain Sciences 21. 803–865.

Hartsuiker, Rob and Casper Westenberg. 2000. Word order priming in written and spoken sentence production. Cognition 75. B27–B39.

Hartsuiker, Robert, Sarah Bernolet, Sofie Schoonbaert, Sarah Speybroeck, and Dieter Vanderelst. 2008. Syntactic priming persists while the lexical boost decays: Evidence from written and spoken dialogue. Journal of Memory and Language 58(2). 214-238.

Hartsuiker, Rob and Casper Westenberg. 2000. Word order priming in written and spoken sentence production. Cognition 75. B27-B39.

Hartsuiker, Rob, Martin J. Pickering, and Eline Veltkamp. 2004. Is syntax separate or shared between languages?: Cross-linguistic syntactic priming in Spanish-English bilinguals. Psychological Science 15(6). 409-414.

Haegeman, Liliane M.V. 1991. Introduction to Government and Binding Theory. Oxford, United Kingdom: Basil Blackwell

Hitch, Graham J. and Robert H Logie. 1996. Working Memory: A Special of the Quarterly Journal of Experimental Psychology (Section A). London, United Kingdom: Psychology Press.

Hoey, Michael. 2005. Lexical Priming: A New Theory of Words and Language. New York, NY: Routledge.

Hofmeister, Philip. 2008. Representational Complexity and Memory Retrieval in Language Comprehension. Ph.D. Dissertation. Stanford, CA: Stanford University.

Howard, Marc W. and Michael J. Kahana. 1999. Contextual variability and serial position effects in free recall. Journal of Experimental Psychology: Learning, Memory, and Cognition 25(4). 923-941.

Hudgins, Jo Carol and Walter L. Cullinan. 1978. Effects of sentence structure on sentence elicited imitation responses. Journal of Speech and Hearing Research 21. 809-819.

Hutchinson, Keith. A. 2003. Is semantic priming due to association strength or featural overlap? A micro-analytic review. Psychonomic Bulletin & Review 10(4). 785-813.

Huttenlocher, Janellen, Marina Vasilyeva, and Priya Shimpi. 2004. Syntactic priming in young children. Journal of Memory and Language 50. 182-195.

Jäger, Gerhard and Anette Rosenbach. 2008a. Priming and unidirectional language change. Theoretical Linguistics 34(2). 85-113.

Jäger, Gerhard and Anette Rosenbach. 2008b. Priming as a testing ground for historical linguists? – A reply to Chang, Eckardt, and Traugott. Theoretical Linguistics 34(2). 85-113.

Jaeger, T. Florian and Neal Snider. 2008. Implicit learning and syntactic persistence: Surprisal and cumulativity. Proceedings of the Cognitive Science Society Conference. Washington, DC.

Jarvell, R. J. 1971. Syntactic processing of connected speech. Journal of Verbal Learning and Verbal Behavior 10. 409-416.

Joordens, Steve and Suzanna Becker. 1997. The long and short of semantic priming effects in lexical decision. Journal of Experimental Psychology: Learning, Memory, and Cognition 23(5). 1083-1105.

Joshi, Aravind. K. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? Natural Language Parsing: Psychological, Computational and Theoretical Perspectives, ed. by David Dowty, Lauri Karttunen, and Arnold M. Zwicky, 206–250. New York, NY: Cambridge University Press.

Joshi, Aravind. K., Leon S. Levy and Masako Takahashi. 1975. Tree adjunct grammars. Journal of Computer and System Sciences 10. 136–163.

Jurafsky, Daniel and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (Second Edition). Upper Saddle River, NJ: Prentice-Hall.

Just, Marcel A. and Patricia A. Carpenter. 1992. A capacity theory of comprehension: Individual differences in working memory. Psychological Review 99. 122–149.

Just, Marcel A., Patricia A. Carpenter, and Timothy A. Keller. 1996. The capacity theory of comprehension: New frontiers of evidence and arguments. Psychological Review 103. 773–780.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. The Mental Representation of Grammatical Relations, ed. by Joan Bresnan, 173-281. Cambridge, MA: MIT Press.

Kaschak, Michael P. 2007. Long-term structural priming affects subsequent patterns of language production. Memory and Cognition 35(5). 925-937

Kaschak, Michael P. and Kristin L. Borreggine. 2008. Is long-term structural priming affected by patterns of experience with individual verbs? Journal of Memory and Language 58. 862-878.

Kaschak, Michael P., Renrick A. Loney, and Kristin L. Borreggine. 2006. Recent experience affects the strength of structural priming. Cognition 99. B73-82

Keller, Frank. 2009. The interaction of syntactic theory and computational psycholinguistics. Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics, 43–46. Athens, Greece.

Kempen, Gerard and Edward Hoenkamp. 1987. An incremental procedural grammar for sentence formulation. Cognitive Science: A Multidisciplinary Journal 11(2). 201-258.

Kennison, Shelia M. 2002. Comprehending noun phrase arguments and adjuncts. Journal of Psycholinguistic Research 31(1). 65-81.

Khader, Patrick , Andre Scherag, Judith Streb and Frank Rösler. 2003. Differences between noun and verb processing in a minimal phrase context: a semantic priming study using event-related brain potentials. Cognitive Brain Research 17: 293–313.

Kim, Albert E., Bangalore Srinivas and John C. Trueswell. 2002. A computational model of the grammatical aspects of word recognition as supertagging. The Lexical Basis of Sentence Processing: Formal, Computation, and Experimental Issues, ed. by Paola Merlo and Suzanne Stevenson, 109-135. Amsterdam, Netherlands: John Benjamins Publishing.

King, Jonathan and Just, Marcel A. Just. 1991. Individual differences in syntactic processing: The role of working memory. Journal of Memory and Language 30. 580–602.

Kintsch, Walter. 1974. The Representation of Meaning in Memory. Hillsdale, NJ: Erlbaum.

Kintsch, Walter. 1988. The use of knowledge in discourse processing: A construction-integration model. Psychological Review 95. 163-182.

Kintsch, Walter. 1998. Comprehension: A paradigm for cognition. New York, NY: Cambridge

University Press.

Kleinhow, Jennifer and Anne Smith. 2000. Influences of length and syntactic complexity on the speech motor stability of the fluent speech of adults who stutter. Journal of Speech, Language, and Hearing Research 43. 548-559.

Krivokapic, Jelena. 2007. Prosodic planning: Effects of phrasal length and complexity on pause duration. Journal of Phonetics 35. 162–179.

Kromann, Matthias T. 2004. Optimality parsing and local cost functions in discontinuous grammar. Electronic Notes in Theoretical Computer Science 53. 163-179.

Lasnik, Howard. 1999. On the locality of movement: Formalist syntax position paper. Functionalism and Formalism in Linguistics (Volume 1): General papers (SLCS 41), ed. by n Michael Darnell, Edith Moravscik, Michael Noonan, Friedrick Newmeyer, and Kathleen Wheatley, 33-54. Amsterdam, Netherlands: John Benjamins Publishing.

Lebiere, Christian. 1998. The Dynamics of Cognition: An ACT-R Model of Cognitive Arithmetic. Ph.D. Dissertation. CMU Computer Science Dept Technical Report CMU-CS-98-186. Pittsburgh, PA: Carnegie Mellon.

Lebiere, Christian and John R. Anderson. 1998. Cognitive arithmetic. The Atomic Components of Thought, ed. by John R. Anderson and Christian Lebiere, 297-342. Hillsdale, NJ: Erlbaum.

Ledoux, Kerry, Matthew J. Traxler, and Tamara Y. Swaab. 2007. Syntactic priming in comprehension: Evidence from event-related potentials. Psychological Science 18(2). 135-43.

Levelt, Willem. 1989. Speaking: From Intention to Articulation. Cambridge, MA: MIT Press.

Levelt, Willem and Stephanie Kelter. 1982. Surface form and memory in question answering. Cognitive Psychology 14. 78-106.

Levelt, Willem, Ardi Roelofs, and Antje S. Meyers. 1999. A theory of lexical access in speech production. Behavioral Brain Science 22(1). 1-38.

Levelt, Willem J. M., Herbert Schriefers, Dirk Vorberg, Antje S. Meyer, Thomas Pechmann, and Jaap Havinga. 1991. The time course of lexical access in speech production: A study of picture naming. Psychological Review 98(1). 122-142.

Levin, Beth. 1993. English Verb Classes and Alternations. Chicago, IL: University of Chicago Press.

Lewis, Clayton H. and John R. Anderson. 1976. Interference with real world knowledge. Cognitive Psychology 8. 311-335.

Lewis, Richard L. 1996. Interference in short-term memory: The magical number two (or three) in sentence processing. Journal of Psycholinguistic Research 25. 93–115.

Lewis, Richard L. and Shravan Vasishth. 2005. An activation-based model of sentence processing as skilled memory retrieval. Cognitive Science 29. 375-419.

Lewis, Richard L., Shravan Vasishth, and Julie A. Van Dyke. 2006. Computational principles of working memory in sentence comprehension. Trends in Cognitive Science 10. 447-454.

Lightfoot, David. 1991. How to Set Parameters: Arguments from Language Change. Cambridge, MA: MIT Press/Bradford Books.

Loebell, Helga and Kay Bock. 2003. Structural priming across languages. Linguistics 41. 791-824.

Love, Tracy and David A. Swinney. 1996. Coreference processing and levels of analysis in object relative constructions: Demonstration of antecedent reactivation with the cross-modal priming paradigm. Journal of Psycholinguistic Research 25. 5-24.

Lucas, Margery. 2000. Semantic priming without association: A meta-analytic review. Psychonomic Bulletin and Review 7. 618-630.

Luckatela, Georgije, Milan Savic, Zoran Urosevic, and M. T. Turvey. 1997. Phonological ambiguity impairs identity priming in naming and lexical decision. Journal of Memory and Language 36. 360-381

Luka, Barbara J. and Lawrence W. Barsalou. 2005. Structural facilitation: Mere exposure effects for grammatical acceptability as evidence for syntactic priming in comprehension. Journal of Memory and Language 52. 436-459.

MacDonald, Maryellen. C., Marcel A. Just, and Patricia A. Carpenter. 1992. Working memory constraints on the processing of syntactic ambiguity. Cognitive Psychology 24. 56–98.

Manning, Christopher. 2003. Probabilistic syntax. Probabilistic Linguistics, ed. by Rens Bod, Jennifer Hay, and Stefanie Jannedy, 298-341. Cambrige, MA: MIT Press.

Marinis, Theodore. 2003. Psycholinguistic techniques in second language acquisition research. Second Language Research 19(2). 144–161.

Matessa, Michael. 2001. Simulating adaptive communication. Ph.D. Dissertation. Pittsburgh, PA: Carnegie Mellon University.

Matessa, Michael and Anderson, J. R. 2000. Modeling Focused Learning in Role Assignment. Language and Cognitive Processes 15(3). 263-292.

McClelland, James L., and David E. Rumelhart. 1986. A distributed model of human learning and memory. Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Volume 2), ed. by James L. McClelland, David E. Rumelhart, and the PDP Research Group, 170-215. Cambridge, MA: MIT Press.

McElree, Brian and Teresa Griffith. 1995. Syntactic and thematic processing in sentence comprehension: Evidence for a temporal dissociation. Journal of Experimental Psychology: Learning, Memory, and Cognition, 21(1). 134-157.

McKoon, Gail and Roger Ratcliff. 1992. Spreading activation versus compound cue accounts of priming: Mediated priming revisited. Journal of Experimental Psychology: Learning, Memory, and Cognition 18(6). 1155-1172.

McKoon, Gail and Roger Ratcliff. 1998. Memory-based language processing: Psycholinguistic Research in the 1990s. Annual Review Psychology 49: 25-42.

McKoon, Gail, Roger Ratcliff, and Gregory Ward. 1994. Testing theories of language processing: An empirical investigation of on-line lexical decision task. Journal of Experimental Psychology: Learning, Memory, and Cognition 20(5). 1219-1228.

McKoon, Gail, Roger Ratcliff, Gregory Ward and Richard Sproat. 1993. Syntactic prominence effects on discourse processes. Journal of Memory and Language 32. 593-607.

McNamara, Timothy. 1992. Priming and constraints it places on theories of memory and retrieval. Psychological Review 99. 650-662.

McNamara, Timothy. 2005. Semantic Priming: Perspectives from Memory and Word Recognition. New York, NY: Psychology Press/Taylor & Francis.

Meyer, David E., Roger W. Schvaneveldt, and Margret G. Ruddy. 1974. Functions of graphemic and phonemic codes in visual word recognition. Memory and Cognition 2(2). 309-321.

Miller, George A. 1956. The magical number seven plus or minus two: Some limits on our capacity for processing information. Psychological Review 63. 81–97.

Miyake, Akira and Priya Shah (Eds.). 1999. Models of Working Memory: Mechanisms of Active

Maintenance and Executive Control. New York, NY: Cambridge University Press.

Morris, Robin K. and Jocelyn R. Folk. 1998. Focus as a contextual priming mechanism in reading. Memory Cognition 26(6). 1313-1322.

Morton, John. 1969. The interaction of information in word recognition. Psychological Review 76. 165–178.

Murdock, Bennet. B., Jr. 1962. The serial position effect in free recall. Journal of Experimental Psychology 64. 482-488.

Nakano, Yoko, Claudia Felser, and Harald Clahsen. 2002. Antecedent Priming at Trace Positions in Japanese Long-Distance Scrambling. Journal of Psycholinguistic Research 31(5). 531-571.

Nicol, Janet. 1993. Reconsidering reactivation. Cognitive models of speech processing: The second Sperlonga Meeting, ed. by Gerry Altmann and Richard Shillcock, 321-347. Hove, United Kingdom: Erlbaum.

Nicol, Janet, Janet D. Fodor, and David Swinney. 1994. Using cross-modal lexical decision tasks to investigate sentence processing. Journal of Experimental Psychology: Learning, Memory, and Cognition 20. 1229 - 1238.

Nicol, Janet and David Swinney. 1989. The role of structure and co-reference assignment during sentence comprehension. Journal of Psycholinguistic Research 18. 5-19.

Oehrle, R.T. 1976. The grammar of the English dative alternation. PhD Dissertation, Cambridge, MA: MIT Department of Linguistics and Philosophy.

Ogawa, Kenji, Toshio Inui, and Masato Ohba. 2008. Syntactic processing of complex sentences in left lateral premotor cortex. NeuroReport 19(8). 811-815.

Pearl, Lisa. 2005. The input to syntactic acquisition: Solutions from language change modeling. Proceedings of the Second Workshop on Psychocomputational Models of Human Language Acquisition, 1-9. Ann Arvor, MA.

Pearl, Lisa & Weinberg, Amy. 2007. Input filtering in syntactic acquisition: Answers from language change modeling. Language Learning and Development 3(1). 43-72.

Pearlmutter, Neal J., Susan M. Garnsey, and Kay Bock. 1999. Agreement processes in sentence comprehension. Journal of Memory and Language 41. 427-456.

Perea, Manalo and Eva Rosa. 2000. Repetition and form priming interact with neighborhood density at a short stimulus-onset asynchrony. Psychonomic Bulletin and Review 7. 668-677.

Perea, Manalo and Eva Rosa. 2002. The effects of associative and semantic priming in the lexical decision task. Psychological Research 66. 180-194.

Pickering, Martin and Holly Branigan. 1998. The representation of verbs: Evidence from syntactic priming in language production. Journal of Memory and Language 39. 633-651.

Pickering, Martin and Holly Branigan. 1999. Syntactic priming in language production. Trends in Cognitive Science 3. 136-141.

Pickering, Martin, Holly Branigan, and Janet F. McLean, J. 2002. Constituent structure is formulated in one stage. Journal of Memory and Language 46. 586-605.

Pickering, Martin and Victor Ferreira. 2008. Structural priming: A critical review. Psychological Bulletin 134(3). 427-459.

Pinker, Steven. 1989. Learnability and Cognition: The Acquisition of Argument Structure. Cambridge, MA: MIT Press.

Pintzuk, Susan. 1999. Phrase Structures in Competition: Variation and Change in Old English Word Order. New York, NY: Garland.

Pintzuk, Susan and Ann Taylor. 2006. The loss of OV order in the history of English. Blackwell Handbook of the History of English, ed. by Ans van Kemenade and Bettelou Los, 249-278. Oxford, United Kingdom: Blackwell Publishing.

Plaut, David. C. and Laura M. Gonnerman. 2000. Are non-semantic morphological effects incompatible with a distributed connectionist approach to lexical processing? Language and Cognitive Processes 15. 445–485.

Potter, Mary C. and Linda Lombardi. 1998. Syntactic priming in immediate recall of sentences. Journal of Memory and Language 38(3). 265-282.

Rapp, Brenda and Matt Goldrick. 2000. Discreteness and interactivity in spoken word production. Psychological Review 107. 460-499

Rapp, Brenda and Matt Goldrick.2004. Feedback by any other name is still interactivity: A reply to Roelofs's comment on Rapp & Goldrick 2000. Psychological Review 111. 573-578

Ratcliff, Roger, William Hockley, and Gail McKoon. 1985. Components of activation:

Repetition and priming effects in lexical decision and recognition. Journal of Experimental Psychology: General, 114. 435-450.

Ratcliff, Roger and Gail McKoon. 1994. Retrieving information from memory: spreading-activation theories versus compound-cue. Psychological Review 101(1). 177-184.

Reitter, David. 2008. Context effects in language production: Models of syntactic priming in dialogue corpora. Ph.D. Dissertation (unpublished). Edinburgh, United Kingdom: School of Informatics, University of Edinburgh.

Rips, Lance, Edward Shoben and Edward Smith. 1973. Semantic distance and the verification of semantic relations. Journal of Verbal Learning and Verbal Behavior 12. 1-20.

Roelofs, Ardi. 1992. A spreading activation theory of lemma retrieval in speaking. Cognition 42. 107–142.

Roelofs, Ardi. 1993. Testing a non-decompositional theory of lemma retrieval in speaking: Retrieval of verbs. Cognition 47. 59–87.

Roland, Douglas, Frederic Dick, and Jeffery L. Elman. 2007. Frequency of basic English grammatical structures: A corpus analysis. Journal of Memory and Language 57. 348-379.

Ross, John R. 1967. Constraints on variables in syntax. Ph.D. Dissertation. Cambridge, MA: MIT.

Sag, Ivan A. To appear. English Filler-Gap constructions. *Language*.

Sag, Ivan A., Thomas Wasow, and Emily Bender. 2003. Syntactic Theory: A Formal Introduction: second edition. Stanford: CSLI Publications.

Sachs, Jacqueline. 1967. Recognition memory for syntactic and semantic aspects of connected discourse. Perception and Psychophysics 2. 437-42.

Saffran, Eleandor M. and Nadine Martin. 1997. Effects of structural priming on sentence production in aphasics. Language and Cognitive Processes 12(5). 877 – 882.

Scarborough, Don L., Charles Cortese, and Hollis S. Scarborough. 1977. Frequency and repetition effects in lexical memory. Journal of Experimental Psychology: Human Perception and Performance 3. 1-17.

Scheepers, Christoph. 2003. Syntactic priming of relative clause attachments: persistence of structural configuration in sentence production. Cognition 89(3). 179-205.

Seidenberg, Michael S. and James L. McClelland. 1989. A distributed, developmental model of word recognition and naming. Psychological Review 96(4). 523-568.

Shapiro, Lewis P., Elizabeth Oster, Rachel Garcia, Andrea Massey, and Cynthia Thompson. 1999. On-line comprehension of wh-questions in discourse. Presentation of CUNY Human Sentence Processing Conference. New York, NY.

Shieber, Stuart M. 1986. An introduction to Unification-Based Approaches to Grammar, Volume 4 of CSLI Lecture Notes Series. Stanford, CA: Center for the Study of Language and Information.

Sloman, Steven A., C.A. Gordon Hayman, Nobou Ohta, Janine Law and Endel Tulving. 1988. Forgetting in primed fragment completion. Journal of Experimental Psychology: Learning, Memory, and Cognition 14. 223-239.

Smith, Mark and Linda Wheeldon. 1999. High level processing scope in spoken sentence production. Cognition 73. 205-246.

Snider, Neal. 2008. Evidence for a unified theory of structural and lexical priming. Paper presented at the 21st CUNY Sentence Processing Conference. Chapel Hill, NC.

Spivey, Michael. 2007. The Continuity of Mind. New York, NY: Oxford University Press.

Spivey, Michael J., Michael K. Tanenhaus, Kathleen M. Eberhard, and Julie C. Sedivy. 2002. Eye movements and spoken language comprehension: Effects of visual context on syntactic ambiguity resolution. Cognitive Psychology 45. 447-481.

Stockwell, Robert P. and Donka Minkova 1991. Subordination and word order change in the history of English. Historical English Syntax, ed. by Dieter Kastovsky, 367-408. Berlin, Germany: Walter de Gruyter.

Stowe, Laurie A., Cees A. J. Broere, Anne M. J. Paans, Albertus A. Wijers, Gijsbertus Mulder, Wim Vaalburg, and Frans Zwarts. 1998. Localizing components of a complex task: sentence processing and working memory. NeuroReport 9. 2995–2999.

Sturt, Patrick. 2003. The time-course of the application of binding constraints in reference resolution. Journal of Memory and Language 48(3). 542–562.

Sturt, Patrick and Vincenzo Lombardo. 2005. Processing coordinated structures: Incrementality and connectedness. Cognitive Science 29(2). 291–305.

Swinney, David A. 1979. Lexical access during sentence comprehension (re)consideration of context effects. Journal of Verbal Learning Behavior 18. 645-659.

Szmrecsanyi, Benedikt M. 2005. Language users as creatures of habit: a corpus-based analysis of persistence in spoken English. Corpus Linguistics and Linguistic Theory 1. 113–149.

Tannen, Deborah. 1987. Repetition in conversation: Toward a poetics of talk. Language, 63.574-605.

Tanenhaus, Michael K., Michael J. Spivey- Knowlton, Kathleen M. Eberhard, and Julie C. Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. Science 268. 1632–1634.

Tenpenny, Patricia L. 1995. Abstractionist versus episodic theories of repetition priming and word identification. Psychonomic Bulletin and Review 2(3). 339–363.

Thothathiri, Malathi and Jesse Snedeker. 2008. Give and take: Syntactic priming during spoken language comprehension. Cognition 108. 51-68.

Trueswell, John C., Michael K. Tanenhaus, and Susan M. Garnsey. 1994. Semantic influences in parsing: Use of thematic role information in syntactic ambiguity resolution. Journal of Memory and Language 33. 285–318.

Tutunjian, Damon and Julie E. Boland. 2008. Do We Need a Distinction between Arguments and Adjuncts? Evidence from Psycholinguistic Studies of Comprehension. Language and Linguistics Compass 2(4). 631–646.

Ullman, Michael T. 2001. The declarative/procedural model of lexicon and grammar. Journal of Psycholinguistic Research 30. 37-69.

Ullman, Michael T. 2004. Contributions of neural memory circuits to language: The declarative/procedural model. Cognition 92(1-2). 231-270.

Ullman, Michael. T., Susan Corkin, Marie Coppola, Gregory Hickok, John H. Growdon, Walter J. Koroshetz, and Stephen Pinker. 1997. A neural dissociation within language: Evidence that the mental dictionary is part of declarative memory, and that grammatical rules are processed by the procedural system. Journal of Cognitive Neuroscience 9. 266-276.

van Berkum, Jos J. A., Colin M. Brown, and Peter Hagoort. 1999. Early referential context effects in sentence processing: Evidence from event-related brain potentials. Journal of Memory and Language 41. 147-182.

Van Dyke, Julie A., & Richard L. Lewis. 2003. Distinguishing effects of structure and decay on

attachment and repair: A cue-based parsing account of recovery from misanalyzed ambiguities. Journal of Memory and Language, 49. 285–316.

Van Dyke, Julie A. and Brian McElree. 2006. Retrieval interference in sentence comprehension. Journal of Memory and Language 55. 157-166.

van Gompel, Roger P., Martin J. Pickering and Mathew J. Traxler. 2001. Unrestricted race: A new model of syntactic ambiguity resolution. Reading as a Perceptual Process, ed. by Alan Kennedy, Ralph Radach, Dieter Heller, and Joël Pynte, 621-648. Oxford, United Kingdom: Elsevier.

Veríssimo, João and Harald Clahsen. 2009. Morphological priming by itself: A study of Portuguese conjugations. Cognition 112(1). 187-194.

Wang, Suhong, Xuan Dong, Yanling Ren and Yilin Yang. 2009. The development of semantic priming effect in childhood: an event-related potential study. NeuroReport 20(6). 574-578.

Wasow, Tom. 2002. Postverbal Behavior. Stanford: CSLI Publications.

Watson, Duane and Edward Gibson. 2004. The relationship between intonational phrasing and syntactic structure in language production. Language and Cognitive Processes 19. 13-755.

Watson, Duane, Maria Breen, and Edward Gibson. 2006. The role of syntactic obligatoriness in the production of intonational boundaries. Journal of Experimental Psychology: Learning, Memory and Cognition 32. 1045-1056.

Weiner, E. Judith and William Labov. 1983. Constraints on the agentless passive. Journal of Linguistics, 19. 29-58.

Wester, Fernke, Dieuwke de Goede, Roelien Bastiaanse, Lewis Shapiro, and David Swinney 2004. Verb activation patterns in on-line sentence processing of Dutch matrix clauses. Brain and Language 9(1). 120-121.

Zervakis, Jennifer and David Rubin. 2002. Production and Recognition Bias of Stylistic sentences Using a Story Reading Task. Journal of Psycholinguistic Research 31. 107-130.