

UNIVERSITY OF CALIFORNIA

Los Angeles

Generating Descriptions of Motion from Cognitive Representations

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Linguistics

by

Benjamin Keil

2010

UMI Number: 3424163

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

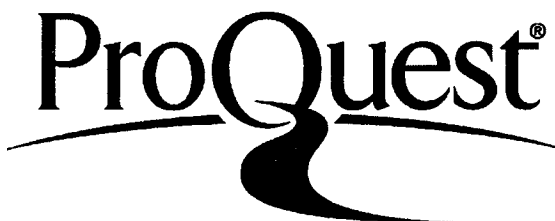
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3424163

Copyright 2010 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

© Copyright by
Benjamin Keil
2010

The dissertation of Benjamin Keil is approved.

Herbert B Enderton

Herbert Enderton

Ed Stabler

Edward Stabler

Edward Keenan

Edward Keenan

Marcus Kracht

Marcus Kracht, Committee Chair

University of California, Los Angeles

2010

This dissertation is dedicated to the memory of Andrea Scamihorn. She got so many wonderful things rolling during her stay here. May those of us who knew her add to their momentum.

TABLE OF CONTENTS

1	Introduction	1
1.1	Goal	1
1.1.1	Why Motion is a Suitable Focus	3
1.1.2	Place of this Dissertation within Talmy’s Program	3
1.1.3	Place of this Dissertation within Levelt’s “Blueprint for the Speaker”	4
1.1.4	This dissertation with respect to “definitions”	5
1.2	Other Approaches to Language Generation	10
1.2.1	Systemic Functional Linguistics	10
1.2.2	Meaning-Text Theory	15
1.2.3	Pratt-Hartmann	18
1.2.4	Generation from Proof Nets	25
1.3	Dissertation Overview	29
2	Cognitive Representations	31
2.1	Overview of Cognitive Representations	31
2.1.1	Motion + Co-Event	32
2.1.2	Motion + Path	33
2.1.3	Motion + Figure	34
2.1.4	Implications for Sentence Generation	36
2.2	More about Motion	37

2.2.1	A MOVE	38
2.2.2	Other Mid-Level Motion Morphemes	41
2.3	Conclusions	42
3	Graphs, grammars, and parsing	43
3.1	Overview	43
3.2	Rendering Cognitive Representations as Graphs	44
3.3	From String Grammars to Graph Grammars	47
3.4	Flowgraphs	52
3.4.1	Example Flowgraph Derivation	56
3.5	Converting from Graphs to Flowgraphs	58
3.5.1	Motivation	58
3.5.2	Process	60
3.6	Parsing with Context Free Flowgraph Grammars	64
3.6.1	Designing a Flowgraph Grammar for Cognitive Representations	64
3.6.2	Patches	66
3.6.3	Lutz's Algorithm	70
3.6.4	Implementing Lutz's Algorithm	73
3.6.5	Equivalence of Patches	73
3.6.6	Extension of Partial Patches by Complete Patches	74
3.7	An Example Parse	76
3.8	Conclusions	84

4	From Graph Parses to Linguistic Expressions	85
4.1	Overview	85
4.2	Definitions	88
4.2.1	Trees	88
4.2.2	Terms	94
4.2.3	From terms to trees and back	95
4.3	Parse Terms	98
4.3.1	String Derivations and Parse Trees	98
4.3.2	Parse Terms for Graphs	101
4.4	Tree Transductions	104
4.4.1	Transducing to Syntactic Structures	106
4.4.2	Transduction to Derivation Trees	112
4.5	Generative power of Macro Tree Transducers	112
4.6	Conclusions	115
5	Conclusion	117
	Appendix	120
A	A (slightly) Larger Graph Grammar	120
A.1	Introduction	120
A.2	(Vector and) Conform	121
A.3	Productions	122
A.3.1	Prepositions($\boxed{\text{Path}} \Rightarrow$)	123

A.3.2 Path verbs ($\boxed{S} \Rightarrow$)	124
References	124

LIST OF FIGURES

1.1	A blueprint for the speaker. Simplified from Levelt, 1989, p. 9 . . .	6
1.2	A judgment in the logic of Merenciano and Morrill (1997).	28
3.1	Example flowgraph	53
3.2	A sketch of Lutz's (1996, p. 367) parsing algorithm, simplified for only bottom-up parsing	72
4.1	Transduction from a parse term to a traditional syntactic analysis	108
4.2	Transduction from tree yielding <i>aaabbb</i> to tree yielding <i>aaabbbccc</i> .	114
4.3	Hierarchy of languages generated by iterated <i>MTT</i> and <i>EDT0L</i> transductions	115

LIST OF TABLES

1.1	Pratt-Hartmann's normal form conjuncts Pratt-Hartmann, 2003, p. 38	20
3.1	Talmy's list of universal vectors (Talmy, 2000b, p.53f)	46
3.2	Complete patches by input	79
3.3	Complete patches by output	80
A.1	A few more Conforms	123

ACKNOWLEDGMENTS

Marcus Kracht introduced me to computational semantics and guided my research throughout both my master's thesis and my dissertation. His views on language and meaning have shaped my own in innumerable ways. It has been an honor to be his student. Ed Keenan taught me almost everything I know about mathematical linguistics. On several occasions his pragmatic approach to solving problems has allowed me to progress in my research when I otherwise would have stalled. I will be forever grateful to him for that. From Ed Stabler, I learned the awe-striking clarity that a computational model can bring to its subject. I hope that my dissertation shows at least a hint of that potential. Herbert Enderton's eye for detail has also been of great value. For their roles in the development of my intellect and of my dissertation, then, I cannot sufficiently thank the members of my committee. For the imperfections that I have introduced along the way, I humbly apologize.

I benefited, naturally, from all of the time that I spent in the company of the faculty of the UCLA Linguistics department. I thank all of them, especially the following: Colin Wilson and Kie Zuraw, who were very generous with their time when I wanted help exploring various ideas; Bruce Hayes, who showed me all manner of pedagogical tricks that helped be an effective TA; Pat Keeting and Daniel Buring, who allowed me to serve as their research assistants in the summer and prevented me from going broke; and Pam Munro, who showed me through example how to evaluate the quality of research independent of its assumed framework. It was also a great privilege to attend the mini-course offered by Peter Svenonius during his visit to UCLA; my discussions with him were invaluable in understanding many aspects of spatial linguistics.

I thank also the regents of the University of California Los Angeles for the dissertation year fellowship that supported a portion of this research, as well as Noa Golding and Alessandro Tonchia of Business Semantics, Inc., who kept me well employed for the duration of the thesis.

My stay at UCLA was enhanced by many magnificent colleagues, foremost among them Greg Kobele and Jason Riggle. These are two of the most intelligent human beings I have ever encountered and they volunteered their time so generously to lead my entering class through early morning workshops on various topics. They were equally generous with their time when they served as TAs in my courses and when I asked them for guidance in my research. I profited from every moment that I spent in their company.

Other colleagues whose assistance with this dissertation must be acknowledged include Harold Torrence, Leston Buell, John Foreman, Marcus Smith, Ying Lin, Katya Pertsova, David Schueler, Jeff Heinz, Sarah VanWagenen, Michael Pan, Ananda Lima, Andy Martin, Heather Willson, Julie Morgenlender, Christina Kim, Sameer ud Dowla Kahn, Benjamin Jones, Byron Ahn, Asia Furmanska, Chad Vicenik, Ben George, Roy Becker-Kristal, Kristine Yu, Heather Burnett, Denis Paperno, and Natasha Abner. I thank each of you for your help and I beg the forgiveness of anyone I have forgotten.

I wrap up my acknowledgments for UCLA with a huge “THANK YOU” to the linguistics department staff, who have been so helpful over the years in all matters large and small. Natasha Levy, Kathryn Roberts, Melanie Levin, Lisa Harrington, and Anya Essiounina: without you, this would have been impossible.

My education did not begin, of course, at UCLA. From my early childhood my patient parents—Stan and Kathy Keil—, my generous elder brothers—John and C. Edward Emmer—, and my godmother Gloria Dugan were always ready

to help me investigate whatever it was that caught my curiosity (mathematics and grammar included).

I received most of my primary and secondary education at Burris Laboratory School, a division of Ball State University. I had many wonderful teachers there including Donna Biggs, Richard Hays, Brad Meyerdierks, Pamela Popovich, and Nancy Watson. I am especially grateful to Ron Bullock, Theresa Greenwood, and Martha Kendrick, who all encouraged me to excel rather than coast. Most of all, I am indebted to Bruce Robbins who, by example, taught me to read analytically, think critically, and recognize important commonalities across disparate works.

My junior year of high school was spent at the Albert Schweizer Gymnasium in Crailsheim, Germany. The families of Agathe Masserer and Werner Schebesta were kind enough to provide me with both hospitality and instruction in the German language and the Swabian/Franconian/Hohenlohish way of life.

I began the study of linguistics at Indiana University Bloomington. My introduction to the field and much of my development in it came from the lectures of Stuart Davis. One could not ask for a better beginning. Kenneth de Jong, George Fowler, Julie Auger, and Samuel Obeng also provided first-class instruction. It was Steven Franks who encouraged and convinced me to apply to UCLA. I am thankful that I had the opportunity to learn from each of them.

Finally, I wish to thank a few friends who have inspired me at various points along the long journey from birth to doctorate. These are Kavon Behforouz, Matthew Robey, Matthew Molter, Ryan Kinney, Laura Robinson, Sally O'Brien, Leta Echelbarger, Kristen Murphy, Elaine Vozar, Brandon Mundell, Stanley Florek, Heather Frankland, Jim Richmond, Sarah Hoff, Sara Fox, Kelley Runyon Lewis Farmer, Molly Johnson, Adam Johnson, Janelle Boys-Chen, and Kurt Lawson. I have been quite fortunate to know each of you.

VITA

1979	Born, Muncie, Indiana, USA.
1991	Completed 7th year, Hampstead School, London, England
1995	Completed 10th class, Albert Schweizer Gymnasium, Crailsheim, Germany
1997	Graduated, Burriss Laboratory School, Muncie, Indiana
2001	B.A. Linguistics and Cognitive Science with concentration in Computer Science, Indiana University Bloomington
2005	M.A. Linguistics, University of California Los Angeles
2005–	Technical Team Lead, Business Semantics, Inc.

PUBLICATIONS

An Associative Semantics for Malagasy. Proceedings of the 12th Annual Conference of the Austronesian Formal Linguistics Association

ABSTRACT OF THE DISSERTATION

Generating Descriptions of Motion from Cognitive Representations

by

Benjamin Keil

Doctor of Philosophy in Linguistics

University of California, Los Angeles, 2010

Professor Marcus Kracht, Chair

This dissertation presents a novel method of sentence generation, drawing on the insight from Cognitive Semantics (Talmy, 2000a,b) that the effect of uttering a sentence is to evoke a Cognitive Representation in the mind of the listener. Under the assumption that this Cognitive Representation is also present in the speaker and defines (part of) the speaker's communicative intention, sentence generation is seen as the process of transforming Cognitive Representations into sentences. Starting with a brief exposition of the Cognitive Representations of motion events, this dissertation demonstrates how Cognitive Representations can be rendered as graphs, the graphs can be analyzed (parsed) with a graph grammar, and the parse term obtained from the graph can be transformed into the linguist's traditional representation of a sentence, a syntactic tree.

The process of generation is presented in two phases: graph parsing and parse transformation. The graph parsing phase itself has two sub-phases. An initial descriptive phase motivates graphs as a means of depicting Cognitive Representations and tours the development of graph grammars from string grammars. The following analytic phase introduces flowgraphs, a formalism closely related

to graphs. It is demonstrated that from any graph a unique flowgraph (up to isomorphism) can be obtained. The analytic phase finishes with a detailed exhibition of an algorithm for parsing flowgraphs based on the work of Lutz (1996).

The parse transformation phase shows first how the important details of a parse can be recorded in a parse term while also discarding irrelevant details of the parse. The transformation of the parse term into a syntactic tree is then manifested with a top-down deterministic macro tree transducer (Engelfriet and Vogler, 1985), and the generative powers of macro tree transducers is briefly explored.

CHAPTER 1

Introduction

1.1 Goal

The goal of this dissertation is to exhibit a method for generating linguistic expressions, specifically descriptions of motion. By “generate” I intend not merely the enumeration of the grammatical descriptions of motion, but also the choice among these grammatical descriptions to meet some communicative need. I mean by “generation,” in other words, the process of taking a particular meaning, perhaps one gleaned from an observation of a motion event, and constructing a linguistic expression that encodes that meaning.

A method of generation, then, requires an idea of what meanings are. Where can we look for answers? Semantics is nominally the study of meaning, and semanticists in general reduce a linguistic expression to its truth conditions. One might, therefore, work on the recovery of a linguistic expressions from its truth conditions. Given the right kind of truth conditions, such recovery is possible, and this is indeed a tenable method of sentence generation. For example, a first-order logic (FOL) formula that expresses the truth conditions of a sentence in Pratt-Hartmann’s two-variable English (E2V) can be run backwards through the interpretation process to recover the precise E2V sentence that generated it. The transduction from FOL to E2V is interesting, but it does not answer (or even suggest any answers to) any questions about how the observation of an event

could lead to a description of that event. As an alternative, this work explores the use of the Cognitive Representations (CRs) of motion events laid out by Talmy (2000a,b) as the source from which sentences are generated. Talmy's theory of CRs is psychologically motivated and driven by an exhaustive typological study of motion event descriptions across several language families.

In particular, the fundamental claims of this dissertation are listed in (1.1):

- (1.1) a. Cognitive representations of events are a useful starting point for generating linguistic expressions describing those events.
- b. Cognitive representations can be rendered as graphs.
- c. Using a graph grammar to parse the graph of a cognitive representation yields a parse term that can be transduced to a structured linguistic expression.
- d. A graph can have multiple parses and each parse term corresponds to a different linguistic expression with the same meaning.

In addition, my dissertation suggests that cognitive representations are a linguistic level of representation whose existence—like the existence of Phonological Form and of Logical Form—is necessary to satisfy the criteria laid out by Chomsky (2002, pp. 85–91) for the adequacy of a theory of language. The necessity of this level of representation is closely related to (1.1-d). Just as understanding the homonymy of the two morpheme sequences “an aim” and “a name” is best facilitated by reference to a phonological level of representation where the two are the same (or very close), the synonymy of the two sentences in (1.2) also motivates a level of representation on which they are the same. This level is certainly not Phonological Form, and it also does not seem to be Logical Form, where the sentences in (1.2) would be expected to have different representations, although

their representations might be related in some theories by meaning postulates. The two sentences will, however, have the same cognitive representation. Even a more abstract logical form is not likely to suffice. The two sentences in (1.3) are truth-conditionally equivalent, but represent different conceptualizations of the situation they depict.

- (1.2) a. The bottle floated into the cave.
 b. The bottle entered the cave floating.
- (1.3) a. A line of trees extends from the beach to the bluff.
 b. A line of trees extends from the bluff to the beach.

1.1.1 Why Motion is a Suitable Focus

Certainly, any practical system for generating sentences and texts should account for more topics than just motion. In this dissertation I focus on motion for two reasons. First, the Cognitive Semantics literature is better developed in its descriptions of motion-related language. Second, our understanding of space and motion seems to shape our understanding of many other realms.

1.1.2 Place of this Dissertation within Talmy's Program

Talmy (2000b, p. 22) outlines the steps in (1.4) as a means to develop a cognitive semantics:

- (1.4) ("entities" = elements, relations, and structures: both particular cases
 and categories of these) (ibid., p. 22)
 - a. Determine various semantic entities in a language.

- b. Determine various surface entities in the language.
- c. Observe which (a) entities are expressed by which (b) entities—in what combinations and with what relationships—noting any patterns.
- d. Compare (c)-type patterns across different languages, noting any metapatterns.
- e. Compare (c)-type patterns across different stages of a single language, noting any shifts or non-shifts that accord with a (d)-type metapattern.
- f. Consider the cognitive processes and structures that might give rise to the phenomena observed in (a) through (e).

This dissertation builds on the work that has already been done on (1.4-a-e) and concentrates on a new step, (1.4-g):

- (1.4) g. Validate, through modeling and computation, that the processes and structures considered in (f) actually can give rise to the phenomena (a) through (e).

1.1.3 Place of this Dissertation within Levelt’s “Blueprint for the Speaker”

Levelt (1989) offers a comprehensive cognitive model of the process that starts with the intention to speak and ends with the articulation of an utterance that more or less conveys the speaker’s intention. The model includes several processing components, interconnected such that the input of one component is the output of another. At a coarse level, there are the CONCEPTUALIZER, which creates from the intention a “preverbal message”; the FORMULATOR, which pro-

duces a “phonetic plan” (or “internal speech”) from the preverbal message; and the ARTICULATOR, which produces overt speech from the phonetic plan. These elements are complemented by the AUDITION, which captures the overt speech and produces a phonetic string, and the SPEECH COMPREHENSION SYSTEM, which parses both the internal phonetic plan and the phonetic analysis and informs the CONCEPTUALIZER, which monitors the parse for any deviations from the intention and can issue new preverbal messages to correct them when they arise.

In this dissertation, I am concerned with the action of the FORMULATOR. I assume that I already have preverbal messages available to me, in the form of Cognitive Representations. Moreover, I am only concerned with that part of the FORMULATOR that Levelt labels “grammatical encoding,” which is emphasized in bold in Figure 1.1.

Guhe (2007) is an in-depth investigation of a cognitively motivated, incremental implementation of the CONCEPTUALIZER, called INC. The approach taken by Guhe dovetails with this dissertation in spirit, although there is some disparity between the preverbal messages generated by INC and the cognitive representations that I use to drive the formulation of sentences.

1.1.4 This dissertation with respect to “definitions”¹

This dissertation might be seen as advocating a definitional theory of language, and some scholars object to any theory of language that revolves around definitions. I maintain that I do not use cognitive representations as definitions and so

¹This section draws heavily from discussions with both Kracht and Stabler; if I convey any insight into the topic, then much of it is due to them. Any over-simplifications and failures of argumentation are, of course, wholly my own.

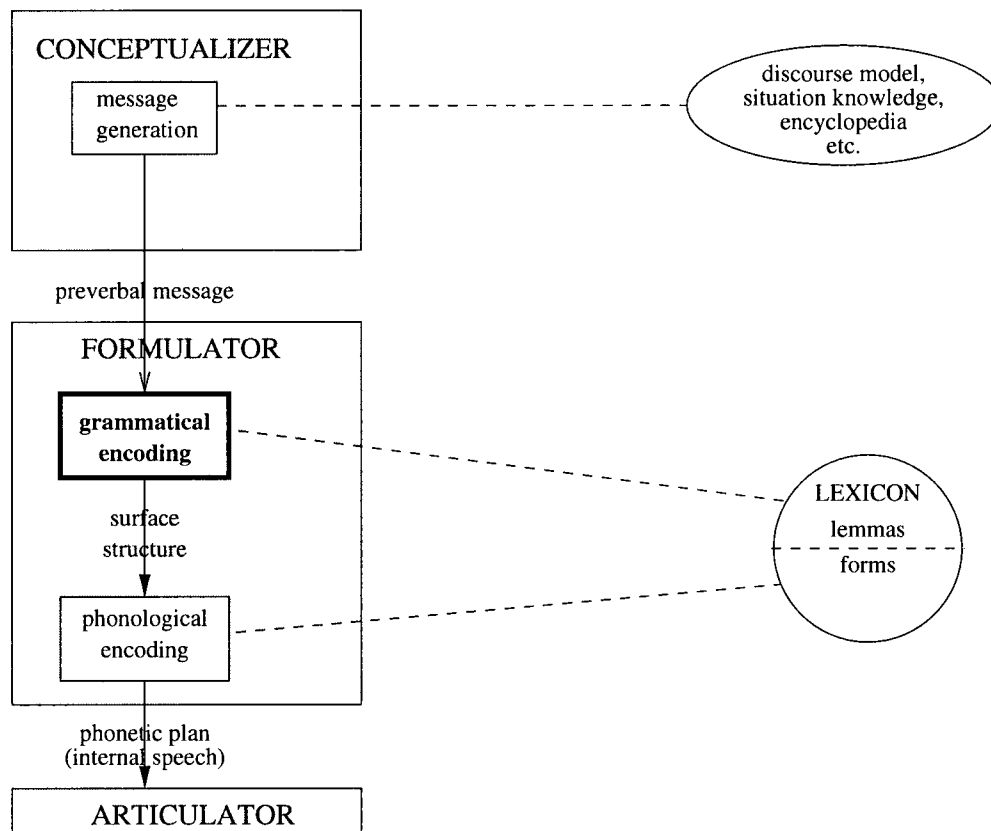


Figure 1.1: A blueprint for the speaker. Simplified from *ibid.*, p. 9

the content of this dissertation retains its value whether one accepts definitional theories of language or rejects them.

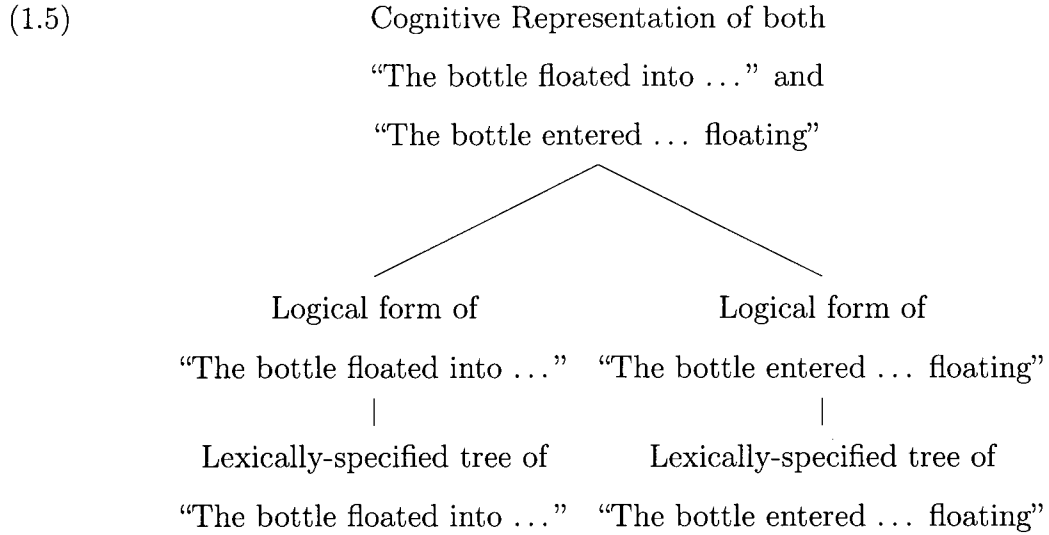
Fodor et al. (1980) are among those who reject definitional theories of language, rejecting specifically the idea “that the morphemes of a natural language typically have internal structure at the ‘semantic level’,” which they refer to as “The Standard Picture” (TSP). They cite a lack of direct evidence, a paucity of persuasive examples, and a failure of definitions to actually fulfill the needs of definitional theories. They also present experimental evidence that suggests to them that definitions do not play an active part in adult comprehension of language. As the language generation process that I present in this dissertation does involve positing some relationship between, say, ‘enter’ and the pair ‘move’ and ‘into’, it falls to me to explore to what extent my work is subject to their criticism.

To develop their criticism of definitions, Fodor et al. (ibid.) use the word ‘bachelor’ and its widely recognized definition: “unmarried male.” They first object to this definition on the grounds that the terms ‘unmarried’ and ‘male’ in the definition are not obviously closer to the “primitive basis of language” than ‘bachelor’ is. Although I make no claims in this dissertation about the primitive basis of language, one may consider Talmy to do so. Talmy’s *deep-* and *mid-level morphemes* (the closest elements to definitions in Talmy’s work) are, however, based on rigorous typological surveys across several language groups. In this light, my use of Talmy’s typological work to motivate my framework for generation is comparable to the use of the typological work of Jakobson (1962) by Prince and Smolensky (1993, chap. 6) to motivate their optimality-theoretic analysis of syllable structure.

A second objection of Fodor et al. (1980) to definitions is their interposition

between syntax (“lexically-specified trees”) and logical form and what they see as TSP’s use of this position to “underwrite the validity of informally valid arguments.” Returning to ‘bachelor’, this refers to the validity of an argument like “John is a bachelor. Therefore John is unmarried.” If a lexical item is replaced by its definition in the logical form of a sentence, then the logical form of “John is a bachelor.” is something like “John is a male and John is unmarried.” The conclusion “Therefore John is male,” would therefore be an instance of the familiar “ \mathcal{P} and \mathcal{Q} . Therefore \mathcal{P} .” This breaks down, however, when other informally valid arguments are tested. Fodor et al. (1980) give the example of “ x is red. Therefore x is colored.” There is no definition of ‘red’ that makes the argument valid. If one takes ‘colored’ to be the defined term, and gives it a definition like “red or blue or green or cyan or magenta or yellow or...” then the argument is valid, but the definition appears to be intractable. In this dissertation, I do not place cognitive representations between syntax and logical form. Rather, I consider cognitive representations to be sources² of logical forms, with the logical forms of both *The bottle floated into the cave* and *The bottle entered the cave floating* as being connected to the same source cognitive representation, as in (1.5).

²“Sources,” that is, for the purpose of generation. For other purposes they may be derived objects.



The reader may wonder if I intend the two sentences at the leaves in (1.5) to have exactly the same meaning. The goal of this dissertation is the generation of descriptions of motion events from Cognitive Representations, so my intention is that the two sentences are equally good reports of the observation of a bottle moving across the surface of a body of water from a point outside of a cave to a point inside of that cave. Cognitive Representations of motion events are extracted from such observations, abstracting away from several details (the decision as to which elements are kept in this abstraction and which are discarded is the work of the CONCEPTUALIZER in Figure 1.1). Thus, two sentences with the same Cognitive Representation may differ subtly in meaning in ways that are not attended to by the generation system.

Meaning postulates (Carnap, 1952) are a proposed alternative to definitions. In a system built around meaning postulates, knowledge of the English word ‘bachelor’ includes knowledge of the postulate “ x is a bachelor just in case x is male and x is unmarried,” but the lexical item ‘bachelor’ is a peer to lexical items such as ‘male’ and ‘unmarried’, rather than deriving its meaning from

them. Fodor et al. (1980) prefer meaning postulates to definitions for the underwriting of informally valid arguments. Meaning postulates, they say, have greater descriptive power than definitions; definitions are essentially limited to biconditional statements, where as all logical operations are available to meaning postulates. There can be, therefore, a meaning postulate that “ x is red implies x is colored,” underwriting an argument where definitions failed to do so.

The use of cognitive representations to generate sentences parallels the use of meaning postulates to assess the validity of an argument. Perhaps what Talmy (2000a,b) is cataloging is just a system of regularities in the meaning postulates that arise in the domain of motion and the lexical items whose behavior they govern.

1.2 Other Approaches to Language Generation

The claims in (1.1) lay out an approach to generation that seeks to combine the insights of the cognitive and generative linguistic traditions. Other traditions in linguistics and logic have inspired quite different models of generation. The ways in which the absence of either a cognitive or generative inspiration cause them to differ from the thesis developed here are elaborated below.

1.2.1 Systemic Functional Linguistics

Systemic Functional Linguistics is a school of linguistics that centers around the reasons that humans use language, the *functions* that language performs. For example, Halliday (1975, p. 37) lists seven functions that language performs for children:

Instrumental language allows a child to acquire what is wanted or needed

Regulatory language allows a child to control the behavior of others by giving commands

Interactional language allows a child to form and maintain interpersonal relationships

Personal language allows a child to express opinions and feelings

Heuristic language allows a child to gain knowledge about the environment

Imaginative language allows a child to tell fanciful stories and jokes

Representational language allows a child to share information with others

Halliday (1970) argues that every function that language performs touches to varying degrees on each of the three *metafunctions* of language. These are the *ideational*, the *interpersonal*, and the *textual* metafunctions. The ideational function of language is to “express content,” to “give structure to experience” and to “determine our way of looking at things.” The interpersonal function of language is to “establish and maintain social relations,” to express social roles, to “[get] things done,” to “delimit social groups,” to identify and reinforce the individual, and to “[enable] him to interact with others.” The textual function of language is to “links with itself and with features of the situation in which it is used,” to establish “cohesive relations from one sentence to another in a discourse,” to enable a speaker or writer to form a coherent text, and to enable a listener or reader to distinguish a coherent text from a random list of sentences (ibid., p. 175).

Systemic Functional Linguistics is also a *stratificational* theory of linguistics in the sense of Lamb (1966), with the strata in (1.6). In a stratificational system, each stratum is said to “realize” the stratum above it and to “abstract” the

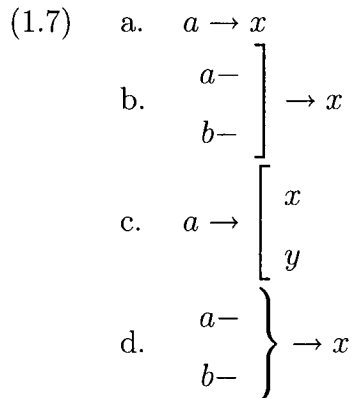
stratum below it. A “behavioral potential” is realized as a “linguistic potential” which is then realized as sound. Semantics and phonology are considered to be intermediate steps which facilitate these realizations.

(1.6)

SUBSTANCE	(interlevel)	FORM	(interlevel)	CONTEXT
Sounds	Phonology	Grammar	Semantics	Situation

(Teich, 1999, p. 13)

The term *systemic* refers to the way that each stratum realizes the stratum above it through a series of choices. Each local choice is called a *system* and the collection of all systems dealing with a particular grammatical rank (e.g., morpheme, word, phrase, clause) is called a *system network*. Each system is drawn with an *entry condition* on the left and an *output term* on the right, and a rightward pointing arrow in the middle as in (1.7-a). Disjunction is allowed in the entry condition, as in (1.7-b), where it means “if either *a* or *b* then choose *x*.” It is also allowed in the output term, as in (1.7-c), with the meaning “if *a* then choose either *x* or *y*.” Likewise, conjunction is allowed in the entry condition (1.7-d), and in output term (1.7-e), where it is called “simultaneity” and is interpreted as “if *a* then simultaneously choose *x* and choose *y*.”



$$\text{e. } a \begin{cases} \rightarrow x \\ \rightarrow y \end{cases}$$

Systemic linguists recognize some patterns in system networks as recurring frequently and have given them names. For example, one system in a network often represents a refinement of another system, such as the system for choosing mood in (1.8). In a grammar containing this system, one can only choose between interrogative and declarative mood if indicative has already been chosen over imperative mood. This is called a *delicacy* ordering.

$$(1.8) \quad \text{MOOD} \rightarrow \begin{cases} \text{indicative} \\ \text{imperative} \end{cases} \rightarrow \begin{cases} \text{interrogative} \\ \text{declarative} \end{cases}$$

(Teich, 1999, p. 12)

Another example is a system where two choices made independently force a third choice, as in (1.9), where independent choices in the topic of a sentence (called here its “theme”) and the topic’s type force a particular choice for the topic’s role. The forced choice is called a *gate*.

$$(1.9) \quad \begin{cases} \dots \rightarrow \text{unmarked} \\ \dots \rightarrow \text{nonpredicated theme} \end{cases} \rightarrow \begin{cases} \text{interpersonal} \\ \text{textual} \\ \text{other} \\ \text{experiential} \\ \text{other-simple} \\ \text{identifying} \end{cases} \left. \vphantom{\begin{cases} \text{interpersonal} \\ \text{textual} \\ \text{other} \\ \text{experiential} \\ \text{other-simple} \\ \text{identifying} \end{cases}} \right\} \rightarrow \text{most-inherent-role}$$

(adapted from *ibid.*, p. 122)

Teich (1999) explores the use of Systemic Functional Linguistics in natural language generation. The input to a systemic generation system is tied the choices in the system network being used. At each choice point, the generator uses the input to decide between the multiple options. A typical input is (1.10), where the top line of each box is a concept with an associated variable and the remaining lines are attribute-value pairs. The expected output for (1.10) is “Kim devours the cookies.”

(1.10)	<table> <tr> <td data-bbox="424 696 617 737">devour_d</td><td data-bbox="617 696 948 896"></td></tr> <tr> <td data-bbox="424 737 617 778">actor:</td><td data-bbox="617 737 948 778"> <table> <tr> <td data-bbox="617 737 737 778">person_p</td><td data-bbox="737 737 948 778"></td></tr> <tr> <td data-bbox="617 778 737 819">name:</td><td data-bbox="737 778 948 819">Kim</td></tr> </table> </td></tr> <tr> <td data-bbox="424 819 617 896">actee:</td><td data-bbox="617 819 948 896"> <table> <tr> <td data-bbox="617 819 737 896">cookie_c</td><td data-bbox="737 819 948 896"></td></tr> <tr> <td data-bbox="617 896 737 935">identifiability:</td><td data-bbox="737 896 948 935">identifiable</td></tr> <tr> <td data-bbox="617 935 737 976">number:</td><td data-bbox="737 935 948 976">plural</td></tr> </table> </td></tr> <tr> <td data-bbox="424 976 617 1017">time:</td><td data-bbox="617 976 948 1017">present</td></tr> <tr> <td data-bbox="424 1017 617 1058">theme:</td><td data-bbox="617 1017 948 1058"><i>p</i></td></tr> <tr> <td data-bbox="424 1058 617 1099">speech act:</td><td data-bbox="617 1058 948 1099">statement</td></tr> </table>	devour _d		actor:	<table> <tr> <td data-bbox="617 737 737 778">person_p</td><td data-bbox="737 737 948 778"></td></tr> <tr> <td data-bbox="617 778 737 819">name:</td><td data-bbox="737 778 948 819">Kim</td></tr> </table>	person _p		name:	Kim	actee:	<table> <tr> <td data-bbox="617 819 737 896">cookie_c</td><td data-bbox="737 819 948 896"></td></tr> <tr> <td data-bbox="617 896 737 935">identifiability:</td><td data-bbox="737 896 948 935">identifiable</td></tr> <tr> <td data-bbox="617 935 737 976">number:</td><td data-bbox="737 935 948 976">plural</td></tr> </table>	cookie _c		identifiability:	identifiable	number:	plural	time:	present	theme:	<i>p</i>	speech act:	statement
devour _d																							
actor:	<table> <tr> <td data-bbox="617 737 737 778">person_p</td><td data-bbox="737 737 948 778"></td></tr> <tr> <td data-bbox="617 778 737 819">name:</td><td data-bbox="737 778 948 819">Kim</td></tr> </table>	person _p		name:	Kim																		
person _p																							
name:	Kim																						
actee:	<table> <tr> <td data-bbox="617 819 737 896">cookie_c</td><td data-bbox="737 819 948 896"></td></tr> <tr> <td data-bbox="617 896 737 935">identifiability:</td><td data-bbox="737 896 948 935">identifiable</td></tr> <tr> <td data-bbox="617 935 737 976">number:</td><td data-bbox="737 935 948 976">plural</td></tr> </table>	cookie _c		identifiability:	identifiable	number:	plural																
cookie _c																							
identifiability:	identifiable																						
number:	plural																						
time:	present																						
theme:	<i>p</i>																						
speech act:	statement																						

(ibid., p. 64)

The system network approach makes the finely detailed decisions of generation quite easy to implement. The choice of whether or not to use the word ‘the’ in a sentence like “Kim devours the cookies” is decided by having a specific indicator of identifiability in the input and having a system that chooses to use ‘the’ when an object is identifiable. When the system has to work with larger syntactic constituents, however, Systemic Functional grammars suffer a “syntagmatic gap” (ibid., pp. 2, 52–53), a lack of any way to express syntactic generalizations. In generative grammar, on the other hand, devising abstract syntactic representa-

tions and defining rules over them has been one of central focuses of the theory, (see Chomsky, 2002, for example).

Also, any input to a system network is by necessity highly specified. Among other things, (1.10) includes a specification of which verb to use in the sentence, so one could presumably not produce both of the sentences in (1.2) from the same input and system network. The formalized Cognitive Representations developed in this thesis are designed to do precisely this. The CRs developed here do, however, fall short in some ways that the Systemic Functional Grammars do not. Most notably, I will not develop here any means of generating “referring expressions” like ‘the cookies’. The generation of referring expressions is an extremely complicated endeavor (see Reiter and Dale, 2000, for an overview), and would detract from the focus on motion events.

1.2.2 Meaning-Text Theory

Meaning-Text Theory (Mel’čuk, 1981) presents another model of language that is very relevant to the work in this dissertation. It shares the stratificational heritage of Systemic Functional Linguistics, but it also places a high value on paraphrase. As a preface to an article on generation within Meaning-Text Theory (MTT), Iordanskaja, Kittredge, and Polguère (1991) write

One measure of the power and completeness of a language model is its ability to represent all the possible ways that a human speaker could choose to say “the same thing” by using linguistic knowledge (as opposed to world knowledge).

The importance of this measure is two-fold. First, it is important for the theory of linguistics, as it serves to measure the grammatical and lexical coverage of the

model. Second, is important for the practical application of a language model to the generation problem, as generators sometimes face conflicting constraints that must be “paraphrased around.”

MTT aims to “cover *all* possible *linguistic* knowledge ... governing the usage of words in texts” (Iordanskaja, Kittredge, and Polguère, 1991, citing Mel’čuk, 1984) in formal dictionaries called *Explanatory Combinatorial Dictionaries* (see Mel’čuk, 1988, for a systematic exposition). The linguistic knowledge is stratified across seven levels of representation:

- (1.11) a. Semantic Representation (SemR)
- b. Deep Syntactic Representation (DSyntR)
- c. Surface Syntactic Representation (SSyntR)
- d. Deep Morphological Representation (DMorphR)
- e. Surface Morphological Representation (SMorphR)
- f. Deep Phonetic Representation (DPhonR)
- g. Surface Phonetic Representation (SPhonR)

Within MTT there are at least four recognized means of paraphrase:

- (1.12) a. **Reductions:** within SemR, a number of simple morphemes may be replaced by a semantically more complex morpheme. For example, reducing (i) to (ii):
- (i) The referred-to user(s) of the system used (before now), during the referred-to period of 7 hours, 32 minutes and 12 seconds, more than one program of a type such that someone compiles something with these programs and someone edits something with these programs.

- (ii) The aforementioned users of the system used programs of compiler type and of editor type during the referred-to period of 7 hours, 32 minutes and 12 seconds. (Iordanskaja, Kittredge, and Polguère, 1991, p. 300).
- b. **Alternative Maximal Reductions:** to be as brief as possible requires applying as many “reductions” as are applicable, but it is often the case that there are several maximally reduced SemR’s derivable from the same unreduced SemR. The sentences (i)–(iii) are sentences portraying three such maximal reductions:
 - (i) Fred limped across the road quickly.
 - (ii) Fred hurried across the road with a limp.
 - (iii) Fred crossed the road, limping quickly. (ibid., p. 301)
- c. **Passage from SemR to DSyntR:** once the semantic representation has been chosen, the next step is to structure the information into a sentence. Something must become a subject. Something must be predicated of that subject. These choices are encoded in DSyntR. The different choices for the subject of the same SemR:
 - (i) The user who ran editors is named Martin.
 - (ii) The name of the user who ran editors is Martin.
 - (iii) Martin is the name of the user who ran editors. (ibid., p. 302).
- d. **Passage from DSyntR to SSyntR:** even when the lexical choices and the information structure are set, there are still several choices to be made in the surface syntactic representation. For example, should a relative clause be a full or reduced relative clause (*users who run editors* vs. *users running editors*. MTT also considers operations like “dative shift” (*give Mary the book* vs. *give the book to Mary*) to be surface phenomena.

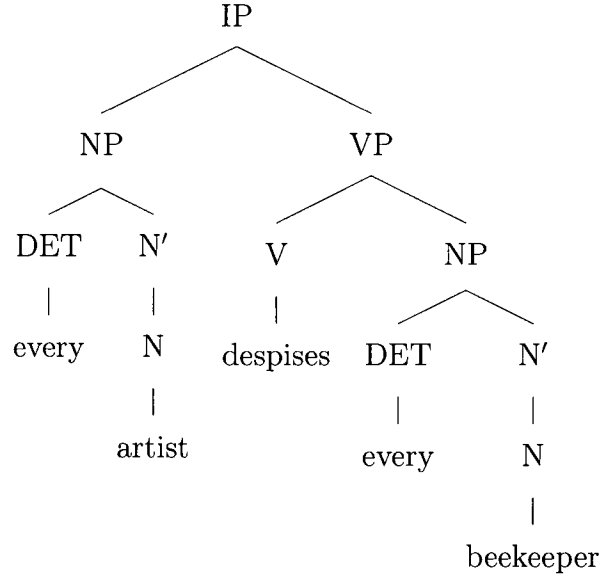
In this dissertation, I won't investigate the kinds of paraphrase in (1.12-c,d), but the trio of sentences in (1.12-b. i-iii) bears a strong resemblance to the pair in (1.2). In fact, there is a remarkable similarity between Talmy's work and MTT; even more striking as MTT is driven by what its researchers deem necessary for formal linguistic description and Talmy is motivated by theories of cognition. A reader coming from an MTT perspective would be kind to consider this whole dissertation as an implementation of (1.12-a,b).

1.2.3 Pratt-Hartmann

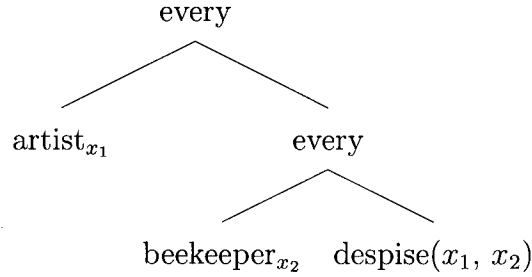
Pratt-Hartmann (2003, 2005) has done much work on mappings between logics “whose computational characteristics are well-understood” and “regimented fragment[s] of a natural language.” (Pratt-Hartmann, 2003, p. 14) In particular, he has studied two different two-variable fragments of first order logic: \mathcal{L}^2 , which does not have counting quantifiers and \mathcal{C}^2 which does. These fragments are interesting because they are decidable logics, although the decision problem is NEXPTIME-hard in both cases. The goal of his work, then, is similar to the goal of this dissertation.

Pratt-Hartmann has identified (i.e., constructed a grammar for) a moderately-rich fragment of English called “E2V”, for “two-variable English.” The grammar produces syntactic trees such as (1.13-a). Along with his grammar, he gives an interpretation function that maps the each of grammar's syntactic trees to an intermediate logical form, which can then be translated into a statement in \mathcal{L}^2 . (1.13-a), for example, has the intermediate form (1.13-b) and the \mathcal{L}^2 interpretation (1.13-c).

(1.13) a.



b.



c. $\forall x_1(\text{artist}(x_1) \rightarrow \forall x_2(\text{beekeeper}(x_2) \rightarrow \text{despise}(x_1, x_2)))$

Pratt-Hartmann (2003, p. 37) also provides a translation function from \mathcal{L}^2 back into E2V. This translation is way of generating language, and its scope is quite impressive. It will generate a text for any formula in \mathcal{L}^2 and it will preserve a relation of mutual entailment between the interpretation of the text and the formula. It is not as impressive—from the point of view of someone who intends to generate natural language—in terms of its results. Although Pratt-Hartmann makes use of a generative grammar to specify and interpret E2V, the grammar is entirely ignored in the translation from logic into E2V. Instead, an E2V text is produced by choosing a sequence of sentences drawn from just the 15 templates listed in Table 1.1. Impoverishing the syntax of E2V has severe consequences for

$\forall x \forall y (a(x, y) \rightarrow b(y, x))$	Everything bees everything which ays it
$\forall x (\exists y a(x, y) \rightarrow b(x, x))$	Everything which ays something bees itself
$\forall y (\exists x a(x, y) \rightarrow b(y, y))$	Everything which something ays bees itself
$\forall x (b(x, x) \rightarrow \forall y a(x, y))$	Everything which bees itself ays everything
$\forall x (b(x, x) \rightarrow \forall y a(y, x))$	Everything ays everything which bees itself
$\forall x (\exists y a(x, y) \rightarrow b(x))$	Everything which ays something eqs a bee
$\forall x (\exists y a(y, x) \rightarrow b(x))$	Everything which something ays eqs a bee
$\forall x (b(x) \rightarrow \forall y a(x, y))$	Every bee ays everything
$\forall x (b(x) \rightarrow \forall y a(y, x))$	Everything ays every bee
$\forall x \forall y (a(x, y) \rightarrow \neg b(x, y))$	Nothing bees something which it ays
$\forall x \forall y (a(x, y) \rightarrow b(x, y))$	Everything bees everything which it ays
$\forall x \forall y a(x, y)$	Everything ays everything
$\forall x \exists y a(x, y)$	Everything ays something
$\forall x \forall y (\neg b(x, y) \rightarrow a(x, y))$	Everything ays everything which it does not bee
$\forall x \forall y ((b(x, y) \wedge c(x, y)) \rightarrow a(x, y))$	Everything which bees something which it cees ays it

Table 1.1: Pratt-Hartmann’s normal form conjuncts *ibid.*, p. 38

the translation, as the following example will demonstrate.

The translation function first translates the logical sentence into a normal form, a conjunction of terms each of the form $\forall x \forall y \phi$ or $\forall x \exists y \phi$, where in each case ϕ is quantifier-free.³ Moreover, there are (up to renaming of predicate letters) only finitely many formulas ϕ that need to appear in this normal form (those appearing on the left hand side of Table 1.1). Pratt-Hartmann’s method of translating \mathcal{L}^2 into E2V is to translate each conjunct of the \mathcal{L}^2 sentence into the E2V sentence to its right in Table 1.1. The \mathcal{L}^2 sentence has the same truth conditions as the conjunction of all the \mathcal{L}^2 interpretations of the resulting translations.

For example, his syntax for E2V allows sentences like the one in (1.14-a), which his semantics interprets as (1.14-b):

³The forms in Table 1.1 are in this form, but some have the quantification “late,” that is, with one of the quantifiers “pushed down” into the embedded subformula. This is Pratt-Hartmann’s program of “efficient” quantifier use that lets him express formulas with two variables that would require more than two variables if quantification were done “early.”

- (1.14) a. Some artist does not despise every beekeeper.
 b. $\exists x_1(\mathbf{artist}(x_1) \wedge \neg \forall x_2(\mathbf{beekeeper}(x_2) \rightarrow \mathbf{despise}(x_1, x_2)))$. Pratt-Hartmann, 2003, p. 26

To generate an E2V sentence or text in a mutual entailment relation with (1.14-b) (henceforth ϕ) following Pratt-Hartmann's algorithm, first one must convert it into Scott normal form. The normalization process, outlined by Grädel and Otto (1999, p. 88), starts by finding the minimal subformula of ϕ with quantifier-depth one, in this case (1.15-a), and assigning that subformula to a new sentence letter, ψ^1 (my notation). The process then creates a new unary predicate letter P_{ψ^1} and an assertion θ_{ψ^1} (1.15-b), which guarantees that an assignment of an individual to x_1 makes $P_{\psi^1}(x_1)$ true just in case it makes ψ^1 true as well—intuitively, then, P_{ψ^1} is the property possessed by all and only those individuals that hate every beekeeper. The next step is to substitute any occurrence of ψ^1 in ϕ with P_{ψ^1} , creating ϕ' as in (1.16). The conjunction of ϕ' and θ_{ψ^1} is truth-functionally equivalent to ϕ .

- (1.15) a. $\forall x_2(\mathbf{beekeeper}(x_2) \rightarrow \mathbf{despise}(x_1, x_2))$
 b. $\theta_{\psi^1} =_{def} \forall x_1 \forall x_2 (P_{\psi^1}(x_1) \leftrightarrow (\mathbf{beekeeper}(x_2) \rightarrow \mathbf{despise}(x_1, x_2)))$

- (1.16) $\phi' =_{def} \exists x_1(\mathbf{artist}(x_1) \wedge \neg P_{\psi^1}(x_1))$

The process then recurses through ϕ' , picking out the new smallest subformula with quantifier-depth 1 (which in this case is the whole of ϕ'), and giving it a new sentence letter, ψ^2 . The newly created predicate letter P_{ψ^2} is governed by the assertion θ_{ψ^2} in (1.17). This makes P_{ψ^2} a very strange kind of property; given θ_{ψ^2} is true, P_{ψ^2} is true of all individuals if there is some artist who does not hate every beekeeper and false of all individuals if no such artist exists. The truth

of ψ^2 (which has no unbound variables), then, is equivalent to P_{ψ^2} being true of any individual at all, so we can substitute all occurrences of ψ^2 in ϕ' with $P_{\psi^2}(\mathbf{c})$, where \mathbf{c} is a special constant used by the translation process. The result of this substitution is ϕ'' (1.18). ϕ'' contains no quantifiers, so at this point the end of the recursion has been reached, and the resulting formula is given a special sentence letter, $\hat{\phi}$.

$$(1.17) \quad \theta_{\psi^2} =_{def} \forall x_1 \exists x_2 (P_{\psi^2}(x_1) \leftrightarrow (\mathbf{artist}(x_2) \wedge \neg P_{\psi^1}(x_2)))$$

$$(1.18) \quad \hat{\phi} =_{def} \phi'' =_{def} P_{\psi^2}(\mathbf{c})$$

Having reached $\hat{\phi}$ through recursion, the next step in the process is to set $\phi^* = \exists x_1 \hat{\phi}[\mathbf{c}/x]$ and collect the θ_{ψ^n} generated during the recursion. The conjunction of all of these elements, (1.19), is the Scott normal form of ϕ .⁴

$$(1.19) \quad \forall x_1 \exists x_2 P_{\psi^2}(x_2) \wedge \forall x_1 \exists x_2 ((\mathbf{artist}(x_2) \wedge \neg P_{\psi^1}(x_2)) \leftrightarrow P_{\psi^2}(x_1)) \\ \wedge \forall x_1 \forall x_2 ((\mathbf{beekeeper}(x_2) \rightarrow \mathbf{despise}(x_1, x_2)) \leftrightarrow P_{\psi^1}(x_1))$$

Already at this point, we are far from anything that looks like an interpretation of the original E2V sentence (1.14-a), but the translation into Pratt-Hartmann's normal form will take us further afield.

The first step in conversion to Pratt-Hartmann normal form is to rewrite the quantifier free subformulae in the Scott normal form using only negation and conjunction, as in (1.20), referred to by Pratt-Hartmann as $f'(\phi)$.

$$(1.20) \quad \forall x \exists y (P_{\psi^2}(y)) \wedge$$

⁴Presumably the vacuous—in a non-empty universe—universal quantification on ϕ^* is also part of the process, but it is not mentioned in Grädel and Otto, 1999

$$\begin{aligned}
& \forall x \exists y (\neg(\neg(\mathbf{artist}(y) \wedge \neg P_{\psi^1}(y)) \wedge P_{\psi^2}(x)) \wedge \neg(\neg P_{\psi^2}(x) \wedge (\mathbf{artist}(y) \wedge \\
& \neg P_{\psi^1}(y)))) \wedge \\
& \forall x \forall y (\neg(\neg\neg(\neg \mathbf{beekeeper}(y) \wedge \mathbf{despise}(x, y)) \wedge P_{\psi^1}(x)) \wedge \neg(\neg P_{\psi^1}(x) \wedge \\
& \neg(\neg \mathbf{beekeeper}(y) \wedge \mathbf{despise}(x, y))))
\end{aligned}$$

The next step is to get all of the atoms in (1.20) to be of the form $b(x, y)$, in the process creating new binary relation letters a_1, \dots, a_n as well as many more conjuncts which express the restrictions on those new relation letters. Pratt-Hartmann refers to the result of applying (1.21) to $f'(\phi)$ as $f''(\phi)$.

(1.21) Replace an atom not of the form $b(x, y)$ with a new binary relation $a_i(x, y)$. Repeat as long as there still exist atoms not of the form $b(x, y)$.

- a. Replace $P_{\psi^1}(x)$ with $a_1(x, y)$, imposing the following conditions:
 - (i) $\forall x ((\exists y a_1(x, y)) \rightarrow P_{\psi^1}(x))$
 - (ii) $\forall x (P_{\psi^1}(x) \rightarrow \forall y a_1(x, y))$
- b. Replace $P_{\psi^1}(y)$ with $a_2(x, y)$, with the conditions:
 - (i) $\forall x \forall y (a_2(x, y) \rightarrow a_1(y, x))$
 - (ii) $\forall x \forall y (a_1(x, y) \rightarrow a_2(y, x))$
- c. Replace $P_{\psi^2}(x)$ with $a_5(x, y)$, $P_{\psi^2}(y)$ with $a_6(x, y)$, imposing conditions as before; likewise $\mathbf{artist}(x)$ with $a_7(x, y)$, $\mathbf{artist}(y)$ with $a_8(x, y)$, and again $\mathbf{beekeeper}(x)$ with $a_3(x, y)$, and finally $\mathbf{beekeeper}(y)$ with $a_4(x, y)$.

After (1.21), the first conjunct of (1.20) is now $\forall x \exists y a_6(x, y)$, which is one of the forms on the left-hand side of Table 1.1. Moreover, all of the conditions imposed by (1.21-a) and (1.21-b) were also forms from that table; the transformation

process is nearing its end.

The conversion to Pratt-Hartmann normal form proceeds by working a similar transformation on the smallest non-atomic units in the result of (1.21). Because (1.20) was written using only conjunction and negation, there are exactly two kinds of minimal non-atomic sentences. An example of the translation of each form is given in (1.22).

- (1.22) Replace the smallest non-atomic subformula with a new binary relation $a_i(x, y)$. Repeat as long as there still exist non-atomic subformula.
- a. The subformula $\neg P_{\psi^1}(y)$ of (1.20), which was replaced with $\neg a_2(x, y)$ in step (1.21), is now replaced with $a_9(x, y)$, and the following conditions are placed:
 - (i) $\forall x \forall y (a_9(x, y) \rightarrow \neg a_2(x, y))$
 - (ii) $\forall x \forall y (\neg a_2(x, y) \rightarrow a_9(x, y))$
 - b. The subformula $(\mathbf{artist}(y) \wedge \neg P_{\psi^1}(y))$ of (1.20), which was replaced with $(a_8(x, y) \wedge a_9(x, y))$ in (1.21), is replaced by $a_{10}(x, y)$ imposing three conditions:
 - (i) $\forall x \forall y ((a_8(x, y) \wedge a_9(x, y)) \rightarrow a_{10}(x, y))$
 - (ii) $\forall x \forall y (a_{10}(x, y) \rightarrow a_8(x, y))$
 - (iii) $\forall x \forall y (a_{10}(x, y) \rightarrow a_9(x, y))$

After (1.22) has been exhaustively applied to $f''(\phi)$, all that will remain is atomic formulas of the form $a_i(x, y)$. In this manner (1.20) will be transformed to (1.23), imposing in the process 60 conditions on the interpretation of the 26 new predicate letters introduced during the translation. Let Θ be the conjunction of all of the conditions imposed by steps (1.21) and (1.22), then the Pratt-Hartmann normal form of ϕ , $f(\phi)$ is the conjunction of (1.23) and Θ .

$$(1.23) \quad \forall x \exists y a_6(x, y) \wedge \forall x \exists y a_{26}(x, y) \wedge \forall x \forall y a_{25}(x, y)$$

Each of these conjuncts (the three in (1.23) and the 60 in Θ) will be translated into an E2V sentence, making a text of 63 sentences where one solitary sentence of E2V would have been sufficient. Moreover, contained in these 63 sentences are 26 new transitive verbs (“ay-one,” “ay-two,” . . . , “ay-twenty-six,” which are interpreted as $a_{1\dots 26}$) and 2 new nouns (“psi-one” and “psi-two,” which are interpreted as $P_{\psi 1\dots 2}$). The text will contain sentences like the translation of (22-b-i), given as (1.24); it seems somewhat unfair to call this a “fragment of a natural language.”

$$(1.24) \quad \text{Everything which ay-eights something which it ay-nines ay-tens it.}$$

To be fair, the goal of Pratt-Hartmann’s translation algorithm is not to provide a means of translating \mathcal{L}^2 into English (or E2V), but to prove that any \mathcal{L}^2 formula has an E2V text with the same (i.e., a mutually entailing) meaning. For those formulas of \mathcal{L}^2 that are actually interpretations of E2V sentences—(1.14-b), for example—it is relatively easy to retreat from the interpretation back to the intermediate form and from there back to the original E2V sentence. Moreover, the set of E2V interpretations is directly generable by a simple grammar. Given a means of selecting a relevant member of the set of all E2V interpretations, then, one could craft a reasonable system for generating English sentences.

1.2.4 Generation from Proof Nets

Where Pratt-Hartmann’s (2003) work centers around proving the equivalent expressive power of E2V and \mathcal{L}^2 given the ability to create arbitrary novel nouns and verbs in E2V, another tradition in generation focuses on deriving sentences from their interpretations using a fixed vocabulary. In this tradition the lexical

items of a language are treated as resources to be used to build sentences and their interpretations. Associating a sentence and an interpretation is done by proving that the resources used to build one can be used in the same configuration to build the other.

Traditional logic does a poor job of managing resources, but Girard (1987) introduces a logic “as strong as intuitionism but more subtle,” calling it *linear logic*. Linear logic is a “resource conscious” logic, meaning that propositions are treated as resources that can only be used once. So, if one uses a proposition \mathcal{A} and a proposition $\mathcal{A} \Rightarrow \mathcal{B}$ to prove a proposition \mathcal{B} , then, at the end, all one has available is the proposition \mathcal{B} ; the other two propositions have been used up. This can be difficult to appreciate with the standard semantics for propositional logic, but—adapting an example from Wadler (1993)—it is much easier to appreciate if one takes \mathcal{A} to mean “I have a five dollar bill”, \mathcal{B} to mean “I have a pizza,” and $\mathcal{A} \Rightarrow \mathcal{B}$ to mean “I can exchange a five dollar bill for a pizza.” If there is an additional proposition on the table, for example, $\mathcal{A} \Rightarrow \mathcal{C}$ “I can exchange a five dollar bill to get a plate of brownies,” then it becomes clear that from one instance of \mathcal{A} , i.e., the possession of one five dollar bill, one can obtain either \mathcal{B} , a pizza, or \mathcal{C} , a plate of brownies, but not both. The choice of \mathcal{B} or \mathcal{C} given the availability of \mathcal{A} is written $\mathcal{A} \vdash \mathcal{B} \& \mathcal{C}$. Now, two five-dollar bills *can* be exchanged for both a pizza and a plate of brownies, written $\mathcal{A}, \mathcal{A} \vdash \mathcal{B} \otimes \mathcal{C}$.

Girard (1987) also introduces *proof nets*, a natural deduction system for linear logic. Proof nets are proofs with multiple conclusions and are structured in such a way that they avoid any spurious ambiguity, much like a traditional syntactic derivation tree (see Chomsky, 2002, p. 27).

Lambek (1958) aims to “obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences, which works not only for ... formal languages

..., but also for natural languages such as English.” To this aim, he develops a calculus for combining syntactic types. The calculus starts with basic types, like n , for a nominal (e.g., ‘John’, or ‘the dog’) and s for a sentence. More complicated types are built from these basic types using two “slashes,” a forward slash $/$ (read “over”) and a backward slash \backslash (read “under”). These slashes determine how the types combine with each other, such that $A \backslash B$ combines with A on its *left* to yield B and B/A combines with A on its *right* to yield B . For example, the type $n \backslash s$ may be assigned to ‘sleeps’, because placing ‘John’ of type n to its left produces ‘John sleeps’ of type s . Any sequence of words, taken together with an assignment of types, can be shown to either yield s or not yield s , thus distinguishing sentences from nonsentences.

The “Lambek Calculus” anticipated the resource sensitivity of linear logic. Having used a noun in a sentence does mean that the noun can occur arbitrarily many times; each use must be accounted for. More concretely, given one instance each of ‘the’, ‘dog’, and ‘eats’, one can construct the sentence “The dog eats” but not the sentence “The dog eats the dog.” The latter requires two instances each of ‘the’ and ‘dog’, just as an interested buyer requires two five dollar bills to obtain both a pizza and a plate of brownies.

Roorda (1992) takes advantage of the two systems’ similarities to adapt the proof nets of Linear Logic to the Lambek Calculus. Lecomte (1993) outlines an efficient parsing algorithm using Lambek Calculus proof nets. de Groote and Retoré (1996) demonstrate that the proof nets used for the Lambek Calculus and other “categorical grammars” are a “unique structure that allows the syntactic and semantic aspects of sentence analysis to be unified,” giving a correspondence between proof nets and typed λ -terms. In the unified analysis, a proof net encoding a syntactic analysis is combined with a “semantic recipe” (also a proof net) for

$\text{runs} - \text{run} : N \backslash S, \text{the} - \text{the} : N / CN, \text{dog} - \text{dog} : CN \implies \text{the} + \text{dog} + \text{runs} - \mathbf{w} : S$

Figure 1.2: A judgment in the logic of Merenciano and Morrill (ibid.).

each lexical item, directly yielding the semantic reading of the sentence.

In the same spirit as de Groote and Retoré (1996), Merenciano and Morrill (1997) use a unified syntactic and semantic analysis for the Lambek Calculus as a framework for generation. The logic in their framework deals with “type assignment statements.” A type assignment statement consists of three parts. The first part is a sequence of pronounced or written elements, which they name “prosodics.” The second part is a semantic term, which they initially introduce as a term from the λ -calculus but later replace with a proof net. The third element is a syntactic category, a type from the Lambek calculus. A bag (i.e., a collection that, unlike a sequence, is unordered and that, unlike a set, allows a member to have more than one membership) of type assignment statements is called a “configuration.” A judgement in their logic combines a configuration (the antecedent) with a type assignment statement (the succedent) and indicates that “if the objects referred to in the antecedent are in the types indicated, then the object referred to in the succedent is in the type indicated.” A variable can stand in for either the semantics or the prosodics. If, as in Figure 1.2, a variable stands in for the semantics, then proving that the given prosodics are the pronunciation of a sentence is tantamount to parsing and the proof process will ground the variable for the semantics, giving the interpretation of the sentence corresponding with the parse. If, on the other hand, a variable stands in for the prosodics, proving that the semantics are the interpretation of a sentence is *generation*, grounding the prosodics variable with the pronunciation of a sentence with the given interpretation.

Generation from proof-nets takes its syntactic queues from the Lambek cal-

culus, so it does not suffer from the “syntagmatic gap” that hinders generation based on Structural Functional linguistics. Issues of cognition are not often addressed in the proof-net generation literature, but some work in this field is quite suggestive. A frequently used example (see Merenciano and Morrill, 1997; Pogodalla, 2000, for examples) for proof-net based generation systems is generating the sentence “John seeks Mary” from a semantic representation built from the predicates ‘try’ and ‘find’. Presumably the sentence “John tries to find Mary” would be an equally good output. From there it is just a small step to generating the two sentences in (1.2) from a common representation. In fact, the approach to graph grammars taken by Engelfriet and Vereijken (1997), in which graphs are built up by concatenating sequences of “graph operations,” indicates that results should be highly portable between the framework developed in this dissertation and those frameworks that are used for generating from proof-nets.

1.3 Dissertation Overview

This dissertation tells the story of how Cognitive Representations correspond to the sentences that evoke them. The story starts with the introduction of the main character, the CRs themselves (Chapter 2). The story continues with the CRs being rendered as graphs and parsed using graph grammars (Chapter 3). In the story’s conclusion, the parses are transduced into sentences (Chapter 4).

Chapter 2’s introduction of Talmy’s theory of Cognitive Representations (Section 2.1) acquaints the reader with the fundamental cognitive components of motion events: *Figure*, *Ground*, *Path*, and *Motion*. It also puts forward Talmy’s concept of a *Co-Event*, which is an event distinct from but integrally connected to a motion event. Section 2.2 explores the *Motion* component in greater detail, especially its relation with agency and how it gets lexicalized in English and

Spanish.

Chapter 3 starts with a short overview (Section 3.1), and then discusses the problem of rendering CRs as graphs (Section 3.2). Section 3.6 details the parsing of CRs using flowgraph grammars (detailed in Section 3.1) after Section 3.3 provides a review of string grammars and traces the history of their generalization to graph grammars. The conversion between graphs and flowgraphs is explained in Section 3.5. Chapter 3 ends with a detailed exposition of a flowgraph parse in Section 3.7.

The first section of Chapter 4 gives an outline of how Chapters 2 and 3 fit together to set the stage for sentence generation: the CRs of the former yield parsers in the latter. While parses are general analyzed as terms, linguists are more comfortable using trees for describing sentences, so Section 4.2 gives formal definitions for both trees and terms, and then demonstrates that the two are isomorphic. Section 4.3 explains the connection between parses and parse-terms, showing first how parse terms are given for string derivations and then generalizing from there to graph derivations. Finally, Section 4.4 details the process of transducing a parse term into a traditional syntactic tree.

CHAPTER 2

Cognitive Representations

2.1 Overview of Cognitive Representations

Cognitive Representations, according to Talmy (2000a, p. 21), are a “particular kind of experiential complex” that a sentence evokes in listeners. I assume that the same kind of experiential complex exists in the speaker of a sentence. I also assume that when a speaker utters a sentence, she intends to evoke in her listeners a cognitive representation similar to the one that exists in her.

The Cognitive Representations of motion events in particular are explored in depth by Talmy (2000b). Talmy offers a detailed account of how the mind represents motion events, based on various linguistic universals and typological generalizations. The basic constituents that he ascribes to motion events are: the *Figure*, the subject of an observation; the *Ground*, the (relatively fixed) background against which the Figure is observed; *Motion*, the fact the Figure has either changed its position relative to the Ground or that it has not, and the extent to which the Figure seems to be acting of its own volition; and the *Path*, the description of how the Figure changed its position relative to the Ground. Talmy also identifies several kinds of *Co-Events* that are integrally connected to the perception of motion; for example there are “self-contained motions”—also called “Manners” of motion—, which are changes in the orientation, posture, or configuration of the Figure that do not directly change the Figure’s relation to

the Ground. Other examples would be *Causes* of motion (e.g., a push or kick), *Precursors* of motion (events which precede the motion event but do not cause or assist it), and *Enablings* of motion.

Talmy (2000b) also focuses on which syntactic constituents correspond to which constituents of the cognitive representation. One syntactic constituent frequently corresponds to multiple constituents of the cognitive representation. In this case, the syntactic constituent is said to *conflate* its cognitive correspondents, and each language seems to have a single major pattern of conflation that it uses to encode motion events;¹ the following examples demonstrate a few of these conflation patterns.

2.1.1 Motion + Co-Event

One common pattern across languages is the conflation of Motion and Co-Event. As a concrete example, suppose a person observes the situation described in (2.1). If that person speaks English, one might expect the person to report her observation as in (2.2):

(2.1) A bottle is floating on a lake. The bottle moves—travelling across the surface of the lake—from a point on the exterior of a cave to a point on the interior of the cave.

(2.2) The bottle floated into the cave.

¹Talmy's claim is actually stronger, but see Croft et al. (2008), for example, where it is convincingly argued that

...the Talmy typology is not a typology of how a language encodes complex events in general, but rather a typology of how particular complex event types are encoded by a language. Languages make use of multiple strategies to encode complex events, depending on the type of complex event involved.

Talmy claims that there are direct relations between the syntactic constituents of the sentence and the cognitive constituents of its meaning. In the case of (2.2), Talmy claims the following relationships: “the bottle” is the Figure, “the cave” is the Ground, the prepositional element “into” is the Path, and the verb “floated” is a conflation of both the Motion and the Manner of that motion. This pattern of relationships—i.e., where the verb conflates the Motion with the Manner—is the typical pattern for English speakers. Other examples of this pattern are given in (2.3); moreover, Co-Events of several other types conflate with Motion in English, as in (2.4).

- (2.3) a. The woman swam into the cave.
 b. The duck waddled around the lake.
 c. The snake slithered out of its hole.
 d. The dancer shimmied across the stage.
- (2.4) a. The leaf blew into the cave. (*Cause*, that is, something blowing on it caused the leaf to blow into the cave.)
 b. The man wore jeans into the office. (*Concomitance*)
 c. The glass splintered onto the carpet. (*Precursion*)

Talmy reports that this pattern is the norm for not only English, but for most of the Indo-European language family. The major exception to this generalization is found in the modern Romance languages, the subject of the next section.

2.1.2 Motion + Path

Modern Romance languages, for example, seem to prefer verbal roots that conflate the Motion with the Path, and have few roots that express simultaneously

Motion and Co-Event. Accordingly, one might expect a Spanish-speaking observer of (2.1) to give the report in (2.5):

- (2.5) La botella entró a la cueva (flotando). (Talmy, 2000b, p. 49)
the bottle move+in to the cave (floating)
“The bottle floated into the cave”

In this case, the verb ‘entró’ expresses both Motion and the fact that the Path ends at an interior location, while the manner of motion—if it is expressed at all—must be given separately as a gerund at the end. Spanish has many verbs that exhibit the same behavior:

- (2.6) From *ibid.*, pp. 49–50:

- a. La botella salió de la cueva (flotando).
the bottle move+out from the cave (floating)
“The bottle floated out of the cave.”
- b. La botella pasó por la piedra (flotando).
the bottle move+by past the rock (floating)
“The bottle floated past the rock.”
- c. El globo subió por la chimenea (flotando).
the balloon move+up through the chimney (floating)
“The balloon floated up the chimney.”
- d. El globo bajó por la chimenea (flotando).
the balloon move+down through the chimney (floating)
“The balloon floated down the chimney.”

2.1.3 Motion + Figure

In addition to Motion with Path and Motion with Co-Event, Talmy identifies Motion with Figure as another major pattern of conflation across the world's languages. It is not a frequent pattern in English, but Talmy does offer two

examples, shown in (2.7).

(2.7) From Talmy, 2000b, p. 57:

- a. It rained in through the bedroom window.
- b. I spat into the cuspidor.

In the case of (2.7-b), however, it should be noted that ‘spit’ also invokes some notion of Path as well, as indicated by the contrast with (2.8-a), which presumably involves motion of the same kind of Figure. ‘Spit’ is also compatible with a slightly wider range of Figures, as shown in (2.8-a).

- (2.8)
- a. I drooled into the cuspidor.
 - b. My mouth was full of juice when I spat on him.

Even if (2.7-b) is less than convincing as an instance of Motion + Figure, it does fit into a larger pattern in English. Motion + Figure seems to be the default pattern used for describing substances leaving the body. In addition to ‘spit’, the bold faced verbs in (2.9) follow this pattern, as do numerous slang words for male ejaculation (e.g., ‘jizz’, ‘skeet’, ‘spoooge’, and ‘spunk’, to name but a few).

- (2.9)
- a. The unruly patron **puked** all over the bar.
 - b. If your dog **pees/pisses** on the carpet, you should punish it.
 - c. Before we trained her, the cat would frequently **poop/shit** on the floor.
 - d. It felt like someone was **breathing** down my neck.
 - e. The child skinned her knee and **bled** on the pavement.

Atsugewi (a Hokan language of Northern California), according to Talmy (2000b, p. 58), is a language where this pattern is the fundamental pattern for talking about motion events. It features a large variety of verb roots that indicate that the Figure belongs to some particular class. Two of these roots are given in (2.10). One might object that these verb roots (as Talmy freely admits) “typically function equally in the expression of events of location, of nonagentive motion, and of agentive motion,” and that they therefore do not conflate the Figure with any kind of Motion. What Talmy presumably intends in this case is that these verbs are used to describe motion events and not, say, eating events.

(2.10) From Talmy (*ibid.*, p. 58):

-lup- ‘for a small shiny spherical object (e.g., a round candy, an eyeball, a hailstone) to move/be located’

-caq- ‘for a slimy lumpish object (e.g., a toad, a cow dropping) to move/be located’

2.1.4 Implications for Sentence Generation

So different languages have different major patterns that they use to encode cognitive representations into linguistic expressions, and even within a language certain lexical items use a different pattern than the norm (e.g., English ‘enter’ and ‘spit’). The generation process, from this point of view, involves selecting lexical items whose usages cover the cognitive representation that is to be expressed. In some cases, especially when roots not following a language’s primary pattern are involved, the selection is ambiguous. It seems not unreasonable to assume that the same cognitive representation could be the source of both sentences in (2.11), for example:

- (2.11) a. The officer walked into the room.
 b. The officer entered the room walking.

This aspect of the generation process makes it similar to parsing, where syntactic categories must be selected to carve up a linguistic expression and ambiguities also arise. There are (at least) two parses of (2.12), for example; it could be a sequence of two noun phrases, “the old man” and “the ships”, or it could be a noun phrase “the old” (meaning ‘old people’), the verb “man” (‘occupy stations on’) and, again, the noun phrase “the ships.” I will take advantage of this similarity between sentence parsing and sentence generation by demonstrating a process for parsing cognitive representations.

(2.12) the old man the ships

I am not proposing that cognitive representations are string like. The nature of cognition seems to be fundamentally non-linear. There is no clear ordering between the constituents of a cognitive representation, and the relationships between those unordered constituents may be many-to-many. A cognitive representation that contains both Motion and Manner constituents, for example, connects both of these to the same Figure. It is for this reason that I propose to use graphs for cognitive representations.

2.2 More about Motion

Talmy recognizes a few kinds of Motion. Fundamentally, there are two: MOVE, which “refers to the presence of motion in the event,” and BE_{LOC}, which refers to the presence of “locatedness” in the event (Talmy, 2000b, p. 35). These two are

deep morphemes in Talmy’s parlance, meaning that they represent “a concept that is believed to be both fundamental and universal in the semantic organization of language.” They are, moreover, the only two deep morphemes of Motion according to Talmy.

There is much more to Motion than just two deep morphemes. In particular there are many more of what Talmy calls *mid-level morphemes*. Each of these represents “a particular conceptual complex that consists of a deep-morphemic concept together with certain additional semantic material” and is “recurrent in the semantic organization of a particular language.” Mid-level morphemes are often found across several languages.

2.2.1 _AMOVE

One mid-level morpheme that occurs in several languages is _AMOVE, or “agentively cause to move.” The following sections give a few examples of how _AMOVE can surface in English and Spanish.

2.2.1.1 _AMOVE in English

_AMOVE can surface in English on its own—usually as ‘move’ like in (2.13-a)—, but most verb roots in English that express _AMOVE conflate it with at least one of several kinds of Co-Events. An example of _AMOVE conflated with a Manner is given in (2.13-b), where the verb ‘slide’ describes what the Figure does. The verb ‘push’ in (2.13-c) describes what the Agent does, making that example illustrative of the conflation of _AMOVE and Cause. (2.13-d–f) show _AMOVE conflated with several more kinds of Co-Events.

- (2.13) a. I moved the car into the garage. (no Co-Event)

- b. The barkeep slid the beer to the patron. (Manner)
- c. The barkeep pushed the beer to the patron (Cause)
- d. The barkeep cracked the egg into the whiskey sour. (Precursion)
- e. The barkeep slammed the door shut at closing time. (Concurrent result)
- f. The barkeeper locked the cash in the safe. (Subsequence)

It is interesting, however, that A MOVE is not readily conflated with Co-Events of Concomitance. The examples in (2.14) demonstrate this; the (i) sentences are instantiations of $\text{MOVE} + \text{Concomitance}$ and the (ii) sentences are (infelicitous) attempts at instantiating $\text{A MOVE} + \text{Concomitance}$. One reason for this might be that, as Talmy (2000b, p. 46) writes, “[t]he concomitance relation is not robustly represented in English (thus speakers differ on their acceptance of [examples like (2.14-c-i)]).”

- (2.14)
- a. (i) She wore a green gown to the party.
 - (ii) Her mother *dressed her in/*wore her a green gown to the party.²
 - b. (i) The bullet whistled past his ear.
 - (ii) ??I whistled a bullet past his ear.
 - c. (i) The girl whistled past the graveyard.
 - (ii) *Someone whistled me past the graveyard.³

²Intended meaning: “Her mother made her wear a green gown to the party.”

³Intended meaning: “Someone made me whistle past the graveyard.” There is a grammatical reading of this sentence with the meaning “By whistling, someone (e.g., a traffic cop) caused me to move past the graveyard,” which is an instance of $\text{A MOVE} + \text{Cause}$.

Perhaps more interesting is the contrast between (2.15-b-ii) and (2.15-c-ii). Jackendoff and Goldberg (2004, pp. 540–541) as well as Zubizarreta and Oh (2007, p. 89) present arguments suggesting that the former but not the latter is more accurately interpreted as an instance of _AMOVE + Manner; essentially the whistling in (2.15-b) is causally related to the fact of the bullets motion while the whistling in (2.15-c) does not have that relationship to the girl’s motion. If this is so, the contrast between (2.15-b-ii) and (2.15-c-ii) is predicted, but the degraded nature of (2.15-b-ii) is unexpected, as _AMOVE and Manner can normally be conflated in English. Talmy (2000b, p. 47) tenders the following compromise: “the presumed difference between Manner and Concomitance may have the character more of a gradient than of a sharp deviation.” If one accepts this compromise, then (2.13-b) and (2.14-c-ii) show examples at the extremes of this gradient and (2.14-b-ii) is somewhere in the center.

_AMOVE is not limited to conflation with Co-Events. It occurs, for example, in the Motion + Ground given in (2.15). Note that in (2.15-a) the men are both the Agents of the _AMOVE predicate, but also the Figure of the caused motion event. It may be that all instances of Motion + Ground in English conflate _AMOVE with the Ground.

- (2.15)
- Men of the Fifth Indian Division had no idea where they were going when they emplaned.
 - Airline personnel will deplane any passengers under the influence of alcohol.
 - The angry French mob defenestrated two of Louis XVI's closest friends.

2.2.1.2 _AMOVE in Spanish

Spanish also has _AMOVE, but, following the primary pattern for Spanish, the verb roots expressing _AMOVE conflate it with Path. Also following the central Spanish pattern, any kind of Co-Event is expressed in an independent constituent.

(2.16) From Talmy (2000b, p. 51):

- a. Metí el barril a la bodega rodándolo
I-_AMOVED-in the keg to the storeroom rolling-it
“I rolled the keg into the storeroom.”
- b. Saqué el corcho de la botella retorciéndolo.
I-_AMOVED-out the cork from the bottle twisting-it
“I twisted the cork out of the bottle.”
- c. Quité el papel del paquete cortándolo
I-_AMOVED-off the paper from-the package cutting-it
“I cut the wrapper off the package.”

2.2.2 Other Mid-Level Motion Morphemes

Most other mid-level morphemes that occur in Talmy’s writings are built on top of _AMOVE. GO, for example, is roughly defined as “_AMOVE self,” PUT (sometimes PLACE) is “controlledly _AMOVE through limb motion but without body translocation,” and GIVE is “_AMOVE into the GRASP of.” The Spanish verbs in (2.16), for example, could be analyzed as conflating PUT with the Path, rather than simply _AMOVE.

Some mid-level morphemes are built up from BE_{LOC}, as well. One example Talmy (ibid., p. 39) gives of this form is COVER, with the definition in (2.17-a) and an sample instantiation in (2.17-b). Talmy interprets ‘checker’ in (2.17-b) as conflating COVER with a checkering Manner Co-Event.

- (2.17) a. COVER: BE_{LOC} all-over.
b. Streaks of light checkered the eastern clouds.

2.3 Conclusions

Talmy's program of Cognitive Semantics begins with the idea that a semantics of a sentence is a Cognitive Representation that the sentence evokes in a listener who hears it. Cognitive Representations for motion events have four major constituents—Motion, Path, Figure, and Ground—and a fifth, tightly cognitively linked Co-Event.

The constituents of a Cognitive Representation do not, in general, have a one-to-one correspondence with lexical items in any given language, as lexical items often conflate two or more of them. The patterns of conflation vary across languages and (to a lesser extent) also within a single language. Typological studies show that there are three major patterns of conflation: Motion+Co-Event, Motion+Path, and Motion+Figure.

Chapter 3 develops a more formal model of Cognitive Representations and explores the fine structure of Path. Much has been left unsaid about Figure and Ground, but their fine structures do not play a role in model of sentence generation developed in this thesis. An interested reader is referred to Talmy (2000a, pp. 311–339).

CHAPTER 3

Graphs, grammars, and parsing

3.1 Overview

Levelt (1989, p. 108) writes that “[t]he mother of each speech act is a communicative intention.” The previous chapter could be described using Levelt’s metaphor as a proposal that Cognitive Representations are the form that these “mothers” take on. In this chapter I show how each mother can have multiple offspring.

In order to give a formal account of how Cognitive Representations are processed into linguistic expressions, it is necessary to give them an analyzable form. Now, a Cognitive Representation (CR) has multiple constituents, such as the Motion, Figure, Path, Ground and Manner constituents of the CR of a motion event, so any method of rendering a CR must respect these constituents. The constituents of a CR have no clear linear order with respect to each other, so using strings, which enforce a particular linear ordering of their elements, is a sub-optimal strategy for rendering CRs. Graphs, on the other hand, do not enforce any ordering on their nodes, making them a more suitable candidate. Rendering Cognitive Representations as graphs is the subject of Section 3.2.

As discussed in Section 2.1.4 (p. 36ff), the constituents of a CR need not be in a one-to-one correspondence with the constituents of any linguistic expression evoking that CR. This opens up the possibility that there may be more than one way to select a set of lexical items that both combine to form a sentence

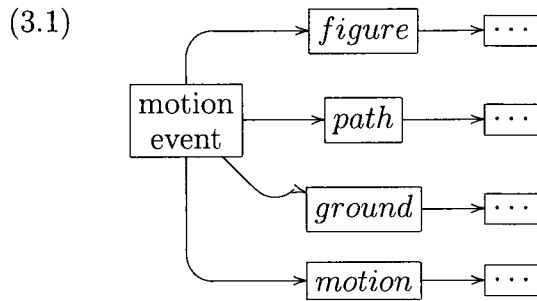
and evoke all of the CR. The potential of having multiple ways to group the constituents of a CR into sets that can be evoked by lexical items mirrors the situation in sentence parsing where there may be multiple ways to group the words of the sentence into constituents allowed by the grammar. The formal problem of grouping the constituents is cast as a parsing problem in Section 3.3. In that section, I introduce *flowgraph grammars*, a type of graph rewriting system. Graph rewriting systems are a generalization of string rewriting systems, falling into the same general categories as string grammars (i.e., regular, context-free, context-sensitive, etc.). Just like string grammars they can be used to establish derivation trees as proofs that a given graph is generated by a particular grammar, and—in the same way that string grammars can generate multiple parses for one string—graph grammars can generate multiple parses for one graph. I make use of the latter property to generate the two sentences in (2.11), for example, from the same graph, each sentence corresponding to a different parse of that graph.


An algorithm for parsing a flowgraph using a flowgraph grammar—due to Lutz (1996)—is sketched in 3.6.3. The details of implementing this algorithm are covered in 3.6.4. An example of the parsing running on a graph that is ambiguous with respect to the parsing grammar is detailed in 3.7.

3.2 Rendering Cognitive Representations as Graphs

In order to treat cognitive representations as graphs, it is necessary to be quite explicit about what elements they have and their constituency is. In the previous section, I mentioned that Talmy’s work picks out Figure, Ground, Motion, and Path as the major constituents of the cognitive representation of a motion event. This suggests that cognitive representation graphs should match a schema like

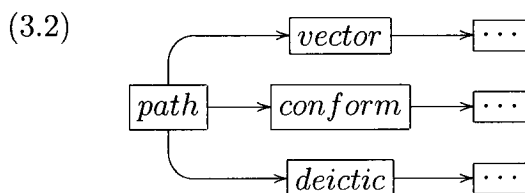
(3.1).



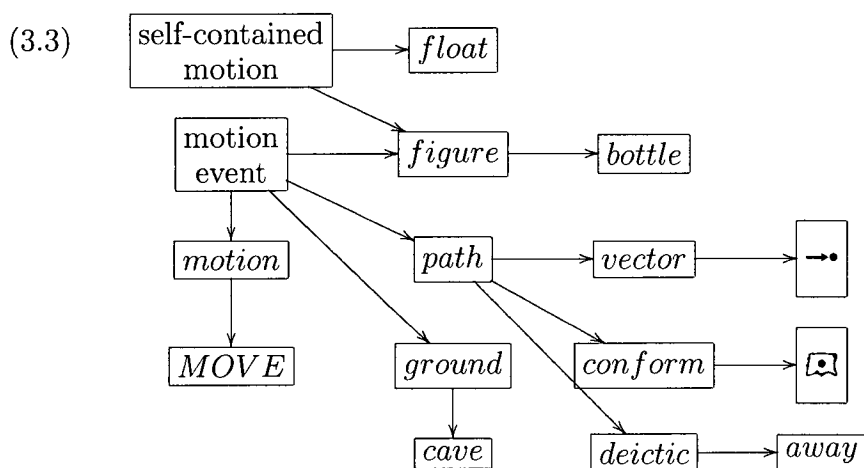
For Path, Talmy also outlines its fine structure: it has a *fundamental schema* for the Figure and the Ground, a *Vector*, a *Conform*, and a *Deictic*. The fundamental Figure schema, Talmy claims (see Talmy, 2000b, p. 53), is always a point, and the fundamental Ground schema (FGS) is one of the following: a point, a pair of points, a bounded extent, an unbounded extent, an extent bounded at the origin of the motion, or an extent bounded at the terminus of the motion. *Vectors* are the various schematic maneuvers that the fundamental Figure schema can execute with respect to the fundamental Ground schema. According to Talmy, they are drawn from a limited set of language-universal “deep prepositions.” The complete list of them is given in Table 3.1. Moreover, each Vector selects exactly one kind of fundamental Ground schema (also shown in Table 3.1), so I will treat them as a single constituent from now on. *Conforms* define the relation between the fundamental Ground schema and the Ground proper, for example for English ‘inside’, the conform is that the fundamental ground schema is *a point “which is of the inside of” the Ground* viewed as an enclosure (I offer  as an abbreviation for this Conform). The *Deictic* element of the path relates the motion to the point of view, for example English the path ‘move’ is neutral, that of ‘come’ is towards the point of view, and ‘go’ often indicates motion away from the point of view. The structure of a Path, then, is as in (3.2):



Vector	FGS
BE _{LOC}	a point
MOVE TO ($\rightarrow\bullet$)	a point
MOVE FROM ($\bullet\rightarrow$)	a point
MOVE VIA	a point
MOVE ALONG ($\rightarrow\boxminus$)	an unbounded extent
MOVE TOWARD	a point
MOVE AWAY-FROM	a point
MOVE ALENGTH ($\rightarrow\boxplus$)	a bounded extent
MOVE FROM-TO	a pair of points
MOVE ALONG-TO	an extent bounded at terminus
MOVE FROM-ALONG	an extent bounded at origin

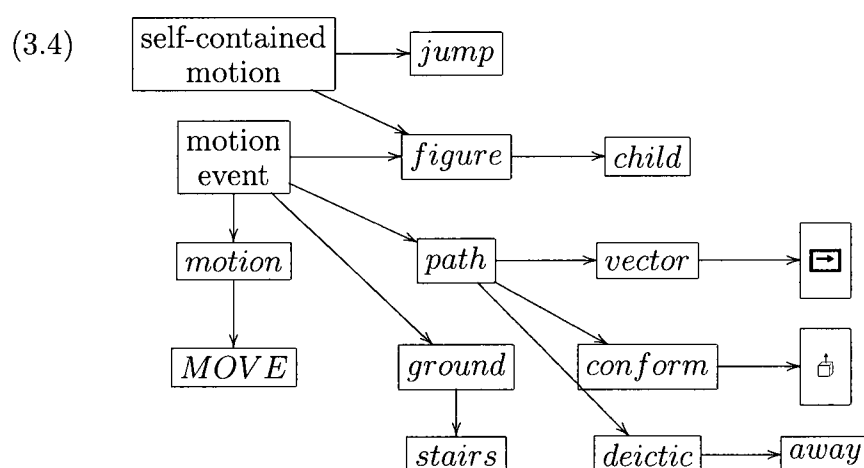
Table 3.1: Talmy's list of universal vectors (ibid., p.53f)



So far, these graphs are trees, but when relationships between the motion event and any co-events is represented, the graphs are no longer tree-like. For Manner, the important relationship is that the self-contained motion and the main motion event have the same figure, as in the fully articulated graph shown in (3.3):



Here the graph represents the conceptualization that is evoked both by *The bottle floated into the cave* and by *The bottle entered the cave floating*. Similar sentence pairs are represented by similar graphs. The graph in (3.4), for example combines the vector —Talmy’s “ALENGTH”, representing motion from the origin of a bounded extent to its terminus—with the conform  a bounded extend “which is aligned with the positive vertical axis of” the Ground, creating the description of a path from the bottom of the Ground to its top. English sentences evoking this conceptualization are *The child jumped up the stairs* and *The child ascended the stairs jumping*.



3.3 From String Grammars to Graph Grammars

The theory of graph grammars is a descendant of Chomsky’s (1956; 1963; 2002, etc.) work on string rewriting systems—which, in turn, is likely a descendant of Thue’s and Emil Post’s work on algebraic term rewriting. There are several extant graph grammar formalisms, but they can all be understood as starting with a particular conceptualization of strings (which varies between the different formalisms) and using that conceptualization to generalize string rewriting systems to more elaborate structures.

I shall start with the notion of a string as a *sequence* of elements chosen from a set of *symbols*. The symbols themselves are primitives, and the set they are drawn from is called an *alphabet*. An example alphabet, then, is $\{a, b, c\}$. A sequence is created by choosing elements from the alphabet and keeping track of the order in which they are chosen. The same element may be chosen several times. A way of writing a sequence is to write the elements (called the *coordinates*) of the sequence in the order they were chosen, separated by commas, and surrounded by angle brackets. So the sequence formed by choosing first c and then a is written $\langle c, a \rangle$. If s is an sequence of n elements, and i is whole number between 1 and n (inclusive), then s_i represents the i^{th} coordinate of s , so $\langle c, a \rangle_1$ is c and $\langle c, a \rangle_2$ is a .

A string rewriting system comprises an alphabet of symbols, one element of which is designated the start symbol, and a set of production rules, which are pairs of strings over the alphabet. The first element of a production rule is called its “left-hand side,” the second element is its “right-hand side,” and the rule is often written by placing the elements on the corresponding side of an arrow. When the rewriting system applies a production rule to a string, it replaces one substring that matches the left-hand side with the right hand side.

For example, let Σ be the alphabet $\{\text{also, amphipods, consume, copepods, eat, enthusiastically, oysters, that}\}$. Strings over Σ include $\langle \text{oysters} \rangle$, $\langle \text{oysters, eat} \rangle$, and $\langle \text{oysters, consume, amphipods} \rangle$. The production rule in (3.5-a) when applied to the string in (3.5-b) produces either of (3.5-c) or (3.5-d). A string rewriting system derives strings by beginning with its start symbol and applying a series of rewrite rules.

$$(3.5) \quad \text{a.} \quad \langle \text{eat} \rangle \rightarrow \langle \text{enthusiastically, consume} \rangle$$

- b. ⟨amphipods, that, oysters, **eat**, also, **eat**, copepods⟩
- c. ⟨amphipods, that, oysters, **enthusiastically**, **consume**, also, **eat**, copepods⟩
- d. ⟨amphipods, that, oysters, **eat**, also, **enthusiastically**, **consume**, copepods⟩

Often, the symbols of the alphabet are divided into two disjoint sets, the “terminal” and “non-terminal” symbols, and each production rule is required to have at least one non-terminal in its left-hand side. Any string that does not contain a non-terminal symbol, then, cannot be further rewritten by a system that makes the distinction. The set of all strings that a rewriting system can derive from its start symbol and that have no non-terminal symbols is called the “terminal language” of that system.

One way to conceptualize strings is as one-dimensional arrays. The generalization that suggests itself from this point of view is to create rewriting systems that use multidimensional arrays. As computer displays are two-dimensional arrays, early work in this direction was done with “picture languages.” The pioneering work was done by Narasimhan (1962), who developed a linguistically inspired process for analyzing photographs of the trail of bubbles that a charged particle leaves as it traverses a chamber of super-heated liquid. Other early work was done by Kirsch (1964), who developed a phrase-structure grammar for a fragment of English along with a grammar for pictures that could be described by the sentences in the fragment. Using the syntactic analysis of a sentence and that of a picture, Kirsch’s paper gives a method for determining if the sentence is true of that picture. Miller and Shaw (1968) provide a summary of further research into picture language grammars.

Strings (or, more generally, arrays) can also be conceptualized as a kind of graph. In this interpretation, each element in the string (array) is a node, and there are edges between nodes that correspond to adjacent elements in the string (cells in the array). Following the combined intuition of Kirsch and D. E. Knuth, Pfaltz and Rosenfeld (1969) used this notion of strings to develop the theory of “web grammars,” where the term “web” is used to mean a graph with a function labeling its nodes. Web grammars have production rules that rewrite a graph, exchanging one of its subgraphs (again, the “left-hand side” of the rule) for a replacement graph (the right-hand side of the rule).

The primary complication in generalizing from strings to graphs or other higher dimensional structures is the question of embedding. In the case of a string, there is just one way to embed the right-side of a production rule into its host string. Each substring neatly divides the string into two parts: the (possibly empty) part that precedes it, and the (possibly empty) part that follows it. When a substring is replaced, the resulting string is the concatenation of the preceding part, the replacement, and the following part. Relieving a graph of one of its subgraphs does not, in general, leave the graph in any predictable number of disconnected pieces, so a more elaborate scheme for embedding the replacement graph into the host is needed. Pfaltz and Rosenfeld (*ibid.*, pp. 610–611) write:

The definition of “rewriting rules” for webs is more complicated [than that for strings]; if we want to replace the subweb α of the web ω by another subweb β , it is necessary to specify how to “embed” β in ω in place of α . This can be done in many different ways; for example, one can specify that there be edges between given points in β and any points of $\omega - \alpha$ (e.g., having given labels, having given numbers of incoming or out-going edges, being on edges to or from particular

points of α in the original ω , etc. etc.) Any such specification of the edges between β and its “host web” will be called an *embedding* of β .

... It is important to emphasize that the definition of an embedding must not depend on the host web ω , since we want to be able to replace α by β in *any* web containing α as a subweb. Thus any properties of points in the host web that are used in defining the embedding must be well defined for an arbitrary ω .

A more specific means of determining the embedding of the replacement material is offered by a third conceptualization of a string. In this conceptualization, each element of the string is a building block with a label from the alphabet and two attaching points, namely the one on its left, and the one on its right. The more general structure suggested by this idea, then, is a building block with more than two attaching points. Feder (1971), working from ideas in Narasimhan (1966), makes this generalization and calls the resulting structures “ n -attaching point entities”—NAPEs, for short—and calls structures built from NAPEs “plexes.” If NAPEs distinguish input and output attaching points, then each connection between NAPEs has a direction and the structures built from such NAPEs are called “directed plexes.” The left- and right-hand sides of plex-rewriting rules are both plexes. Each side of the rule is outfitted with a list of distinguished attaching points called “tie-points.” The two lists are of equal lengths, allowing the right hand side to be embedded in the host graph by attaching each of its tie-points to whatever NAPEs in the host graph its correspondingly indexed tie-point on the left-hand side was attached to. NAPEs can be realized as webs, where a central node in the web is labeled with whatever symbol the NAPE bears and each attaching point of the NAPE is realized as a node connected to that central node, so plex grammars are isomorphic to a subset

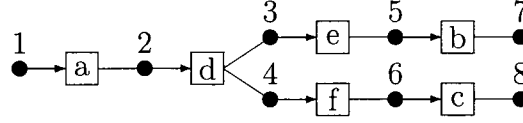
of web grammars. Moreover, every web and directed graph can be realized as a directed plex, making each node of the graph into a NAPE and using the attaching points of the NAPEs to realize the edges of the graph, so web grammars are isomorphic to a subset of plex grammars.

3.4 Flowgraphs

Flowgraphs are a variant of directed plexes where NAPEs (henceforth “napes”) never connect directly to each other via their attachment points, but always indirectly via intermediate tie-points. Every attaching point is connected to exactly one tie-point, but each tie-point is attached to arbitrarily many attaching points. In this dissertation I choose to work with flowgraphs as there is a preexisting algorithm for efficiently parsing them, namely, the one described by Lutz (1996).

A nape in a flowgraph has its attaching points separated into two lists, one for the input attaching points and one for the outputs. When depicted, a nape is shown as a boxed symbol with the connections to the input attaching points coming in from the left and connections from the output points going out to the right. The order of the attaching points is reflected by the vertical order of the connections, with higher connections corresponding to attachment points earlier in their respective list. An example flowgraph is shown in Figure 3.1, with the tie-points being represented as numbered black circles. The figure also shows how a nape can be written as the three-membered list of its symbol and its two lists of tie-points and a flowgraph as the set of all its napes.

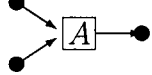
Formal definitions of flowgraphs and their components follow directly. The definitions given here follow Lutz (*ibid.*) in the main, but I have occasionally deviated from and elaborated his definitions where I found it either convenient



$$\{[a,[1],[2]], [d,[2],[3,4]], [e,[3],[5]], [f,[4],[6]], [b,[5],[7]], [c,[6],[8]]\}$$

Figure 3.1: Example flowgraph

or necessary for implementing a flowgraph parser.

- (3.6) **Tie-point** : a tie-point is a pair drawn from $\{\mathbf{true}, \mathbf{false}\} \times \mathbb{N}$. When the first element is **true**, the tie-point is called **instantiated**; when **false** is the first element, the tie-point is called **variable**. For a tie-point with numeric component 3, I write T3 for a variable tie-point and 3 for an instantiated tie-point.
- (3.7) **Alphabet** : an alphabet is an arbitrary set of symbols. The flowgraph in Figure 3.1, for example, makes use of the following alphabet: $\{a, b, c, d, e, f\}$. For flowgraphs depicting Cognitive Representations, the alphabet contains symbols like MOVE, BE_{LOC}, Figure, Ground, etc.
- (3.8) **Nape** : if T is a set of tie-points and Σ is an alphabet, then a T, Σ -nape is a triple $\langle A, I, O \rangle$, where $I, O \in T^*$ are the sequences of **input** and **output tie – points** of the nape, respectively, and $A \in \Sigma$ is the nape's **name**. The triple $\langle A, |I|, |O| \rangle$, where $|s|$ denotes the length of sequence s , is called the nape's **label**. The nape $\langle A, \langle 3, 4 \rangle, \langle T5 \rangle \rangle$, for example, has the label $\langle A, 2, 1 \rangle$, as does the nape $\langle A, \langle 5, T12 \rangle, \langle 13 \rangle \rangle$. This label can be shown graphically as . Additionally, there are functions *inputs*, *outputs*, and *tiepoints* which map a nape to the set of its input tie-points, the set of its output tie-points, and the set of all its tie-points,

respectively.

- (3.9) **Flowgraph** : a flowgraph is a triple $\langle N, T, \Sigma \rangle$, where T is a set of tie-points, Σ is an alphabet, and N is a set of T, Σ -napes.
- a. **input tie-point of a flowgraph** : a tie-point that is an input tie-point of at least one nape in a flowgraph but not an output tie-point of any nape in that flowgraph is called an input tie-point of that flowgraph.
 - b. **input nape of a flowgraph** : a nape which has an input tie-point of a flowgraph as one of its inputs is an **input nape** of that flowgraph. This is what lines 12 and 14 of the algorithm in Figure 3.2 are referring to.
 - c. **output tie-point of a flowgraph** : a tie-point that is an output tie-point for at least one nape but not an input for any nape is an output tie-point of the flowgraph.
 - d. **output nape of a flowgraph** : any nape which is connected to an output tie-point of a flowgraph is an **output nape** of that flowgraph.

In general—since the set of tie-points and the alphabet are likely to be constant across most of the flowgraphs used in a single parsing problem—it is convenient to treat a flowgraph as just the set of napes it contains. Following this convention, if there are two flowgraphs $A = \langle A_N, T, \Sigma \rangle$ and $B = \langle B_N, T, \Sigma \rangle$ then one can write $A \cup B$ for $\langle A_N \cup B_N, T, \Sigma \rangle$, $A \cap B$ for $\langle A_N \cap B_N, T, \Sigma \rangle$, etc.

- (3.10) **Context Free Flowgraph Rewrite Rule** : a context free flowgraph rewrite rule is a pair $\langle n, g \rangle$ where n is a nape and g is a flowgraph.
- a. **left-hand side** : the nape n is called the left-hand side of the

rewrite rule.

- b. **right-hand side** : the flowgraph g is called the right-hand side of the rule.

For any rewrite rule $\langle n, g \rangle$, the tie-points found in n and g are all variable tie-points. Moreover, every input tie-point of n is also an input tie-point of g and every output tie-point of n is also an output tie-point of g . In this way the embedding of g into the host of n is specified.

While the tie-points of n and g are all variable tie-points, the tie-points of any flowgraph where a rewrite rule will be applied are, in contrast, all instantiated tie-points. In this way, there can never be a confusion between the tie-points specified in the rule and the tie-points of the graph that it is operating on. Thus, a context free flowgraph rewrite rule can be applied to any flowgraph containing an appropriately labeled node (i.e., a node labeled in the same as n), no matter where that node appears in the flowgraph.

(3.11) **Context Free Flowgraph Grammar** : a context free flowgraph grammar is a four-tuple $\langle N, T, P, S \rangle$, where N and T are sets of node labels, $S \in N$, and P is a set of context free flowgraph rewrite rules.

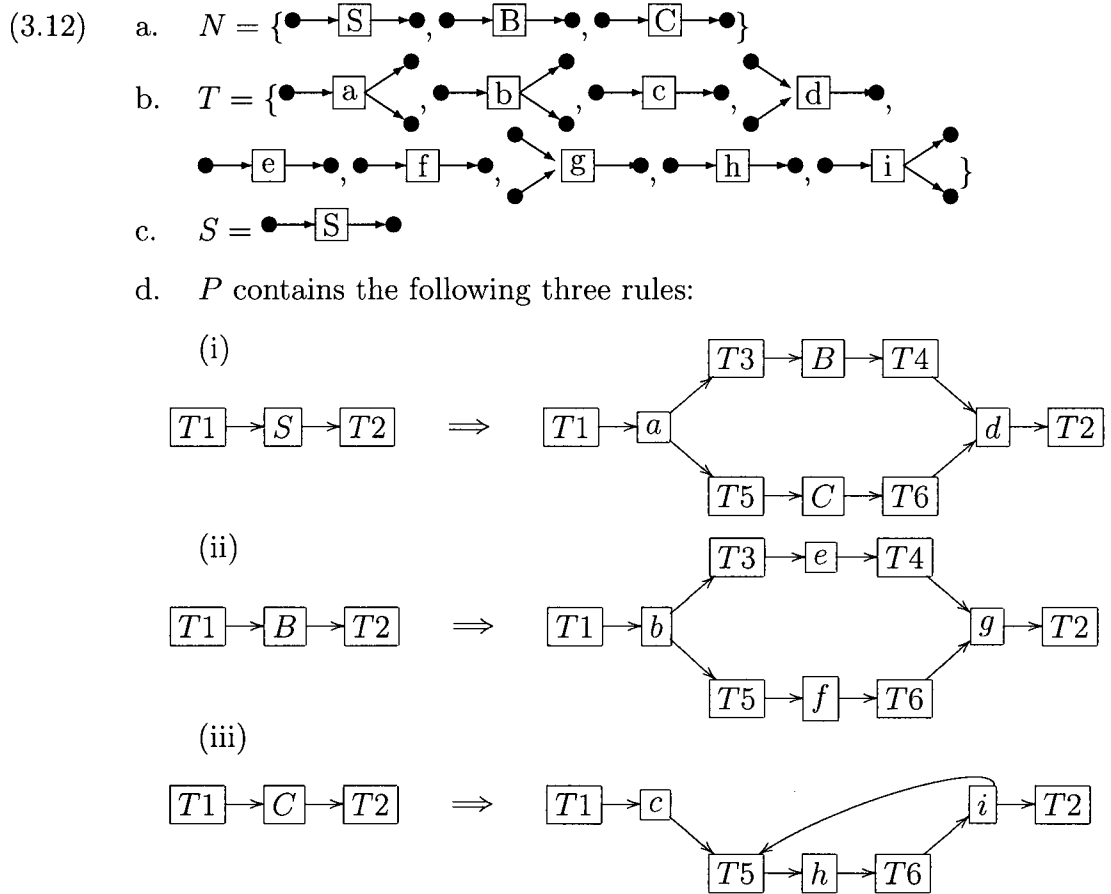
- a. **Non-terminal label** : the elements of N are called the non-terminal labels of a flowgraph grammar.
- b. **Terminal label** : the elements of T are called the terminal labels of a flowgraph grammar.
- c. **Starting label** : the node S is called the starting label of a flowgraph grammar.

Given a grammar $\langle N, T, P, S \rangle$, each rule π in P has as its left-hand side a single

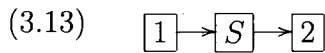
nape labeled by a member of N . Each nape in the right-hand side of π is labeled by a member of $N \cup T$.

3.4.1 Example Flowgraph Derivation

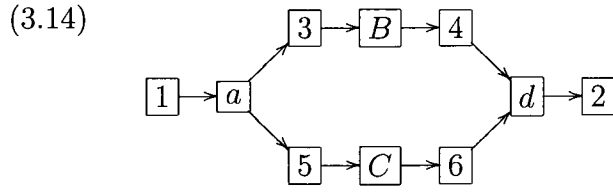
Example (3.12) enumerates the components of a context-free flowgraph grammar that generates exactly one graph: (3.16).



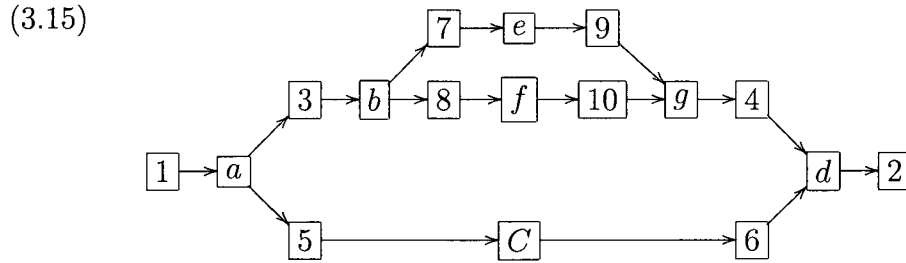
A flowgraph derivation starts with one nape, labeled by the start symbol, as in (3.13).



Any rule with a left-hand side nape labeled $\bullet \rightarrow \boxed{S} \rightarrow \bullet$ can be used to rewrite this graph. In the grammar in (3.12), there is just one, namely (12-d-i). The left-hand side of the rule matches the flowgraph in (3.13) by matching tie-point $T1$ with 1 and $T2$ with 2. New tie-points must be chosen for the instantiations of $T3$, $T4$, $T5$ and $T6$; in (3.14) I choose 3, 4, 5 and 6, although any tie-points not already in the graph would be fine.

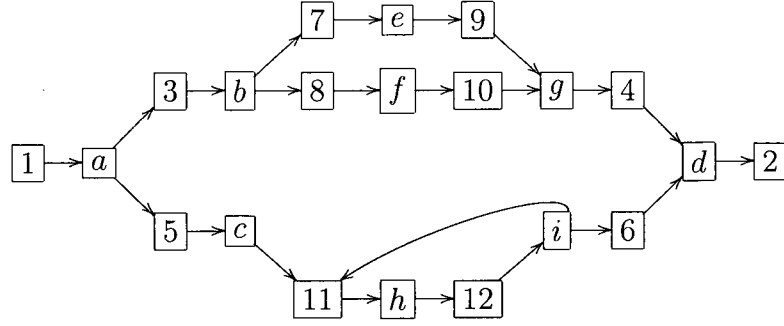


The flowgraph now contains two non-terminally labeled napes, one labeled $\bullet \rightarrow \boxed{B} \rightarrow \bullet$ and one labeled $\bullet \rightarrow \boxed{C} \rightarrow \bullet$. Either one may be rewritten, the former by the rule in (12-d-ii) and the latter by (12-d-iii). Applying (12-d-ii), $T1$ is matched with 3 and $T2$ is matched with 4, and new tie-points are chosen for the remaining tie-points in the left-hand side. The rewritten graph is given in (3.15).



At this point, the only remaining nape with a non-terminal label is the one labeled $\bullet \rightarrow \boxed{C} \rightarrow \bullet$. Rewriting it using (12-d-iii), matching $T1$ with 5 and $T2$ with 6 results in (3.16).

(3.16)



All of the napes in (3.16) are now labeled with terminal labels, so this flowgraph cannot be rewritten any further.

3.5 Converting from Graphs to Flowgraphs

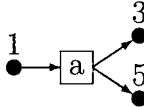
3.5.1 Motivation

Section 3.2 developed graphs as a means of rendering Cognitive Representations, but the algorithm developed in the remainder of this chapter is an algorithm for parsing flowgraphs. To show that this is not a case of bait-and-switch, this section shows how a flowgraph can be obtained from a graph.

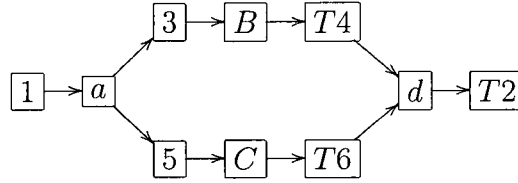
There are two major differences from the graphs in Section 3.2 and flowgraphs. First, there is the addition of tie-points. Their addition is fairly innocent, however, as for the purposes of this dissertation they can be obtained by numbering the edges of the graph in any arbitrary way. The second difference is that the incoming and outgoing edges of a nape are ordered, while those of a graph are not. To arrive at a flowgraph from a graph entails imposing an order on the edges connecting to each node of the graph, and the order must be consistent for each node with the same label. This is worrisome, as the lack of ordering among the constituents of a Cognitive Representation is one of the reasons why graphs

are an attractive means of formalizing them. It is important to keep in mind, then, that the choice of flowgraphs for parsing is a choice made freely among a large array of available graph grammars; a choice made primarily because the algorithm for parsing flowgraphs is both easy to understand and to implement as well as highly efficient.

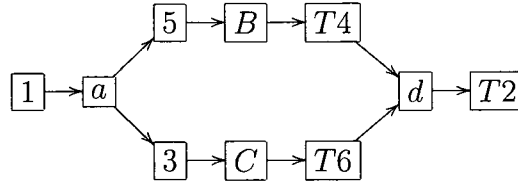
A large part of the efficiency of flowgraph parsing derives from the ordering of a napes input and output tie-points. Parsing proceeds by matching the napes in the right-hand side of a rewrite rule against a subset of the napes in the flowgraph being parsed. The first criterion for the matching is that each nape in the rewrite rule has a corresponding nape with the same label in the graph being parsed. The second criterion is that it must be possible to instantiate the variable tie-points of the rewrite rule as the tie-points in the target flowgraph. By giving the inputs and outputs of each nape an order, there is only one way to try to instantiate any variable tie-points: the first input of one nape to the first input of its correspondent, the second to the second, etc. If no order were given, then all possible instantiations (of which there can be factorially many) would have to be tried. Having ordered input and output tie-points lowers the complexity of nape-to-nape matching from $O(n!)$ to $O(1)$, where n is higher of the cardinalities of the input and output sequences.

The local simplification in nape-to-nape matching has, however, only a limited effect on the global complexity of parsing the graph. When using the grammar in (3.12) to parse (3.16), the nape  indicates that there may be an instantiation of the rule in (12-d-i). Without ordered tie-points, there are two possibilities to consider:

(3.17) a.



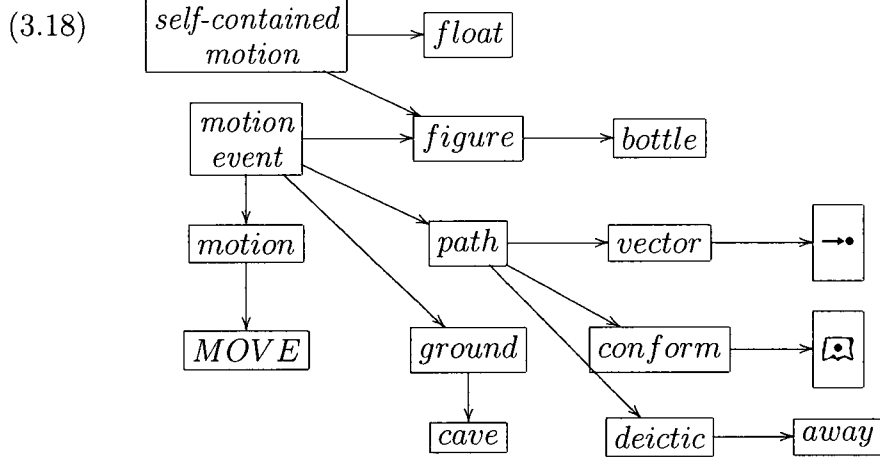
b.



There is no need to investigate either of these possibilities any further, though, until the parser finds evidence of an expansion of $\bullet \rightarrow \boxed{B} \rightarrow \bullet$ or $\bullet \rightarrow \boxed{C} \rightarrow \bullet$. The parser will eventually find a evidence for $\overset{3}{\bullet} \rightarrow \boxed{B} \rightarrow \overset{4}{\bullet}$ and will then investigate (3.17-a), but it will never find any reason to do any more work with (3.17-b).

3.5.2 Process

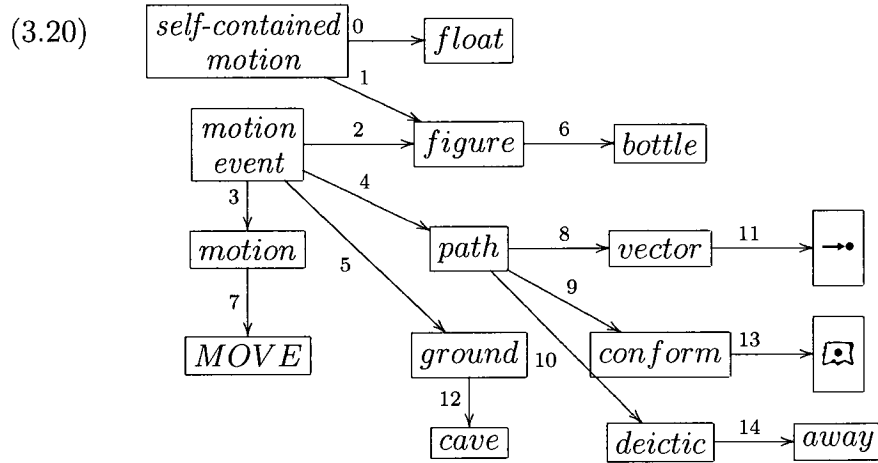
The process of obtaining a flowgraph from a directed graph involves three steps. First, the edges must be numbered. Second, tie-points must be constructed from each edge. Finally, a nape must be constructed from each node in the graph using these tie-points. A specification of the process follows, using the graph in (3.3), repeated as (3.18).



For the purposes of obtaining a flowgraph, it will be useful to consider a graph G as a four-tuple $\langle V, E, \Sigma, L \rangle$, where V is the set of vertices in G , $E \subseteq V \times V$ is the set of edges in G , Σ is an alphabet, and L is a labeling function with domain V and range Σ . The graph in (3.18), then, is represented in (3.19):

- (3.19) $G = \langle V, A, \Sigma, L \rangle$
- $\Sigma \supseteq \{\text{self-contained motion, float, motion event, figure, bottle, motion, path, vector, } \rightarrow\bullet, \text{ MOVE, ground, conform, } \text{📷}, \text{ cave, deictic, away}\}$
 - $V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p\}$
 - $L = \{\langle a, \text{self-contained motion} \rangle, \langle b, \text{float} \rangle, \langle c, \text{motion-event} \rangle, \langle d, \text{figure} \rangle, \langle e, \text{bottle} \rangle, \langle f, \text{motion} \rangle, \langle g, \text{path} \rangle, \langle h, \text{vector} \rangle, \langle i, \rightarrow\bullet \rangle, \langle j, \text{MOVE} \rangle, \langle k, \text{ground} \rangle, \langle l, \text{conform} \rangle, \langle m, \text{📷} \rangle, \langle n, \text{cave} \rangle, \langle o, \text{deictic} \rangle, \langle p, \text{away} \rangle\}$
 - $E = \{\langle a, b \rangle, \langle a, d \rangle, \langle c, d \rangle, \langle c, f \rangle, \langle c, g \rangle, \langle c, k \rangle, \langle d, e \rangle, \langle f, j \rangle, \langle g, h \rangle, \langle g, l \rangle, \langle g, o \rangle, \langle h, i \rangle, \langle k, n \rangle, \langle l, m \rangle, \langle o, p \rangle\}$

Since E in (3.19-d) is a set with 15 members, there are $15!$ bijections between E and the first 15 natural numbers. One such mapping—let it be known as h —is shown in (3.20).



A flowgraph is a set of napes that share an alphabet and a set of tie-points. In creating a flowgraph F from G , the alphabet Σ of the graph can be reused as-is in the flowgraph. The tie-points are easily derived from the number of the edges. Let T be the set of all pairs $\langle n, \mathbf{true} \rangle$ for each n that h maps to some element of E .

Now that T and Σ are determined, F is just a set of T, Σ -napes, one for each node in the graph. Let f be a function that maps each vertex γ in V to a distinct T, Σ nape ν and that meets the conditions in (3.21), which guarantee that the name of ν is γ 's label, that input tie-points of ν are determined by the incoming arcs of γ , and that the outgoing arcs of γ determine the output tie-points of ν . (As in Section 3.3, the notation I_k indicates the k^{th} element of I .) F , then, is the set $\{f(\gamma) | \gamma \in V\}$.

$$(3.21) \quad \text{For } \gamma \in V, \nu = f(\gamma) = \langle L(\gamma), I, O \rangle$$

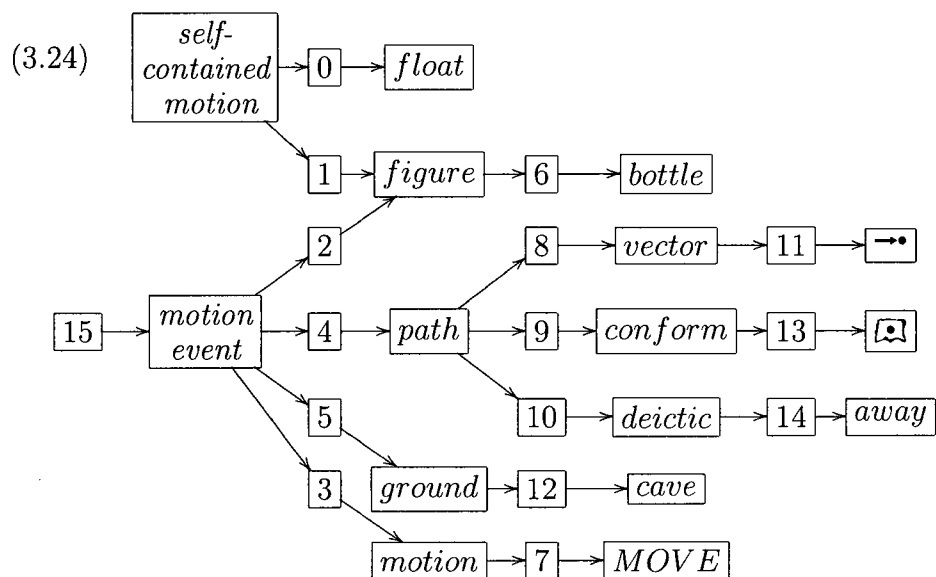
- a. $\{\tau : I_k = \tau, 1 \leq k \leq |I|\} = \{\langle n, \mathbf{true} \rangle : \alpha \in E, h(\alpha) = n, \alpha_2 = \gamma\}$
- b. $\{\tau : O_k = \tau, 1 \leq k \leq |O|\} = \{\langle n, \mathbf{true} \rangle : \alpha \in E, h(\alpha) = n, \alpha_1 = \gamma\}$

What (3.21) leaves undefined is the order of elements in I and O . For most vertices in a Cognitive Representation graph, there is at most one input arc and one output arc, so the order is trivial. There are four exceptions to this, however. A vertex labeled by *motion event* has four outgoing arcs, one labeled *path* has three outgoing arcs, one labeled *self-contained motion* has two outgoing arcs, and one labeled *figure* may have either one or two incoming arcs. In these cases, the order must simply be conventionalized.

- (3.22)
- a. For a Motion Event vertex, the arc going to the Figure is ordered first, followed by the arcs going to the Path, the Ground, and lastly the Motion.
 - b. For a Path vertex, the first out-going arc is the one connecting to the Vector; the second connects to the Conform; and the third connects to the Deictic.
 - c. For a Self-Contained Motion (Manner) node, the arc leading to the type of SCM is ordered first.
 - d. For a Figure vertex, if there is an incoming arc from a Manner, then it is ordered first.

- (3.23) $\{\text{[self-contained motion, [], [0,1]], [float, [0], []], [motion event, [2,4,5,3], []], [figure, [1,2], [6]], [bottle, [6], []], [motion, [3], [7]], [path, [4], [8,9,10]], [vector, [8], [11]], [\rightarrow\bullet, [11], []], [\text{MOVE}, [3], [7]], [\text{ground}, [5], [12]], [\text{conform}, [9], [13]], [\text{cave}, [13], []], [\text{cave}, [12], []], [\text{deictic}, [10], [14]], [\text{away}, [14], []]\}$

There is one final concession to be made to flowgraph parsing. A flowgraph should have an input nape, but (3.23) has none. Since the goal of this dissertation is to generate sentences about motion events, the Motion Event nape is the natural choice. So, let $\langle 15, \mathbf{true} \rangle$ be added to its input tie-points as well as to T . The final flowgraph is given as (3.24).

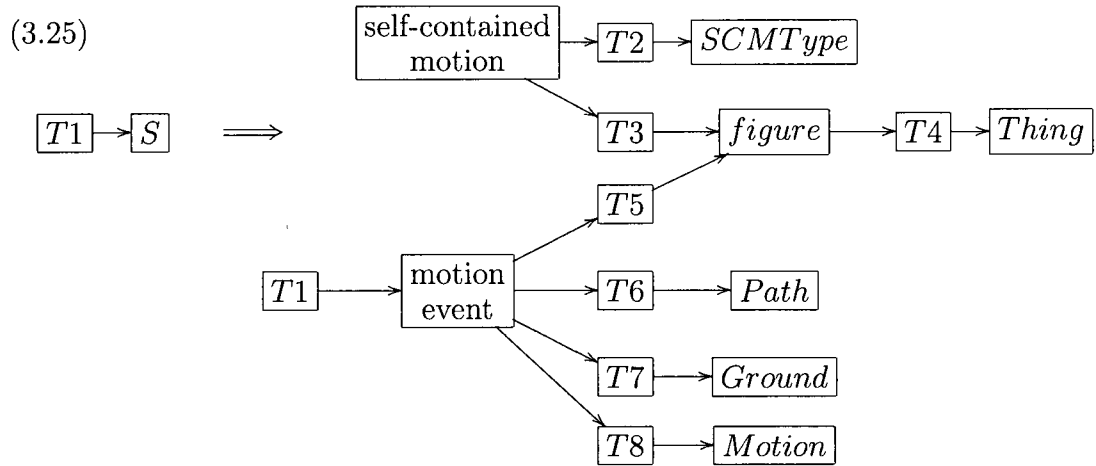


3.6 Parsing with Context Free Flowgraph Grammars

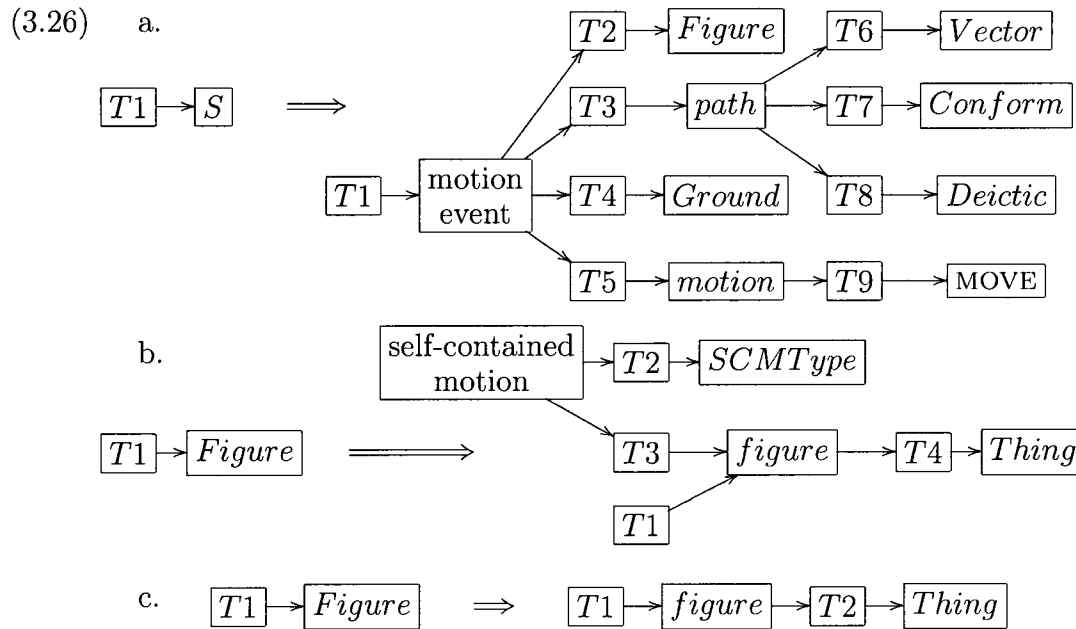
3.6.1 Designing a Flowgraph Grammar for Cognitive Representations

Using (3.24) as an example of what cognitive representations are like when rendered as flowgraphs, I shall demonstrate how the graphs are parsed. The graph in (3.3) illustrates a reasonable rendering of the cognitive representation established by the original observations from (2.1), which an English speaker might report as (2.2): “the bottle floated into the cave.” This particular English sentence, as is usual for English reports of motion events, introduces both the fact of motion and the manner of motion simultaneously, a graph grammar for generating English

expressions should reflect this tendency; the rule in (3.25) is designed to do so:

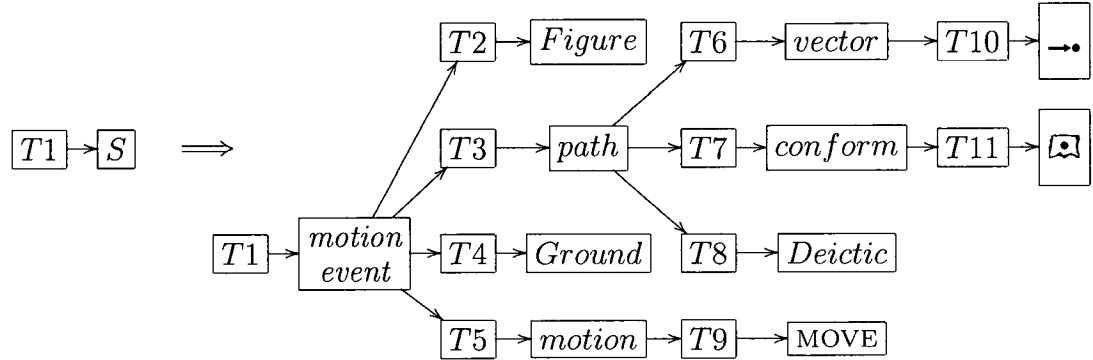


A graph grammar for generating Spanish expressions, on the other hand, should reflect the Spanish tendency to introduce fact of motion together with some aspect of the path of the motion (3.26-a), as well as the fact the manner of motion may be given or not; the latter expressed as the alternative expansions of *Figure*, (3.26-b) and (3.26-c).



In every language there are words that have an idiosyncratic behavior, not following the major pattern of the Language. English ‘enter’, for example, is a Motion+Path verb. Graph grammars for producing a language can reflect the non-general behavior of these words by associating their production with idiosyncratic rules in the graph grammar.

(3.27) English graph grammar rule corresponding to use of ‘enter’:



3.6.2 Patches

Bottom-up parsing of a flowgraph involves searching the flowgraph for connected groups of napes that could be instantiations of the right-hand side of a rule in a grammar. Lutz (1996) calls these groups **patches**, the idea being that a graph parsed, like a quilt, is complete when it is covered in patches. There are two kinds of patches, complete and partial. Complete patches indicate that the entire right-hand side of a rule has been found in the flowgraph being parsed. A partial patch indicates that at least part of a rule’s right-hand side has been found, and has a specification of which parts of the right-hand side should be searched for next. Formal definitions (adaptated from Lutz (ibid.)) of complete and partial patches are given in (3.28) and (3.29).

(3.28) **Complete Patch** : if n is a nape, then the pair $\langle n, \emptyset \rangle$ is a complete

patch; if n is a nape and C is a set of complete patches, then the pair $\langle n, C \rangle$ is also a complete patch, and the elements of C are called the **components** of the patch. If a complete patch has no components, then its nape is one that occurs in the flowgraph being parsed (that is, a nape with a terminal label). If C is not empty, then C is a set of napes that is the instantiation of the right-hand side of a rule and n is the nape that stands on the left-hand side of that same rule, instantiated by the same assignment as the napes in C . The functions *label*, *inputs*, *outputs*, and *tiepoints* are extended from their definitions over napes to complete patches, mapping a complete patch to whatever they match the patch's nape to.

It is important to note here that a flowgraph $\langle N, T, \Sigma \rangle$ uniquely determines a set of complete patches $\{\langle n, \emptyset \rangle \mid n \in N\}$. These complete patches are the starting points of the flowgraph parsing process; the first partial patches found in a bottom-up parse will be built from them.

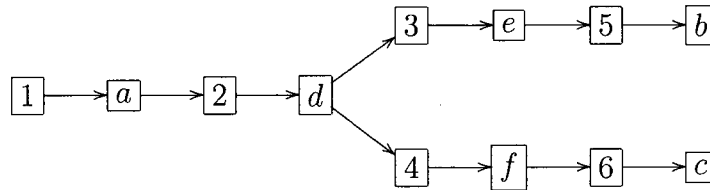
- (3.29) **Partial Patch** : a partial patch is a 5-tuple $\langle n, C, A_i, A_o, N \rangle$, where n is a nape, an instantiation of the left-hand side of a rule; C is a set of complete patches, the components; A_i is a set of tie-points drawn from the input tie-points of n and called the **active inputs**; A_o is a set of tie-points drawn from the output tie-points of n and called the **active outputs**; and N is a flowgraph containing the **needed napes**, those napes which are required to complete the right-hand side of a rule and which are not components of the patch. A partial patch **immediately needs** one of its needed napes just in case one of the active outputs of the patch is an input to the needed nape or one of the active inputs of

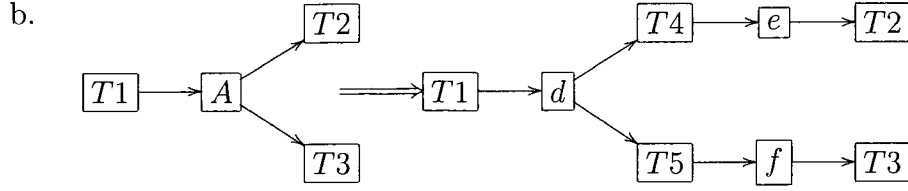
the patch is an output of the needed nape. As with complete patches, the functions *label*, *inputs*, and *outputs* map a partial patch to the same value as the patch's nape. The *tiepoints* function maps a partial patch to the union of the tie-points of its components. So, if B_C is the components of a partial patch B , then $tiepoints(B)$ is:

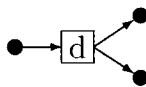
$$\bigcup_{c \in B_C} tiepoints(c)$$

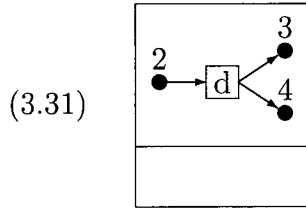
The tie-points in the flowgraph being parsed and the tie-points of a complete patch are all instantiated (i.e., they are $\langle \mathbf{true}, n \rangle$ for some $n \in \mathbb{N}$). When a new partial patch is created, however, all of its tie-points are uninstantiated. As a flowgraph parse proceeds, the parser matches a immediately needed nape of a partial patch with a complete patch having the same label (and so also the same number of input and output tie-points). If there are no conflicts between the tie-points immediately needed nape and of the complete path—that is, all of the instantiated tie-points of the nape match the tie-points of the complete patch, and there is a mapping from the variable tie-points of the nape to those in the complete patch—then a new parch is created by applying the mapping, adding the complete patch to the components of the new patch and removing the nape from its needed napes.

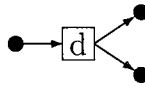
(3.30) a.

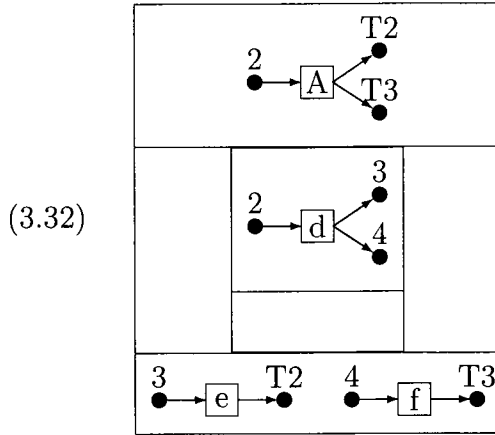




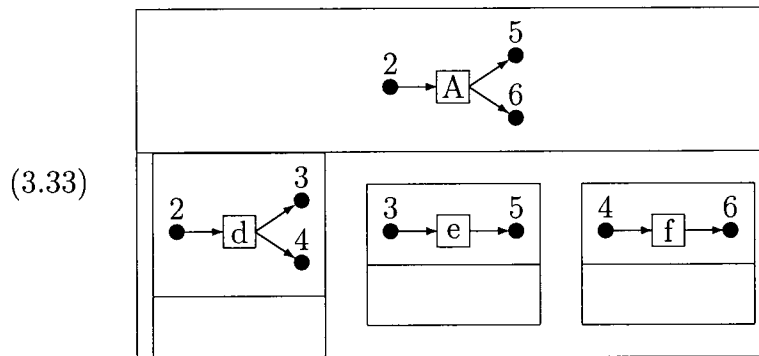
The flowgraph in (3.30-a) and the rule in (3.30-b) provide some useful examples. In the flowgraph there is a nape with the symbol d , the input tie-point 2, and two output tie-points 3 and 4. The *label* of this nape, then, is $\langle d, 1, 2 \rangle$, which—as mentioned earlier—can be written more visually as . This nape also corresponds to a complete patch with no components, represented by the figure in (3.31):



The right-hand side of (3.30-b) also has an input nape with the label . The input nape in the rule and the complete patch in (3.31) are sufficient for a parser to record a partial patch with one component (i. e., (3.31)) and two needed napes, $\overset{3}{\bullet} \rightarrow \boxed{e} \rightarrow \bullet$ and $\overset{4}{\bullet} \rightarrow \boxed{f} \rightarrow \bullet$. The needed napes have their input tie-points instantiated but their output tie-points are still variable, meaning that, for example, any partial patch with label $\bullet \rightarrow \boxed{e} \rightarrow \bullet$ and input tie-point 3 can extend the partial patch independent of its output tie-point.



Once the parser has verified that there are completed patches satisfying the two needed napes, it can record a complete patch for $\bullet \rightarrow A \begin{matrix} \nearrow \\ \searrow \end{matrix}$, (3.33):



3.6.3 Lutz's Algorithm

Lutz (1996) provides the algorithm in Figure 3.2 for bottom-up parsing graphs of the kind described in the previous section. The algorithm works on *patches* (also called “covering patches”). A patch is a statement that a terminal or non-terminal symbol from the grammar has been found in the graph. In the case of a non-terminal symbol, it may be the case that only part of its expansion has been found. In this case the patch is called *partial*. If the symbol is a terminal, or if an entire expansion of it has been found, the patch is called *complete*.

A patch is labeled by the symbol from the grammar which has been found and contains a graph with input and output tie-points where it (the patch) connects to the rest of the graph, as well as the set of patches that have been used to build it. For a patch generated from a terminal symbol the list of component patches will always be empty. An incomplete patch will also have a record of what kind patches need to be found to complete it.

The algorithm is driven by an agenda (a list of patches that need investigation) and a chart which stores information about patches that have already been investigated. When a patch is drawn from the agenda, the algorithm checks the chart to see if the patch has already been investigated and, if it has, ignores the patch. Otherwise, the patch is added to the chart and investigated.

When the patch drawn from the agenda is a partial patch, the chart is checked for any complete patches that might fill in the missing parts of its expansion. Each suitable complete patch is used to create a new patch starting from the partial patch and filling in the corresponding hole in the expansion, and each new patch is then added to the agenda.

If the patch drawn from the agenda is a complete patch, then a similar search is done in the chart for partial patches whose needs might be met by the complete patch, and the suitable partial patches are used to create new patches to add to the agenda. Additionally, each rule in the grammar is checked to see if its expansions contains a peripheral nape with the same label as the complete patch. New partial patches are created for each suitable rule and added to the agenda.

When the agenda is empty, the algorithm is finished. The chart ensures that no work is duplicated. The exhaustive search of the chart for each new complete and partial patch ensures that no work is skipped. After the algorithm has run its course, any complete patch in the chart which is labeled by a start symbol is


```

ParseGraph()
1  INITIALIZE chart AND agenda
2  repeat
3       $A \leftarrow$  PICK PATCH FROM agenda
4      if  $\neg(\text{chart CONTAINS } A)$ 
5          then ADD  $A$  TO chart
6          if  $A$  IS COMPLETE
7              then for each partial patch  $B$  in chart
8                  where  $A$  CAN EXTEND  $B$ 
9                  do  $\alpha \leftarrow B$  EXTENDED WITH  $A$ 
10                     PUT  $\alpha$  ON agenda
11                 for each rule  $R$  in  $P$ 
12                     where RHS( $R$ ) HAS INPUT NAPE LABELED BY
13                     LABEL( $A$ )
14                     do for each input nape  $X$  in  $R$ 
15                         where LABEL( $X$ ) = LABEL( $A$ )
16                         do  $\beta \leftarrow$  NEW EMPTY PATCH
17                         LABEL( $\beta$ )  $\leftarrow$  LHS( $R$ )
18                         NEEDED( $\beta$ )  $\leftarrow$  RHS( $R$ )
19                         INSTANTIATE  $\beta$  w.r.t.  $X$  and  $A$ 
20                         INPUTS( $\beta$ )  $\leftarrow$  INPUTS( $A$ )
21                         ACTIVE-OUTS( $\beta$ )  $\leftarrow$  INPUTS( $A$ )
22                         PUT  $\beta$  ON agenda
23                 else for each complete patch  $B$  in chart
24                     where  $B$  CAN EXTEND  $A$ 
25                     do  $\alpha \leftarrow A$  EXTENDED WITH  $B$ 
26                     PUT  $\alpha$  ON agenda
27  until agenda IS EMPTY

```

Figure 3.2: A sketch of Lutz's (1996, p. 367) parsing algorithm, simplified for only bottom-up parsing

a successful parse of the graph using the grammar.

3.6.4 Implementing Lutz's Algorithm

Implementing the algorithm in Figure 3.2 requires the choice of data structures for the various elements involves as well as the definition of equivalence relations for each kind of patch. Also, the extension of a partial patch by a complete patch is a non-trivial operation, the details of which are not specified by Lutz (1996). The following sections give more explicit definitions and specify how certain steps of the algorithm are implemented for this dissertation.

3.6.5 Equivalence of Patches

The behavior of the algorithm depends profoundly on what it means to say that two patches are the same. The difference comes about in line 4 of the algorithm, where the chart is checked for the presence of the patch just drawn from the agenda. Changing the criteria for being “the same patch” changes which patches will be investigated and which patches will not.

The mathematical definitions in (3.28) and (3.29) each give a clear candidate for an equivalence relation, namely, that two patches are the same just in case they are identical at each element in their tuple. When this criterion is used for sameness, then all possible parses will be found.

An alternative criterion for sameness relaxes the strict equivalence requirement, allowing two patches to count as the same even if they have different components. Using this criterion, the algorithm will always find a parse if there is one, but may fail to find many alternative parses. The advantage is that this criterion allows the algorithm to investigate far fewer patches. Lutz (*ibid.*) explains,

“...for some flowgraphs and some grammars there may well be an exponential number of parses (this is even true of Earley’s algorithm operating on strings!)”

Because this thesis revolves around considering all of the parses for a particular graph and not around the efficiency of the computation, my implementation uses the strict equivalence.

3.6.6 Extension of Partial Patches by Complete Patches

Two parts of the algorithm—lines 8–9 and 24–25—deal with complete patches **extending** partial patches. A complete patch $A = \langle A_n, A_C \rangle$ can extend a partial patch $B = \langle B_n, B_C, B_{A_i}, B_{A_o}, B_N \rangle$ if it matches the following conditions:

- (3.34) a. there is a nape ν such that B immediately needs ν and $label(A) = label(\nu)$
- b. there is an assignment α under which A is the instantiation of ν .

If B needs just one more nape (i.e, if $B_N = \{\nu\}$), then the extension of B by a A is the complete patch with the nape $\alpha(B_n)$ and the components $B_C \cup \{A\}$. If, however, B needs more than one patch to fill out the right-hand side of its originating rule, then the extension of B by A is another partial patch $P = \langle \alpha(B_n), B_C \cup \{A\}, P_{A_i}, P_{A_o}, P_N \rangle$. The active tie-points of P are the active tie-points of B with the additions and subtractions in (3.35), while the P ’s flowgraph of needed napes is the instantiation of B ’s needed flowgraph without the instantiation of ν . The equations in (3.36) summarize the result of the extension.

- (3.35) a. the inputs of A are not active outputs of P
- b. the outputs of A are not active inputs of P
- c. an input τ of A is an active input of P just in case τ is not an input

- of B_n nor an input or output of any nape in B_C
- d. an output τ of A is an active output of P just in case τ is not an output of B_n nor an input or output of any nape in B_C
- (3.36) a. $P_{A_i} = (B_{A_i} - \text{outputs}(A)) \cup (\text{inputs}(A) - (\text{tiepoints}(B) \cup \text{inputs}(B_n)))$
- b. $P_{A_o} = (B_{A_o} - \text{inputs}(A)) \cup (\text{outputs}(A) - (\text{tiepoints}(B) \cup \text{outputs}(B_n)))$
- c. $P_N = \alpha(B_N) - \alpha(\nu)$

3.6.6.1 The Chart

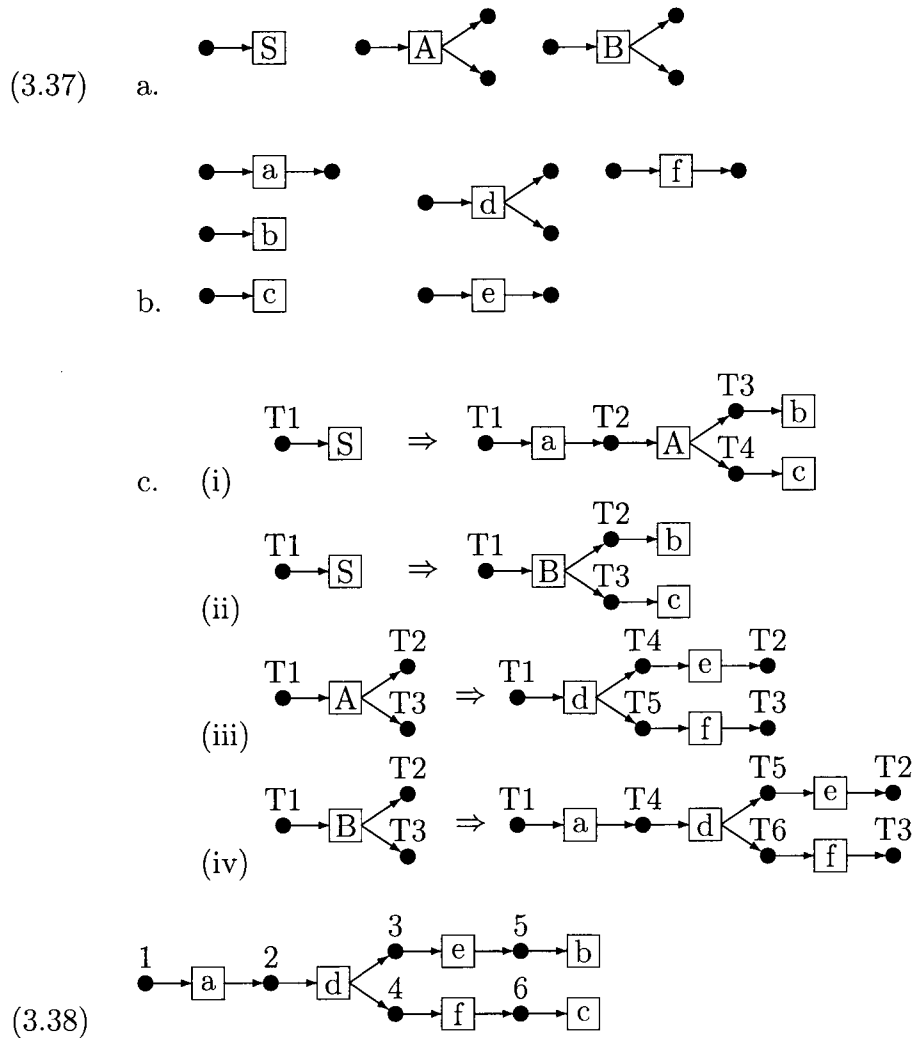
Lutz's algorithm stores the known complete and partial patches in a chart to facilitate the quick retrieval of partial patches that can be extended by a given complete patch and of complete patches that can extend a given partial patch. To this end, the chart is divided into four tables: two for the partial patches and two for the complete ones. Each tables has two axes, the tie-points along one and the labels along the other.

The tables for the complete patches are simple. When a complete patch is entered into the chart, an entry is made in the chart for each of its input and output tie-points. The input tie-point based entries go into one table and the output tie-point based entries go into the other.

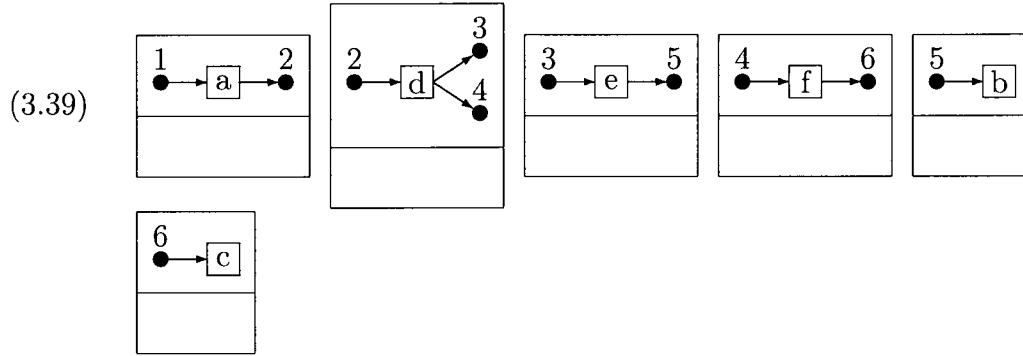
The entries for the partial patches are a little more involved. For each immediately needed nape in the partial patch an entry is made under the immediately needed nape's label at each active input and output tie-point. As with the complete patches, the entries are separated between the two tables based on whether they are derived from an input or an output.

3.7 An Example Parse

The discussion of context-free flowgraph grammars in (3.11) provides a graph and a grammar that may serve for an example run of Lutz's algorithm. The grammar has the three non-terminal napes in (3.37-a), the six terminal napes in (3.37-b), the four production rules in (3.37-c), and the designated start symbol S. The language generated by the grammar in (3.37) is the singleton set consisting of the graph in (3.38), which it can generate in two different ways.

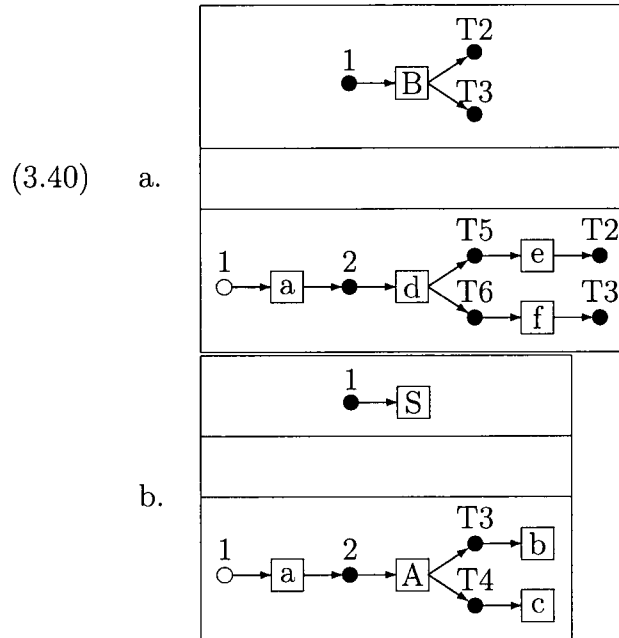


The first step of the algorithm is to initialize the agenda. Each nape in (3.38) is made into a complete patch and added to the agenda. Displaying complete patches as a pair of stacked boxes with the nape in the top box and the components in the bottom box, the structures in (3.39) display the agenda after initialization.

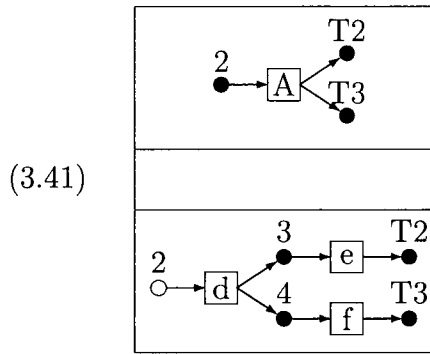


Once the agenda has been initialized, patches are drawn from it. In this example, suppose that the first patch in (3.39) is drawn first. The parser checks its chart—which is empty initially—and finds that the patch is not there. Accordingly, the patch is entered into the chart under (1,a) for complete-patches-by-input and under (2,a) for complete-patches-by-output. The chart is then checked for any partial patches that could be extended with the complete patch. Next, the parser checks the rules for right-hand sides with input napes labeled as $\bullet \rightarrow [a] \rightarrow \bullet$ and finds that the rule for expanding $\bullet \rightarrow [B] \rightarrow \bullet$ meets this condition. The assignment $[T1 \mapsto 1, T4 \mapsto 2]$, which instantiates the input nape of the rule as the nape from the complete patch, is then used to create the new partial patch shown in (3.40-a), where the top two boxes are as in the display for complete patches and the bottom box represents the needed components and the active tie-points are displayed with non-filled circles. The first rule for expanding $\bullet \rightarrow [S] \rightarrow \bullet$ also has an input nape labeled $\bullet \rightarrow [a] \rightarrow \bullet$, so the parser also creates the new partial patch

in (3.40-b). The parser adds (3.40-a) and (3.40-b) to the agenda and then draws from the agenda again.



The second draw from the agenda gives the parser the second patch in (3.39), the one labeled $\bullet \rightarrow [d] \begin{matrix} \nearrow \\ \searrow \end{matrix}$. Since this patch is also not already in the chart, it is added under (2,d) for its input and under (3,d) and (4,d) for its outputs. There are still no partial patches in the chart, so there is nothing for this patch to extend, but a check against the rules reveals that is potentially an input nape for an expansion of $\bullet \rightarrow [A] \begin{matrix} \nearrow \\ \searrow \end{matrix}$. Using the assignment $[T1 \mapsto 2, T4 \mapsto 3, T5 \mapsto 4]$, another new partial patch (3.41) is created and added to the agenda.



None of the remaining complete patches on the agenda can cause the creation of new partial patches, so each is simply added to the chart in turn. This leaves the agenda with just the partial patches in (3.40) and (3.41) and the chart with the entries shown in Tables 3.2 and 3.3.

1						
2						
3						
4						
5						
6						

Table 3.2: Complete patches by input

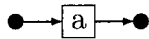
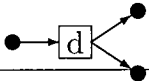
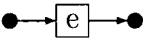
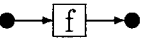
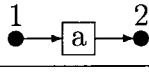
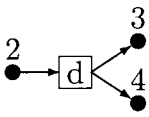
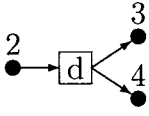
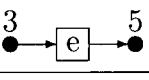
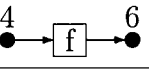
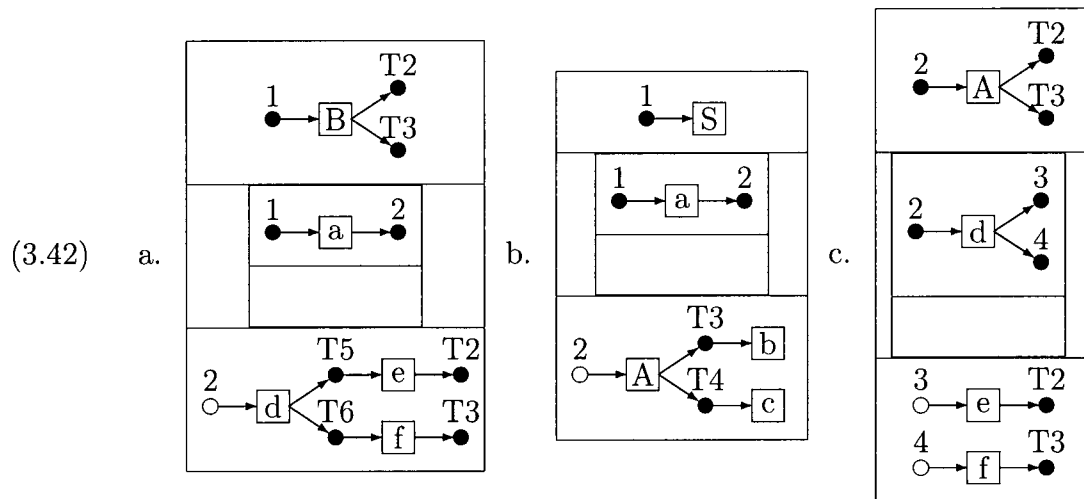
				
2				
3				
4				
5				
6				

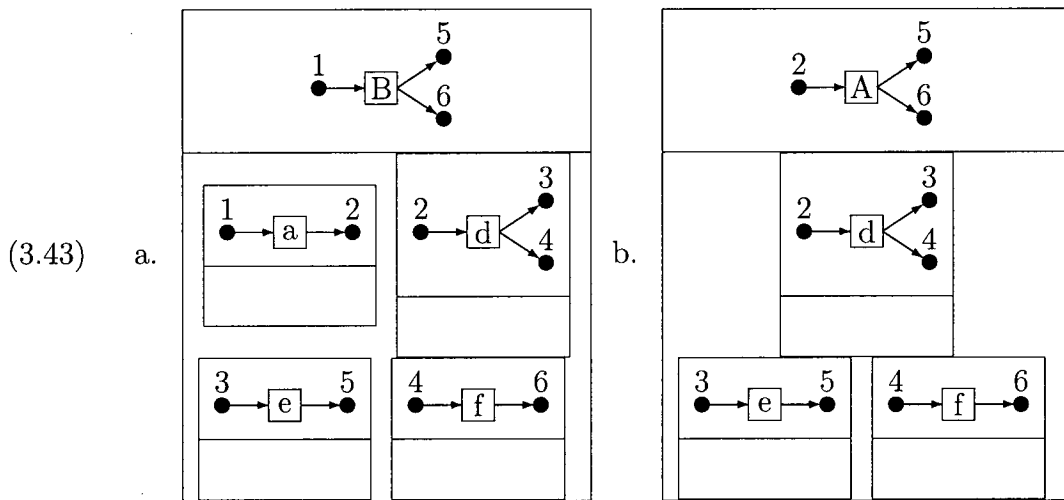
Table 3.3: Complete patches by output

The next patch that the parser draws from the agenda is the one shown in (3.40-a). The parser confirms that this patch is not already in the chart, and begins to process it. It immediately needs a patch that is labeled by $\bullet \rightarrow \boxed{a} \rightarrow \bullet$ and has the tie-point 1 as an input. This has two consequences. First, the partial patch is entered into the partial-patches-by-active-output table of the chart at the intersection of $\bullet \rightarrow \boxed{a} \rightarrow \bullet$ and 1. Second, the parser checks the complete-patches-by-input table in its chart (i.e., Table 3.2) at the same pair of coordinates. In this table it finds the complete patch that can extend the partial patch. The resulting new partial patch, shown as (3.42-a), is then added to the agenda. Similar operations on the patches in (3.40-b) and (3.40) produce the new patches in (3.42-b) and (3.42-c) for the agenda.

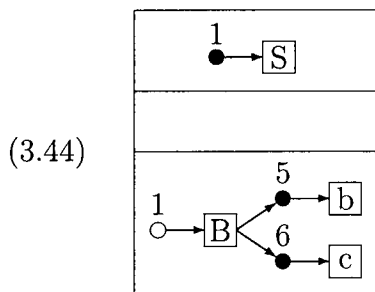


The parser deals with (3.42-a) almost identically to (3.41)—it has the same needed graph and the same active outputs—and there are no complete patches in the chart that can extend (3.42-b) yet, so no new patches can be created from it. When (3.42-c) is drawn from the agenda, however, the parser finds two complete patches that can extend it, so two new partial patches are created: one extended by the complete patch labeled $\bullet \rightarrow \boxed{e} \rightarrow \bullet$ and the other extended by

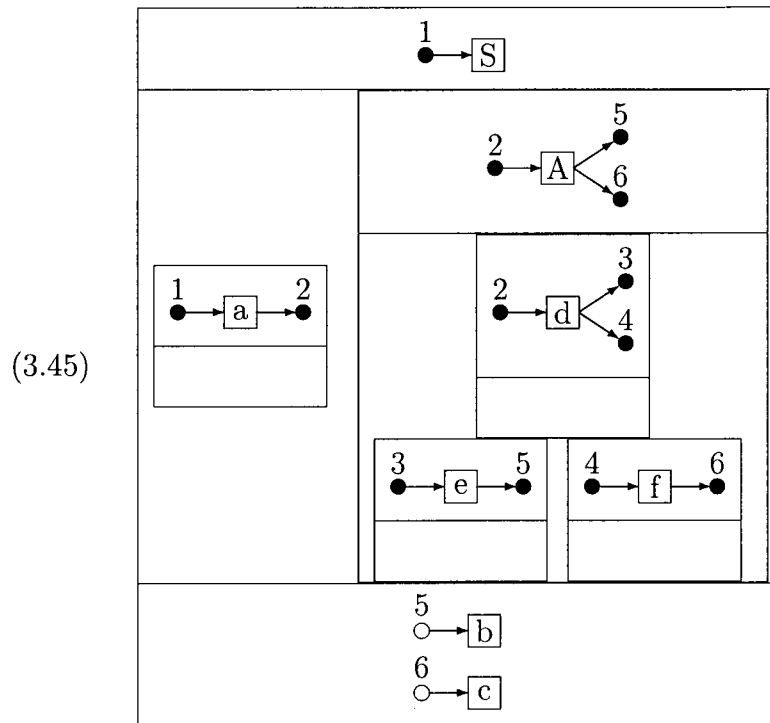
the one labeled by $\bullet \rightarrow \boxed{f} \rightarrow \bullet$. Each of the new partial patches, when drawn from the agenda, is extended by the other patch, so the patch that resulted from extending (3.42-c) by the patch labeled $\bullet \rightarrow \boxed{e} \rightarrow \bullet$ is extended by the patch labeled $\bullet \rightarrow \boxed{f} \rightarrow \bullet$ and vice-versa. Both of these further extensions create the complete patch in (3.43-b). The constant check for patches already in the chart means that (3.43-b) is skipped the second time it is drawn from the agenda. The descendants of (3.42-a) create the complete patch in (3.43-a) via two different paths in a completely analogous fashion.



When (3.43-a) is drawn from the agenda the parser finds no complete patches for it to extend but it does find that the second rule for expanding $\bullet \rightarrow \boxed{S}$ has a matching input nape. The new partial patch for this match is shown as (3.44-a).



The other complete patch, (3.43-b), does not match an input nape of any rule but can extend the $\bullet \rightarrow \boxed{S}$ patch in (3.42). The patch shown in (3.45) is the result of this extension.



The steps from (3.45) to a complete patch for $\bullet \rightarrow \boxed{S}$ involve extending it twice once with the $\bullet \rightarrow \boxed{b}$ or $\bullet \rightarrow \boxed{c}$ patch from the chart and then extending the result with the other. (3.44) will also lead to a complete patch, extended first with the complete $\bullet \rightarrow \boxed{B}$ patch and then enjoying the same fate as (3.45). So, the Lutz algorithm, given the right specification for patch equality, finds both parses available for the flowgraph and the grammar.

3.8 Conclusions

A string grammar determines a set of strings in string language. A graph grammar works in the same way to determine a set of graphs. But grammars don't just determine sets, they also provide a recipe for constructing each of their members. For some members of a set, a grammar may offer multiple recipes, which leads to an ambiguity in parsing.

In this chapter, I offered a grammar for graphs rendering Cognitive Representations using rules that match the patterns of conflation commonly found in lexical items—such as a rule simultaneously introducing a Motion Event and a Co-Event with the same Figure, paralleling the common conflation of fact-of-motion and manner-of-motion. Parsing ambiguities induced by this grammar, then, reflect alternate patterns of conflation that still evoke the same CR. In Chapter 4 I show how different parses are used to generate different sentences, and in Appendix A I offer a more complete grammar for Cognitive Representations of Motion Events.

CHAPTER 4

From Graph Parses to Linguistic Expressions

4.1 Overview

This is the chapter where the generation actually happens. It builds upon all the other parts of the dissertation.

In the introduction to this dissertation, I set out a goal: to generate sentences which, like (4.1), evoke a particular Cognitive Representation of a motion event.

(4.1) The bottle entered the cave floating.

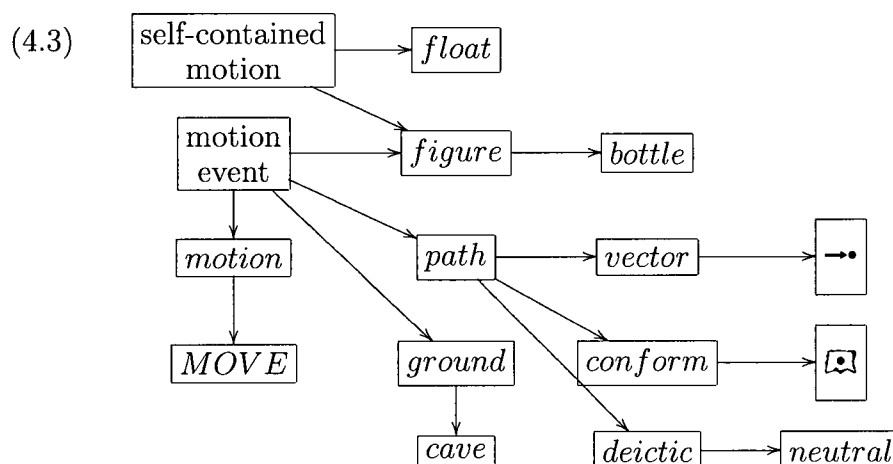
In Chapter 2, I explained what Cognitive Representations are, laying out their rough structure in terms of Motion, Path, Figure, Ground, and Co-Events. In the case of the sentence in (4.1), the components are those in (4.2).

- (4.2)
- a. Motion: MOVE
 - b. Figure: a bottle
 - c. Path: to a point belonging to the interior of the Ground
 - d. Ground: a cave
 - e. Co-Event: floating

Chapter 3 accomplished two things. First, I showed how graphs can be used to

encode the fine structure of a cognitive representation and I sketched a small grammar for graphs of Cognitive Representations. Second, I demonstrated how a graph grammar can be used to parse a graph, and how the same graph can be parsed in different ways by the same grammar.

Encoding Cognitive Representations as graphs was the task of Section 3.2. For the Cognitive Representation described in (4.2), the corresponding graph was (3.3), repeated below as (4.3).



Parsing graphs with a graph grammar was covered at a high level in Section 3.6.3 and at a lower level in Section 3.6.4. The algorithm for parsing detailed there works by filling a chart with “patches”, where a patch is a record that a particular section of the graph (the section of the graph “covered” by the patch) can be generated by an application of a rule from the grammar. The graph in (4.3) is covered by the patch in (4.4), corresponding to the sentence in (4.1).

relevant sentence for each parse.

In order to generate a linguistic expression from a Cognitive Representation, it is necessary to parse its graph with the appropriate graph grammar for the target (human) language. This was the subject of Sections 3.6.3 and 3.6.4. To use the results of the parse, it is useful to have a representation what happened during the parse. These records are called *parse terms*, and obtaining parse terms from a flowgraph grammar is the subject of Section 4.3.

Parse terms are still not linguistic expressions. Terms are closely related to trees, however, and trees can be *transduced* into other trees. Section 4.4 shows how the parse terms of Section 4.3 can be transduced into the kinds of syntactic trees frequently used by syntacticians.

4.2 Definitions

4.2.1 Trees

Trees are useful structures for describing any set of elements that have a hierarchical grouping. Genealogists use them to display progeny, biologists use them to show common evolutionary origins, syntacticians use them to show constituency as well as to show derivations, etc. It is their capacity to describe derivations that is of interest here. A formal treatment of trees follows.

(4.5) **tree** : a tree is a set of primitives called **nodes** together with a binary relation over the nodes called the **dominance relation**, often written D . The dominance relation is reflexive, transitive, and anti-symmetric (i.e., if a node x dominates a node y and y also dominates x then x and y are the same node) and meets two additional conditions:

- a. the **root condition** : one node, the **root node**, dominates all other nodes, and no other node dominates the root.
- b. the **chain condition**: if two nodes x and y both dominate a node z , then either x dominates y or y dominates x (or both, in the case that x and y are the same node).

There are two traditional refinements of the dominance relation:

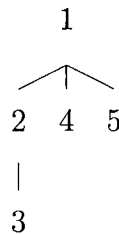
(4.6) **strict dominance (SD)** : a node x *strictly dominates* a node y if both xDy and x and y are distinct nodes.

(4.7) **immediate dominance (ID)** : a node x *immediately dominates* a node y if $xSDy$ and for any node z such that $zSDy$, it is also the case that zDx .

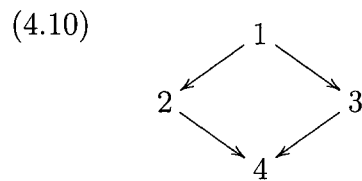
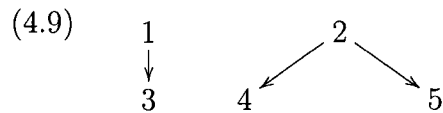
Trees are conventionally drawn with the root at the top and each node above all of the nodes that it dominates. The dominance relation is shown by lines connecting each node to the nodes it immediately dominates. So the nodes N (4.8-a) with the dominance relation D (4.8-b), are drawn as the tree in (4.8-c).

- (4.8) a. $N = \{1, 2, 3, 4, 5\}$
- b. $D = \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 1, 5 \rangle, \langle 2, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 3 \rangle, \langle 4, 4 \rangle, \langle 5, 5 \rangle\}$

c.



When drawn in this way, relations that fail to meet the two additional conditions on dominance relations are easily spotted. The relation in (4.9), for example, violates the root condition as no node dominates both 1 and 2, so no node can dominate all other nodes and the tree has no root. A violation of the chain condition is shown in (4.10), where both 2 and 3 dominate 4, but 2 does not dominate 3 and 3 does not dominate 2.



Nodes on the periphery of a tree often have a special status, so there is a special vocabulary for talking about them:

(4.11) **leaf node** : a leaf node of a tree is a node that dominates only itself.
In (4.8-c) the leaf nodes are 3, 4, and 5.

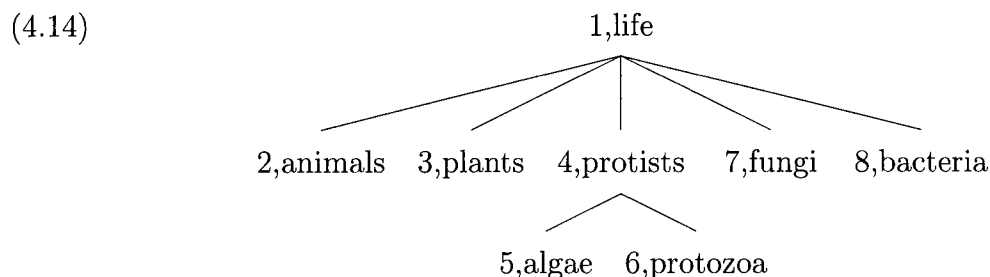
(4.12) **internal node** : an internal node of a tree is any node that is not a leaf node.

A dominance relation determines the fundamental structure of a tree, but it does not order the elements in the tree nor does it place any content in the tree. The next two sections deal with these issues.

4.2.1.1 Labeled trees

(4.13) **labeled trees** : a labeled tree is a quadruple $\langle N, D, \Sigma, \mathbf{L} \rangle$, where N is a set of nodes, D is a dominance relation, Σ is an alphabet, and \mathbf{L} is a function from N into Σ . A labeled tree can be drawn similar to the trees above, but with each node n being written as $n, \mathbf{L}(n)$.

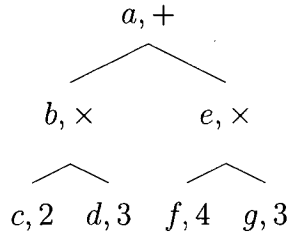
Labeling a tree's nodes allows the tree to present some information in an organized manner. The tree in (4.14) has $N \subset \mathbb{N}$ and has an alphabet Σ that consists of the terms I remember from high school biology. Its dominance and labeling are done such that if one node dominates another, then I believe that the dominating node's label is a term inclusive of the dominated node's label.



The tree in (4.14), could, of course, have been constructed just as well without labels, using the terms I remember from high school biology as the nodes themselves. The advantage of a labeled trees is that different nodes can have the same labels. The tree in (4.15-e) demonstrates this, showing an analysis of the expression $2 \times 3 + 4 \times 3$.

- (4.15) a. $N = \{a, b, c, d, e, f, g\}$
 b. $D = \{\langle a, a \rangle, \langle a, b \rangle, \langle a, c \rangle, \langle a, d \rangle, \langle a, e \rangle, \langle a, f \rangle, \langle a, g \rangle, \langle b, b \rangle, \langle b, c \rangle, \langle b, d \rangle, \langle c, c \rangle, \langle d, d \rangle, \langle e, e \rangle, \langle e, f \rangle, \langle e, g \rangle, \langle f, f \rangle, \langle g, g \rangle\}$

- c. $\Sigma = \{+, \times, 2, 3, 4\}$
- d. $L = \{\langle a, + \rangle, \langle b, \times \rangle, \langle c, 2 \rangle, \langle d, 3 \rangle, \langle e, \times \rangle, \langle f, 4 \rangle, \langle g, 3 \rangle\}$
- e.



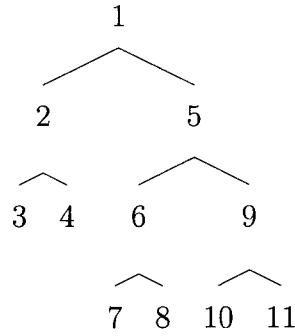
4.2.1.2 Ordered trees

When trees are used as a mechanism for describing strings in natural or formal languages, it is frequently useful to encode the string order in the tree itself.

- (4.16) **leaf-ordered trees** : a leaf-ordered tree is a triple $\langle N, D, \leq \rangle$, where $\langle N, D \rangle$ is a tree as above, and \leq is a total order on the leaf nodes of $\langle N, D \rangle$. The total order \leq is reflexive, transitive, and antisymmetric, and additionally meets the condition that for any two leaf nodes x and y , either $x \leq y$ or $y \leq x$ (or both in the case that x and y are the same node). Leaf-ordered trees are written exactly as other trees, but care is taken to write the leaf nodes left-to-right in the order imposed by \leq .

The total order on the leaf nodes extends to a partial order on the whole set of nodes in the following way: if either a or b is an internal node, then $a \leq b$ just in case all of the leaf nodes that a dominates stand in the \leq relation to all of the nodes that b dominates. In (4.17), for example, $2 \leq 5$, $2 \leq 6$, and $2 \leq 9$, but neither $5 \leq 6$ nor $6 \leq 5$.

(4.17)

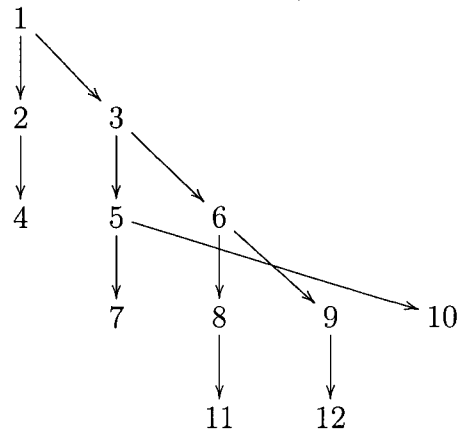


(4.18) **discontinuous constituents** : an internal node n is called a discontinuous constituent if the following hold true:

- a. n dominates at least two distinct leafs a and b ,
- b. there is a third leaf c that n does not dominate, and
- c. $a \leq c \leq b$.

Drawing a tree with a discontinuous constituent often involves crossing lines, as in (4.19), where 5 is a discontinuous constituent dominating both 7 and 10 but not 11 when $7 \leq 11 \leq 10$.

(4.19)



It follows from the definition of a discontinuous constituent that for any two nodes a and b in a tree with *no* discontinuous constituents, either aDb or bDa or

$a \leq b$ or $b \leq a$.

- (4.20) **leaf-ordered, labeled trees** : a leaf-ordered, labeled tree is a quintuple $\langle N, D, \leq, \Sigma, \mathbf{L} \rangle$, where $\langle N, D, \leq \rangle$ is a leaf-ordered tree and $\langle N, D, \Sigma, \mathbf{L} \rangle$ is a labeled tree. Leaf-ordered, labeled trees can be safely written without displaying the nodes (just displaying the labels), as a node can be uniquely identified by its parent and position among its siblings (e.g., first, second, third).

4.2.2 Terms

Many properties of and actions on trees are easier to state as properties of and actions on terms. Comon et al. (2007) offer an excellent introduction to trees and terms, from which the following definitions are adapted.

- (4.21) **Ranked Alphabet** : a ranked alphabet is a pair $\langle \mathcal{F}, \text{arity} \rangle$, where \mathcal{F} is an alphabet in the sense of (3.7), that is, an arbitrary set of symbols and **arity** is a function from \mathcal{F} into \mathbb{N} . The set of symbols f in \mathcal{F} such that **arity**(f) is n is denoted \mathcal{F}_n . The elements of \mathcal{F}_0 are called **constants**, those of \mathcal{F}_1 are called “unary” symbols, \mathcal{F}_2 are called “binary” symbols, and in general the elements of \mathcal{F}_n are called n -ary symbols.

- (4.22) **Ranked Ordered Term** : terms are built from symbols in a ranked alphabet \mathcal{F} as well as a set of constants (i.e., 0-ary symbols) \mathcal{X} called **variables**, with \mathcal{F} and \mathcal{X} disjoint. The set of terms built from \mathcal{F} and \mathcal{X} is denoted $T(\mathcal{F}, \mathcal{X})$ and is the smallest set meeting these conditions:

- a. $\mathcal{F}_0 \subset T(\mathcal{F}, \mathcal{X})$

- b. $\mathcal{X} \subset T(\mathcal{F}, \mathcal{X})$
- c. if $f \in \mathcal{F}_p$ for some $p > 0$ and t_1, t_2, \dots, t_p are in $T(\mathcal{F}, \mathcal{X})$, then $f(t_1, t_2, \dots, t_p)$ is also in $T(\mathcal{F}, \mathcal{X})$. The terms t_1, t_2, \dots, t_p are called **subterms**

4.2.3 From terms to trees and back

Terms as defined in (4.22) are isomorphic to labeled, leaf-ordered trees with no discontinuous constituents, modulo a certain wrinkle: in a leaf-ordered labeled tree, the sets of nodes immediately dominated by nodes with the same label may differ in cardinality, but in a term over a ranked alphabet, each symbol has a fixed arity. So a label C in a tree may correspond to distinct symbols C_1, C_2, \dots in a term where C_1 and C_2 have differing arities.

4.2.3.1 From terms to trees

The transformation of a term into a tree is straightforward. Constants correspond to single node trees, and complex term $f(t_1, t_2, \dots, t_p)$ is built from the trees corresponding to each of its subterms and a node corresponding to f dominating them.

When assembling distinct trees into a single structure, it is important to keep the nodes of each tree distinct. The function in (4.23) can be used for this purpose.

(4.23) For $n \in \mathbb{N}$ and a tree $\delta = \langle N, D, \leq, \Sigma, \mathbf{L} \rangle$, let $\mathbf{P}(n, \delta) = \langle N', D', \leq', \Sigma, \mathbf{L}' \rangle$, where:

- a. $N' = \{n\} \times N$
- b. for $\nu_1, \nu_2 \in N$,

- (i) $\langle \langle n, \nu_1 \rangle, \langle n, \nu_2 \rangle \rangle \in D'$ just in case $\langle \nu_1, \nu_2 \rangle \in D$
- (ii) $\langle \langle n, \nu_1 \rangle, \langle n, \nu_2 \rangle \rangle \in \leq'$ just in case $\langle \nu_1, \nu_2 \rangle \in \leq$
- c. $\mathbf{L}' = \{ \langle \langle n, \nu \rangle, \sigma \rangle \mid \langle \nu, \sigma \rangle \in \mathbf{L} \}$

Starting with a term $t \in T(\mathcal{F}, \mathcal{X})$, the corresponding tree $\mathbf{h}(t) = \langle N, D, \leq, \Sigma, \mathbf{L} \rangle$ is constructed in the following way:

- (4.24)
- a. a constant $t \in (\mathcal{F}_0 \cup \mathcal{X})$ becomes a tree with a single node
 $\mathbf{h}(t) = \langle \{1\}, \langle 1, 1 \rangle, \langle 1, 1 \rangle, \{t\}, \langle 1, t \rangle \rangle$
 - b. a term $t = f(t_1, t_2, \dots, t_p)$ with p subterms becomes a tree with a node corresponding to f dominating all of the nodes of the trees $\mathbf{h}(t_1), \mathbf{h}(t_2), \dots, \mathbf{h}(t_p)$
 - (i) the function \mathbf{P} is used to keep the nodes of each subtree δ_i distinct

$$\delta_i = \langle N_i, D_i, \leq_i, \Sigma_i, \mathbf{L}_i \rangle = \mathbf{P}(i, \mathbf{h}(t_i)), 1 \leq i \leq p$$
 - (ii) the union of the subtree nodes, N' , is $\bigcup_{i=1}^p N_i$
 - (iii) the nodes of $\mathbf{h}(t)$ are the nodes of the subtrees, along with a new node for the root

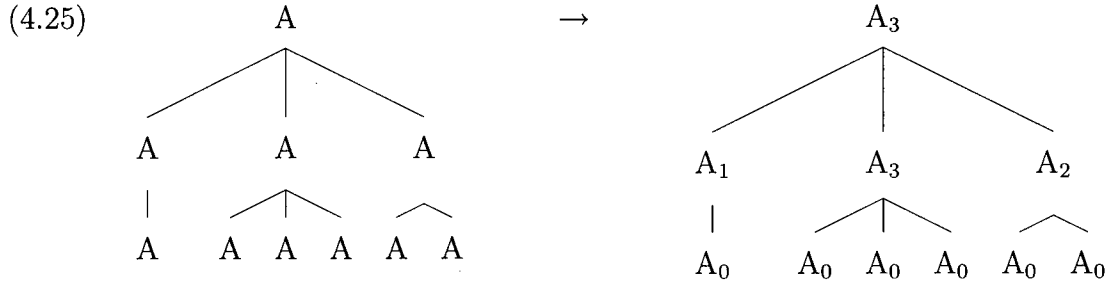
$$N = \{1\} \cup N'$$
 - (iv) all the dominance relations of the subtrees are maintained, plus the new root dominates everything

$$D = (\{1\} \times N) \cup \bigcup_{i=1}^p D_i$$
 - (v) $\Sigma = \bigcup_{i=1}^p \Sigma_i$
 - (vi) $\mathbf{L} = \bigcup_{i=1}^p \mathbf{L}_i \cup \{ \langle 1, f \rangle \}$

4.2.3.2 From trees to terms

As mentioned earlier, even trees with no discontinuous constituents offer slightly more freedom than ranked ordered terms; they allow different nodes with the same label to immediately dominate distinct numbers of nodes.

The first step in transforming a tree into a term, then, is to annotate each label with the number of nodes it immediately dominates. If $\mathbf{L} : N \rightarrow \Sigma$ is the labeling function of a tree, then the new labeling function is $\mathbf{L}'(n) = \mathbf{L}(n)_{|\{x|nIDx\}|}$, where σ_i indicates the symbol σ annotated with the arity i . The pair of trees in (4.25) shows how this annotation works.



Let $\delta = \langle N, D, \leq, \Sigma, \mathbf{L} \rangle$ be a leaf-ordered labeled tree that has no discontinuous constituents and that has undergone this arity annotation process. The labels of δ induce a ranked alphabet \mathcal{F}^δ which each symbol $\sigma_i \in \mathcal{F}_i^\delta$. The term $t = \mathbf{h}^{-1}(\delta)$ corresponding to δ is built from the root of δ down, following the steps in (4.26)

- (4.26) a. Let ρ be the root of δ , and s be the sequence $\langle s_1, s_2, \dots, s_p \rangle$ of the nodes immediately dominated by ρ ordered according to the extension of \leq that covers internal nodes. The elements of s are guaranteed to be ordered by \leq as δ contains no discontinuous constituents and no element of s can dominate any other elements of s , or else the dominating element would not be immediately domi-

nated by ρ .

- b. If the sequence s is empty, then the term t is simply $\mathbf{L}(\rho)$.
- c. If the sequence s is non-empty, then
 - (i) let δ_i , $1 \leq i \leq |s|$ be the subtree of δ rooted at s_i . This tree is:

$$\langle \{n|s_i Dn\}, \{ \langle n_1, n_2 \rangle \in D | s_i Dn_1 \text{ and } s_i Dn_2 \}, \leq, \Sigma, \mathbf{L} \rangle$$

- (ii) the term t is $(\mathbf{L}(\rho))(\mathbf{h}^{-1}(\delta_1), \mathbf{h}^{-1}(\delta_2), \dots, \mathbf{h}^{-1}(\delta_p))$.

4.3 Parse Terms

4.3.1 String Derivations and Parse Trees

Parsing an object with a grammar is essentially equivalent to providing a derivation of the object using that grammar. However, derivations can contain superfluous information, which is often discarded when parsing. As an example of this unwanted information, Chomsky (2002, pp. 26–28) offers the derivation of (4.28), where the offset letter to the right of each row refers to the rule in (4.27) that was used to arrive at that step in the derivation.

- (4.27)
- a. $Sentence \rightarrow NP + VP$
 - b. $NP \rightarrow T + N$
 - c. $VP \rightarrow Verb + NP$
 - d. $T \rightarrow the$
 - e. $N \rightarrow man, ball, \text{etc.}$
 - f. $Verb \rightarrow hit, took, \text{etc.}$

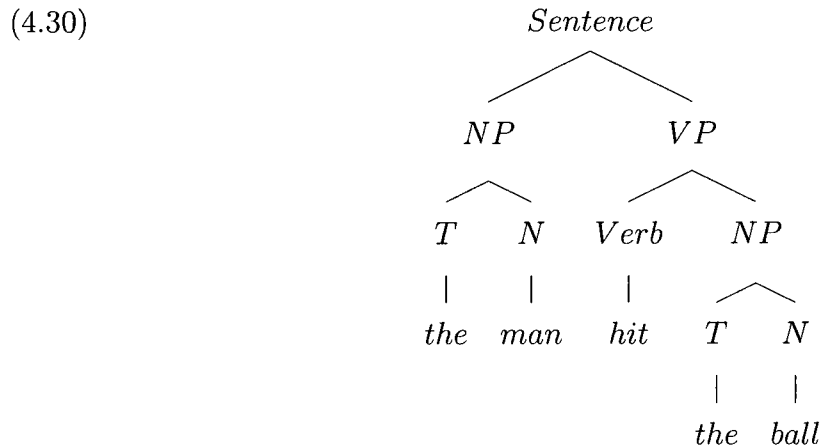
- (4.28)
- a. *Sentence*
 - b. $NP + VP$ (a)
 - c. $T + N + VP$ (b)
 - d. $T + N + Verb + NP$ (c)
 - e. $the + N + Verb + NP$ (d)
 - f. $the + man + Verb + NP$ (e)
 - g. $the + man + hit + NP$ (f)
 - h. $the + man + hit + T + N$ (b)
 - i. $the + man + hit + the + N$ (d)
 - j. $the + man + hit + the + ball$ (e)

So, a derivation is a sequence of rule applications; two different sequence of rule applications are two different derivations. For example, the rule applications in (4.28-e) and (4.28-f) could have been done in the opposite order as in (4.29). This is a different derivation, but it is not different in any interesting way. Inverting (4.28-e) and (4.28-f) has no effect on the following steps of the derivation nor on the final result.

- (4.29)
- e'. $T + man + Verb + NP$ (e)
 - f'. $the + man + Verb + NP$ (d)

For this reason, parses of strings are often represented as *trees*. The root of the tree is the start symbol of the grammar, and the leafs of the tree are the elements of the string, written in their order in the string. The internal nodes of the tree are determined in the following way: For each rule that was applied in generating the string, the node corresponding with the left-hand side of the rule *immediately dominates* (i.e., has as daughters) nodes corresponding to the right hand side of

the rule. In other words, if a derivation makes use of the rule (4.27-c), then the tree corresponding to that derivation will have a node labeled *VP* dominating nodes labeled *Verb* and *NP*. Following this process, the tree in (4.30) represents the derivations of both (4.28) and (4.29):



The *parse tree* in (4.30) records what got rewritten as what, but not the order in which two independent rewrites happened. The former is considered interesting for many reasons, one being that syntactic elements with a common origin (e.g., as *hit*, *the*, and *ball* all share the common origin *VP* in (4.30)) are usually understood to group semantically as well. The latter is usually not attributed any theoretical significance.

A *parse term* is essentially the same thing as a parse tree and the two are often used interchangeably. To arrive at a parse term from a parse tree, each node has its label written like a mathematical function and the daughters of that node, if there are any, are written as its arguments. The parse tree in (4.31-a), for example, corresponds to the parse term in (4.31-b).

there are well understood automata and transducers that operate on trees and these automata can be used to derive more traditional linguistic representations from the graph parses. In order to take advantage of these automata, then, it is necessary to create tree representations for the graph parses.

The biggest decision to make in creating a tree representation for the graph parses is what information the tree should encode. Section 3.6 offers an answer here: what matters for the parse is what rules get used to build each part of the graph, along with the components that each application of the rule assembles. To encode this information in a tree (or term), each rule can be outfitted with a recipe for the construction of a ranked order term, with its subterms (if any) determined by the rules used to rewrite the non-terminal elements of the rule's right-hand side.

The simplest case is the analog of lexical insertion rules, i.e. those rules whose right hand sides contain no non-terminals; the term that they generate cannot vary.

$$(4.32) \quad Term \left(\overset{1}{\rightarrow} \boxed{Thing} \Rightarrow \overset{1}{\rightarrow} \boxed{bottle} \right) = \mathbf{bottle}$$

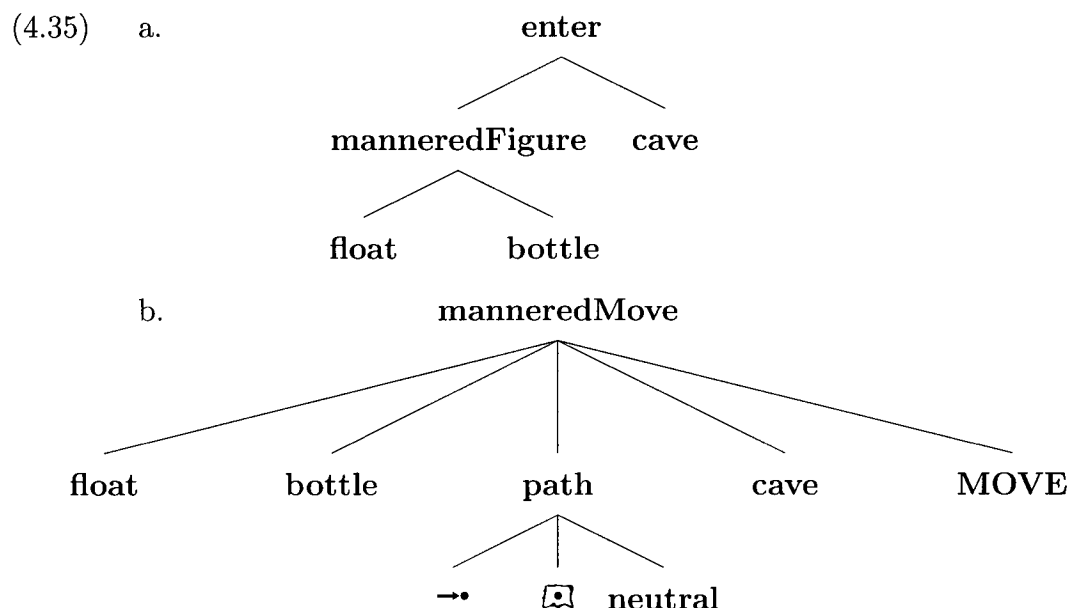
Another simple case is rules that have exactly one non-terminal on the right hand side; the term generated by the rule can have at most one variable, so there can be no ambiguity about which subterm corresponds to which non-terminal. The *Term* function in this case can be expressed as in (4.33), where the right hand side of the equation indicates the term generated by whatever rule expands the non-terminal \boxed{Thing} .

$$(4.33) \quad Term \left(\overset{1}{\rightarrow} \boxed{Ground} \Rightarrow \overset{1}{\rightarrow} \boxed{ground} \rightarrow \boxed{Thing} \right) = Term \left(\rightarrow \boxed{Thing} \right)$$

When the right hand side has multiple non-terminals, some care must be taken to keep them distinct, as two non-terminals could have the same label. One way to differentiate them is to more fully label the edges of the right hand side, and invoke these edge labels in the specification of the term. The rule for introducing Motion with Manner (3.25) is so annotated in (4.34-a), the rule for the idiosyncratic English ‘enter’ (3.27) is annotated as in (4.34-b), and (4.34-c) shows the annotation for the rule introducing an optional Manner to the figure (3.26-b).

$$\begin{aligned}
 (4.34) \quad a. \quad & \text{Term} \left(\begin{array}{c} \xrightarrow{1} S \Rightarrow \\ \begin{array}{c} \xrightarrow{\quad} \text{self-contained motion} \xrightarrow{2} \text{SCMType} \\ \xrightarrow{\quad} \text{figure} \xrightarrow{3} \text{Thing} \\ \xrightarrow{1} \text{motion event} \xrightarrow{4} \text{Path} \\ \xrightarrow{\quad} \text{Motion} \xrightarrow{6} \\ \xrightarrow{\quad} \text{Ground} \xrightarrow{5} \end{array} \end{array} \right) \\
 &= \text{manneredMove} \left(\text{Term} \left(\xrightarrow{2} \text{SCMType} \right), \text{Term} \left(\xrightarrow{3} \text{Thing} \right), \right. \\
 &\quad \left. \text{Term} \left(\xrightarrow{4} \text{Path} \right), \text{Term} \left(\xrightarrow{5} \text{Ground} \right), \text{Term} \left(\xrightarrow{6} \text{Motion} \right) \right) \\
 \\
 b. \quad & \text{Term} \left(\begin{array}{c} \xrightarrow{1} S \Rightarrow \\ \begin{array}{c} \text{MOVE} \\ \uparrow \\ \text{motion} \xrightarrow{2} \text{Figure} \\ \xrightarrow{1} \text{motion event} \xrightarrow{\quad} \text{path} \xrightarrow{\quad} \text{conform} \xrightarrow{\quad} \text{vector} \xrightarrow{\quad} \text{---} \\ \xrightarrow{3} \text{Ground} \xrightarrow{\quad} \text{deictic} \xrightarrow{\quad} \text{neutral} \end{array} \end{array} \right) \\
 &= \text{enter} \left(\text{Term} \left(\xrightarrow{2} \text{Figure} \right), \text{Term} \left(\xrightarrow{3} \text{Ground} \right) \right) \\
 \\
 c. \quad & \text{Term} \left(\begin{array}{c} \xrightarrow{1} \text{Figure} \Rightarrow \\ \begin{array}{c} \xrightarrow{\quad} \text{self-contained motion} \xrightarrow{2} \text{SCMType} \\ \xrightarrow{\quad} \text{figure} \xrightarrow{3} \text{Thing} \end{array} \end{array} \right) \\
 &= \text{manneredFigure} \left(\text{Term} \left(\xrightarrow{2} \text{SCMType} \right), \text{Term} \left(\xrightarrow{3} \text{Thing} \right) \right)
 \end{aligned}$$

With similar terms for the remaining rules, the English generation grammar would generate two parse terms for (4.3). One is (4.35-a), produced using (4.34-b) followed (4.34-c) (corresponding to the sentence “The bottle entered the cave floating”). The other is (4.35-b), using the expansion in (4.34-a) (corresponding to “The bottle floated into the cave”).



4.4 Tree Transductions

The remaining work is to transform terms like (4.35) into linguistic expressions. If syntactic trees are desired, then the remaining work is a translation from one tree to another: an ideal task for a tree transducer. I will demonstrate the technique here with a top-down deterministic macro tree transducer (Engelfriet, 1980).

A *tree transducer* is a set of rewriting rules for trees, translating trees from a source tree language into a target tree language. In this case, the source tree language is the parse terms from the graph parses, and the destination tree language is Government and Binding style syntactic trees (Chomsky, 1981, e.g.).

Tree transducers in general rewrite a term in some way depending on its subterms. A *macro tree transducer* rewrites a term depending both on its subterms and on contextual information called *parameters*.

Formally, a macro tree transducer has five components. It has three ranked alphabets, an initial state, and a set of rules. The three ranked alphabets are: the alphabet Σ of input symbols, the alphabet Δ of output symbols, and the alphabet \mathcal{Q} of states, in which one state q_0 is determined to be the *initial state*. The rules in a macro tree transducer have the form in (4.36), where $q \in \mathcal{Q}$ and $\sigma \in \Sigma$ are of the appropriate rank and t is in the set **RHS** defined in (4.37) (where, e.g. Δ_0 denotes the symbols in Δ with rank 0).

$$(4.36) \quad q(y_1, \dots, y_n, \sigma(x_1, \dots, x_m)) \rightarrow t$$

(4.37) The set of Right Hand Sides (**RHS**) is the smallest set meeting the following conditions (Engelfriet, 1980, p. 260):

- a. $\{y_1, \dots, y_n\} \cup \Delta_0 \subseteq \mathbf{RHS}$,
- b. if $t_1, \dots, t_k \in \mathbf{RHS}$ and $f \in \Delta_k$, then $f(t_1, \dots, t_k) \in \mathbf{RHS}$, and
- c. if $t_1, \dots, t_k \in \mathbf{RHS}$, $q' \in \mathcal{Q}_{k+1}$ and $x_i \in \{x_1, \dots, x_m\}$ then $q'(t_1, \dots, t_k, x_i) \in \mathbf{RHS}$.

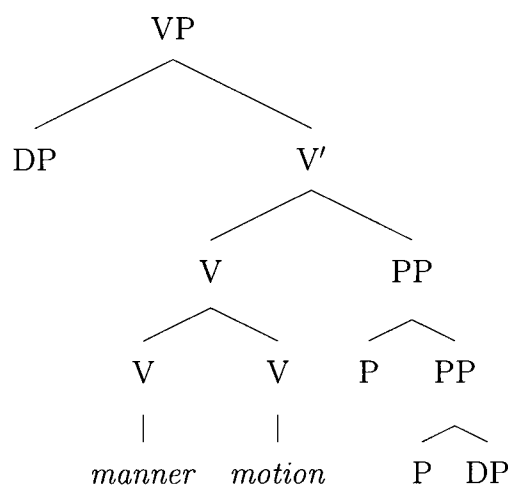
The transduction process starts by creating a term with the initial state q_0 dominating a term over the input language. In each step the term is rewritten in accordance with the rules. In the case of a top-down macro tree transducer, this has the effect of moving the states ever deeper into the term.

4.4.1 Transducing to Syntactic Structures

In the case of transducing graph parse terms to (traditional linguistic) syntactic trees, the input alphabet Σ is the alphabet containing the symbols in the parse terms—i.e., the bold face items in (4.35)—and the output alphabet Δ consists of lexical and categorical symbols of a traditional linguistic syntactic analysis.

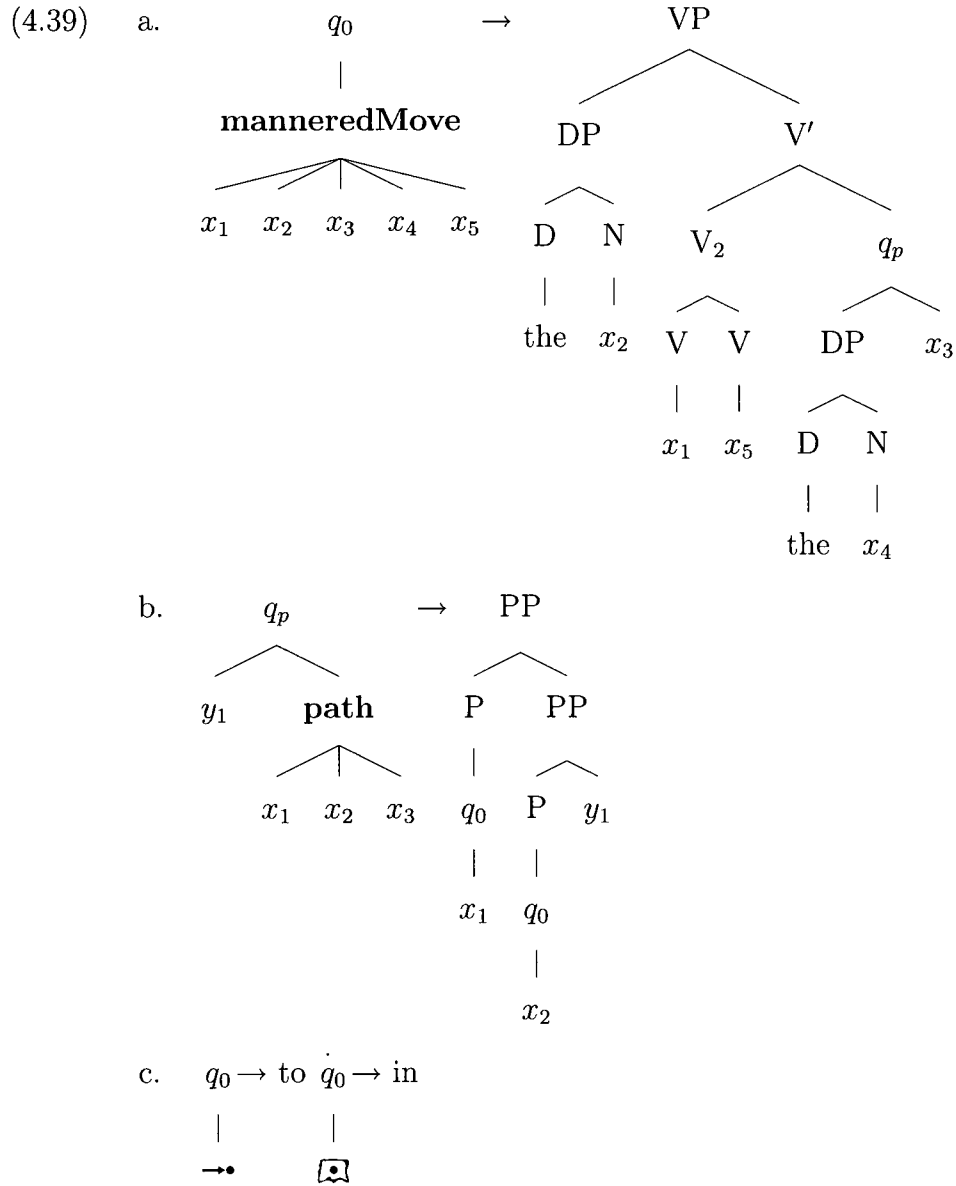
Zubizaretta and Oh (2007) determine the structure in (4.38) to be the syntactic structure that (at least in Germanic languages) is responsible for combining the meaning of manner and motion, and claim that the ability of the Germanic languages to simultaneously express Manner and Motion is correlated with the freedom with which the Germanic languages form same-category compounds. The particular case of Manner and Motion forming a compound (in their analysis) involves a Motion head that is silent when compounded with another verb, but surfaces in English as ‘come’ or ‘go’ or ‘move’ (determined by the Deictic component of the motion event) when not in a compound.

(4.38)



The tree transducer of interest, then, is the one that takes (4.35-b) and produces a tree in the form of (4.38). The transduction rules given in (4.39) are sufficient

for this purpose, and a transduction of (4.35-b) is shown in Figure 4.1. Notable in this transduction is the way that (4.39-a) passes the DP term for the Ground as a parameter to (4.39-b).



For the case of ‘enter’ the transduction is more interesting, due to the optional Manner. The parse term for ‘enter’ with Manner was given in (4.35-a). To

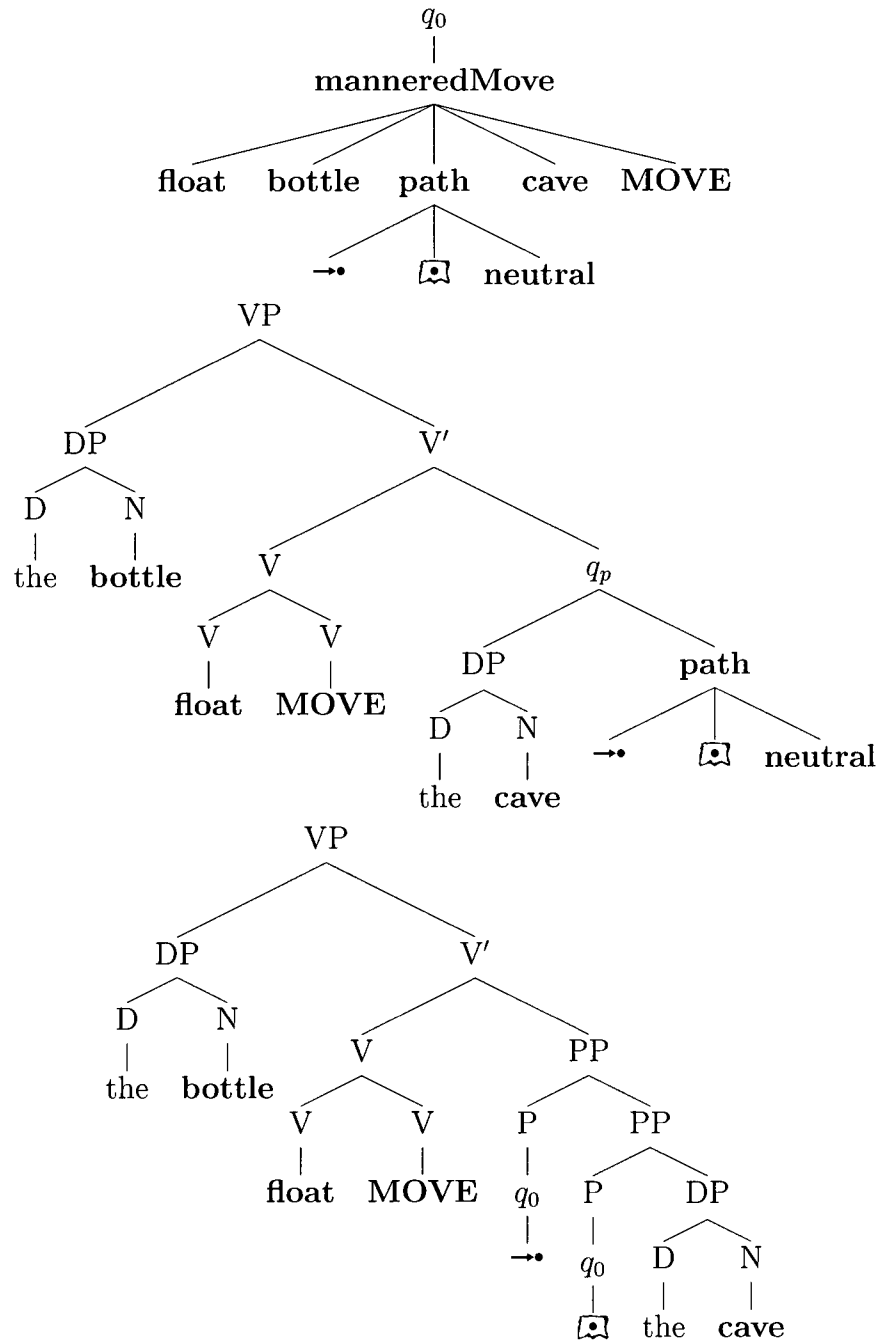


Figure 4.1: Transduction from a parse term to a traditional syntactic analysis

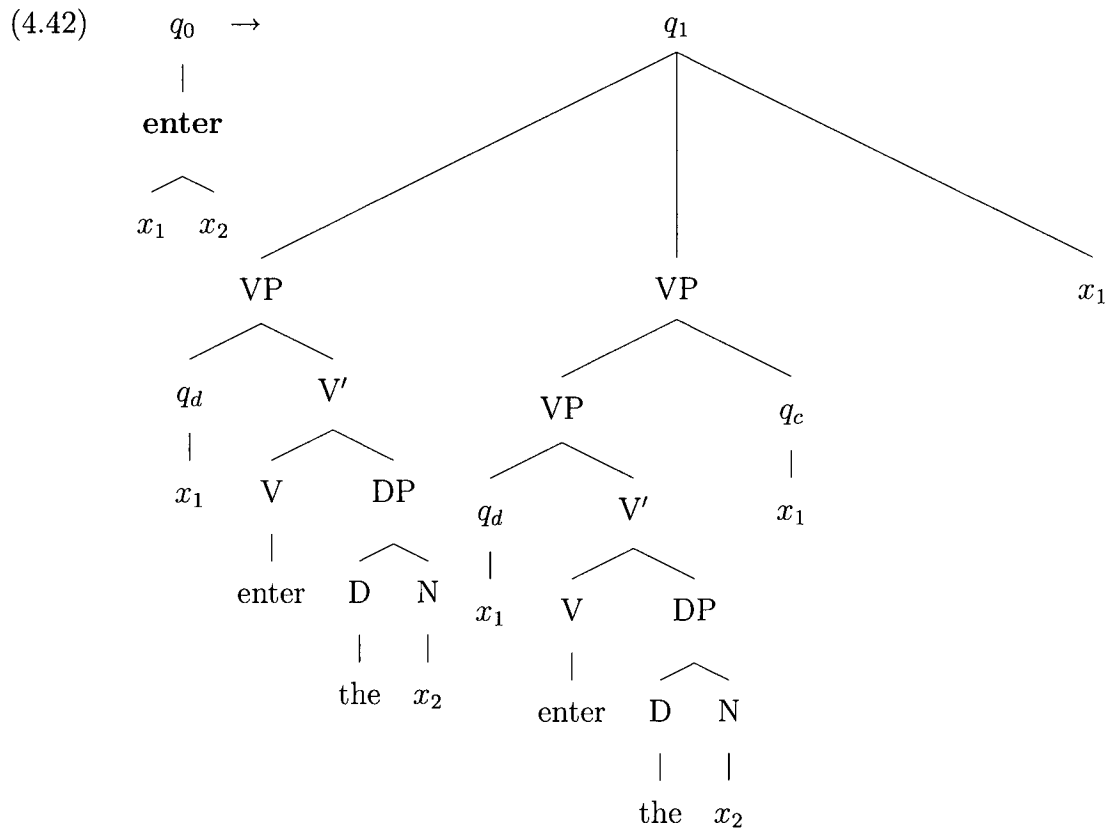
generate a parse term for Cognitive Representation with no Manner, the rule in (4.40) is used, and the resulting parse term is that in (4.41).

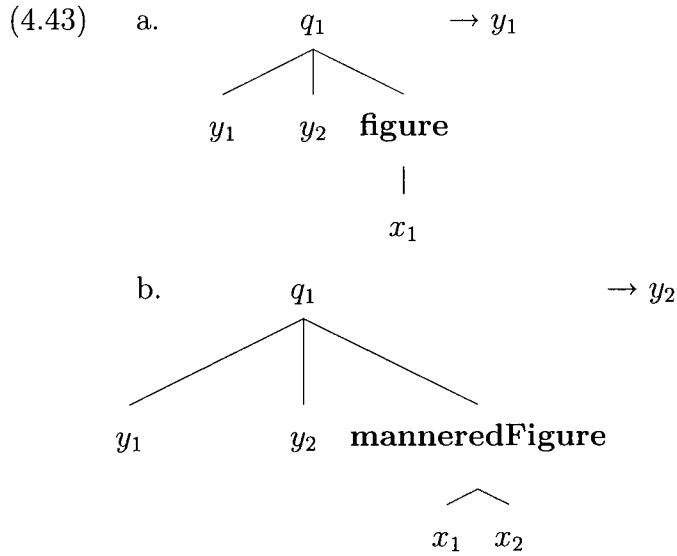
$$(4.40) \quad \text{Term} \left(\overset{1}{\rightarrow} \boxed{\text{Figure}} \Rightarrow \rightarrow \boxed{\text{figure}} \overset{2}{\rightarrow} \boxed{\text{Thing}} \right) \\ = \mathbf{figure} \left(\text{Term} \left(\overset{2}{\rightarrow} \boxed{\text{Thing}} \right) \right)$$

$$(4.41) \quad \begin{array}{c} \mathbf{enter} \\ \swarrow \quad \searrow \\ \mathbf{figure} \quad \mathbf{cave} \\ | \\ \mathbf{bottle} \end{array}$$

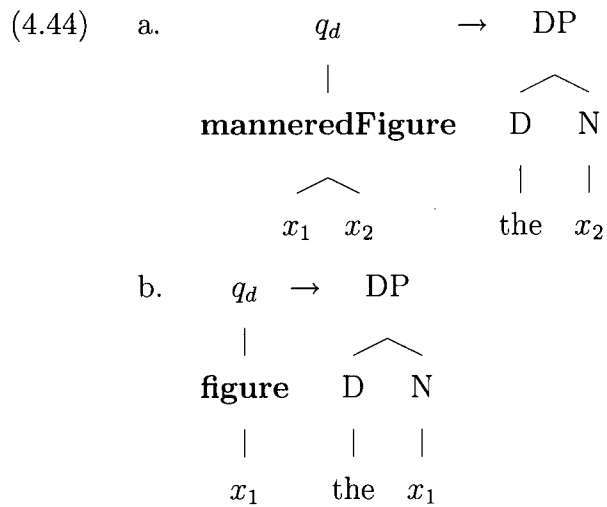
The linguistic target for the transduction depends quite a bit on the presence or absence of the Manner. In the case of the parse term involving **enter** and **manneredFigure**, the output term will have an adjunction site where the Manner will be expressed. When there is no Manner, the transduction should also have no adjunction site. This poses a small challenge, because the rewrite rules rewrite a term without knowledge of its subterms; in terms of trees, they rewrite a node without any knowledge of its daughters. Here, another interesting aspect of Top-Down Deterministic Macro Tree Transducers comes into play: they are closed under regular look-ahead (Engelfriet and Vogler, 1985, pp. 113–116). *Look-ahead* is what it sounds like, the ability to look ahead into the future, which for a term is the term’s subterms and their subterms, etc. With the ability to look ahead to the children of **enter**, there is no problem in determining which linguistic tree to build (i.e., the one with or the one without the adjunction site). *Regular* means that this knowledge of the future is encoded by a finite state machine, so it’s not perfect knowledge about the future, but knowledge that the future fits into one of finitely many categories of possible futures. That suits the need here,

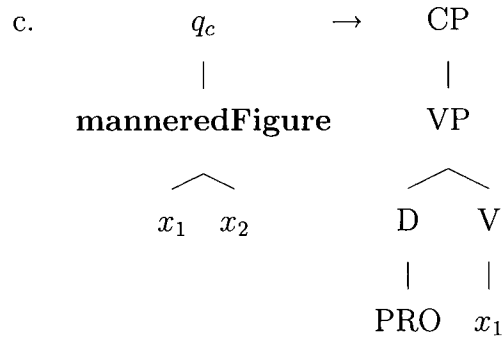
as it only matters whether the first daughter of **enter** is **mannered** or **figure**. *Closed under regular look-ahead*, then, means that for every Top-Down Deterministic Macro Tree Transducer that has this look-ahead ability, there is another one that performs the exact same translations but without looking ahead. The short-sighted transducer achieves this by first creating all of the trees that it might need and storing them in parameters, and then later picking the one that it actually needs. The rule in (4.42) shows the creation and storage of the possible trees, and the pair of rules (4.43-a) and (4.43-b) shows the decision between them.





With the tree decided, the transduction has only to put together the syntactic constituents for the Figure and the optional Manner. The former is taken care of by (4.44-a) and (4.44-b), while (4.44-c) creates a small CP for the latter.





4.4.2 Transduction to Derivation Trees

An interesting alternative would be to transduce parse terms to the derivation trees of some syntactic formalism like that of Stabler (1997). In this way the transduction rules could easily be interpreted not as generating structure directly, but as informing the syntactic cognitive module about what needs to be done. The syntactic component would then be wholly responsible for determining what structures result from following the indications of the cognitive representation parser.

4.5 Generative power of Macro Tree Transducers

As top-down deterministic macro tree transducers are the final step in obtaining an appropriate sentence for a given parse, the generative power of macro tree transducers determines the generative power of the entire process.

For a class of tree transducers, there are three relevant language classes. There is the class of input tree languages, the class of output tree languages, and the class of string languages that are yields of the output tree languages.

The input tree language, that is, the set of trees coming into the transducer, largely determines the set of trees coming out. Comon et al. (2007, p. 61) demon-

strate that the derivation trees of context free languages are the same sets of trees as the *regular tree languages*. Since the input to the transduction in this case is a parse term of a context free language, it follows that the trees given as input form regular tree language. The class of regular tree languages is denoted by $REGT$.

The class of tree languages generated by top-down deterministic macro tree transducers with regular input trees is denoted by $MTT(REGT)$. Kühnemann (1996) gives a pumping lemma for $MTT(REGT)$ and shows that the set of all *monadic trees* (i. e., trees that are essentially just strings with dominance in the tree taking on the role of precedence in the string) with double exponential height is not a member of $MTT(REGT)$, but is unable to establish any strong results on languages of non-monadic trees.

The string yield of a tree means the string of all the labels of the tree's leafs in their natural order. The string yields of $MTT(REGT)$ include all of the context free languages. Consider, for example, the context free language (alternatively, the regular tree language) described by the grammar in (4.45). It yields the language $a^n b^n$. Clearly a transducer performing the identity transformation on its derivation trees yields the same language. The tree transducer in (4.46), on the other hand, yields the language $a^n b^n c^n$ given the same trees as input (demonstrated in Figure 4.2). So, the yields of $MTT(REGT)$ go beyond context free.

- (4.45)
- a. $S_1 \rightarrow \epsilon$
 - b. $S_1 \rightarrow S_2$
 - c. $S_1 \rightarrow S_3$
 - d. $S_2 \rightarrow ab$
 - e. $S_3 \rightarrow aS_2b$

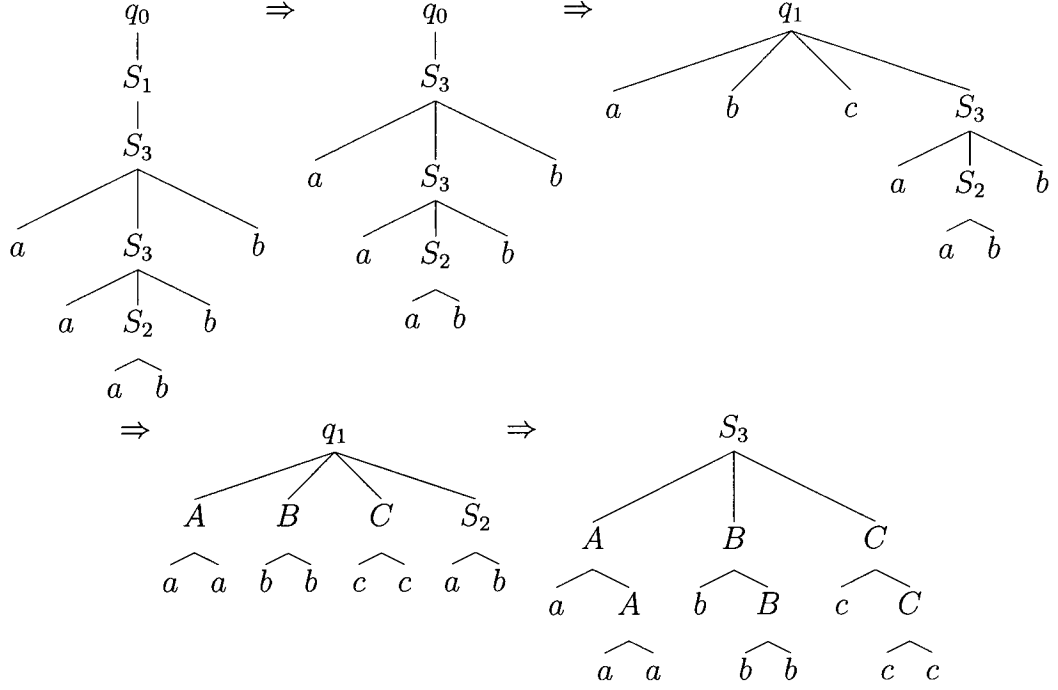


Figure 4.2: Transduction from tree yielding $aaabbb$ to tree yielding $aaabbbccc$

- f. $S_3 \rightarrow aS_3b$
- (4.46) a. $q_0(S_1(x_1)) \rightarrow q_0(x_1)$
b. $q_0(\epsilon) \rightarrow \epsilon$
c. $q_0(S_2(x_1, x_2)) \rightarrow S_3(a, b, c)$
d. $q_0(S_3(x_1, x_2, x_3)) \rightarrow q_1(a, b, c, x_2)$
e. $q_1(y_1, y_2, y_3, S_2(x_1, x_2)) \rightarrow S_3(A(a, y_1), B(b, y_2), C(c, y_3))$
f. $q_1(y_1, y_2, y_3, S_3(x_1, x_2, x_3)) \rightarrow q_1(A(a, y_1), B(b, y_2), C(c, y_3), x_2)$

Engelfriet and Maneth (2002a,b); Maneth (1999) establish some bounds on the string yields of $MTT(REGT)$, which they denote ${}_yMTT(REGT)$. One result they establish is the there are infinite intertwining hierarchies of string language classes created by iterating various kinds of transductions (Figure 4.3). In particular, ${}_yMTT(REGT)$ is sandwiched between $EDT0L(EDT0L(REG))$ and

$$\begin{aligned}
& EDT0L(REG) \subsetneq EDT0L(EDT0L(REG)) \subsetneq {}_yMTT(REGT) \subsetneq \\
& EDT0L^3(REG) \subsetneq {}_yMTT^2(REGT) \dots \subsetneq {}_yMTT^n(REGT) \subsetneq \\
& EDT0L^{n+2}(REG)
\end{aligned}$$

Figure 4.3: Hierarchy of languages generated by iterated *MTT* and *EDT0L* transductions

$EDT0L^3(REG)$, where REG denotes the regular string languages and $EDT0L(\mathcal{L})$ denotes the class of string languages generated by *extended, deterministic, tabular, context-free Lindenmayer* systems controlled by languages of class \mathcal{L} . An EDT0L is a string rewriting system $\langle \Sigma, H, \omega, \Delta \rangle$, where Σ is an alphabet, $\Delta \subseteq \Sigma$ is the terminal alphabet, $\omega \in \Sigma^+$ is the *axiom*, and H is a set $\{h_1, h_2, \dots\}$ of functions $\Sigma \mapsto \Sigma^*$. A string derivation of an EDT0L starts with the axiom and applies a sequence of elements of H as a string homomorphisms. A string w derived by an EDT0L G is in the language $L(G)$ just in case $w \in \Delta^*$. An EDT0L controlled by a language $L \subset H^*$ derives just those strings whose derivations involve a sequence of homomorphisms that is a member of L .

4.6 Conclusions

After parsing Cognitive Representation with an appropriate grammar there is a record of the parse—or of the parses, in the case of an ambiguity. The goal of this dissertation, however, is not to produce parses of CRs, but rather to produce sentences that evoke CRs.

The solution explored here is to transduce the parses into appropriate sentences. Parse terms are a natural way of representing parses, and macro tree transducers have the power to transduce parse terms into trees that are more familiar to linguists.

Clearly macro tree transducers are powerful machines. This has a good side,

in that one can be optimistic that it will be possible to generate string languages of interest to linguists. It also has the disadvantage that it offers no insight into why do humans generate the sentences that they do, and why don't they generate the sentences that they don't. Various restrictions on the powers of macro tree transducers and the effects of these restrictions on the possible string yields is an area of current research.

CHAPTER 5

Conclusion

I have given an exposition of Talmy's (2000a, 2000b) program of Cognitive Semantics and developed a novel method of generating linguistic expressions intended to evoke a targeted Cognitive Representation. This method involves rendering the target Cognitive Representation as a graphs, parsing the graph with a graph grammar, obtaining a term from the parse, and finally transducing the parse term into a traditional syntactic tree for a sentence which evokes the targeted CR.

The primary intuition of this parsing-by-generation approach is that there is a fundamental symmetry between sentence understanding and sentence generation. When a listener is attempting to understand a sentence, she must choose between the possibly large number of syntactic analyses that are compatible with that sentence. In generation, a speaker must choose between the array of sentences that convey her intended meaning. Using a parser for both tasks captures this similarity. In parsing a sentence to be understood, there is ambiguity in how to carve the sequence of words (or sounds in a detailed account) into larger syntactic categories. In parsing a CR, there is ambiguity in how to carve the elements of meaning into lexical items that express them.

I restricted my attention to Talmy's description of the Cognitive Representations of motion events and sketched a grammar for their graph renderings. Typological study of lexicalization patterns reveals that there are a rather limited set of major conflation patterns and the rules of the grammar were tailored

to match these major patterns. In this way the ambiguities encountered by the parser are precisely those ambiguities that a speaker faces when trying to force her meaning into a sequence of lexical items.

In implementing my method of generation, I switched from graphs and graph grammars to flowgraphs and flowgraph grammars. To avoid any complaints of “bait and switch”, I provided an algorithm for obtaining a flowgraph from a graph. The switch was not empirically or theoretically motivated. Rather, it was a pragmatic decision motivated by my superior understanding of algorithms for parsing flowgraphs compared to general graph parsing. As the algorithm for obtaining a flowgraph from a graph can be incorporated into the sentence-generation pipeline, the original claim of the thesis—that rendering Cognitive Representations as graphs is a useful step in generating sentences to evoke them—still stands.

I demonstrated the transduction from parse terms to syntactic trees using top-down deterministic macro tree transducers and presented upper and lower bounds on their generative power. These bounds are quite high, so the algorithm proposed here offers very little in the way of explanation for why humans generate the particular kinds of sentences that they do.

The interesting directions for further related research depart from the two ends of the generation process here. Talmy’s Cognitive Semantics covers much more than just motion events. The most promising area in this direction is likely Talmy’s theory of *Force Dynamics*, as this would bring in more relations between events such as causation and prevention and might lead to the generation of recursive syntactic structures. The other promising branching point for future research is to vary the kind of tree transduction performed on the parse terms. Results about the languages generated by macro tree transducers have proven

difficult (see comments at Kühnemann, 1996, p. 67, for example), but the situation here offers a question should be easier to answer: For any given reduction in the power of the transducer, is it still possible to generate the desired sentences?

APPENDIX A

A (slightly) Larger Graph Grammar

A.1 Introduction

Chapter 3 sketched a graph grammar for Cognitive Representations (Talmy, 2000a,b). The grammar included two sets of rules which generated an ambiguity.

One set of rules contained a rule (3.25) for simultaneously introducing a Motion Event and a Self-Contained Motion, and then rules for specifying the Path and other constituents of the Motion Event. This set of rules was intended to mimic the primary English pattern of verbs simultaneously expressing fact-of-motion and manner-of-motion.



A second set of rules included a rule (3.27) which introduced the Motion Event with the Vector and the Conform of the Path already specified. This set of rules was intended to mimic the exceptional English verb ‘enter’.



A main theme of this dissertation is that the kind of ambiguity of produced by these two sets of rules models the choice between lexical items that a speaker describing a motion event must make. This appendix continues the exposition of the fine structure of Path that began in Section 3.2 and sketches several more productions appropriate for a graph grammar of the Cognitive Representations evoked by English sentences about Motion Events.

A.2 (Vector and) Conform

Talmy defines Path in terms of three constituents, the Vector, the Conform, and the Deictic. Table 3.1 (p. 46) presents what Talmy (2000b, p. 53f) describes as the complete, language-universal list of Vectors. The Deictic element in English plays only a limited role in the selection of verbs and prepositions (it distinguishes, for example, ‘come’ from ‘go’). It remains, then, to provide some exposition of the conform.

A Vector is a schematic maneuver that a Figure can execute with respect to a Ground. The Figure and the Ground themselves have highly schematized representations within a Vector. That of the Figure is always a single point, but the Ground can have one of several varied schematizations, such as one or more points, or an extent bounded or unbounded at either extreme. The purpose of the Conform is to represent the schematized version of the Ground (the “Ground Schema”) to the actual Ground.

Chapter 3 introduced only a single Conform () , where the Ground Schema is a point and the Conform places that point in the interior of the Ground. Combining  with the Vector MOVE TO yields the meaning of English “into”. Similarly, it yields the meaning of “out of” when combined it with MOVE FROM, and “through” with MOVE VIA.

Talmy (ibid.) makes no claims about the universality of Conforms and makes no effort to enumerate them. A notion such as containment (as invoked in ) is likely to be widely attested across languages, and support seems a likely candidate as well. For the latter, I offer the symbol , indicating that the Ground Schema is a point supported on the surface of the Ground. Here, though, cross-linguistic differences are easily spotted. English “on”, “onto” and “off of” all are equally





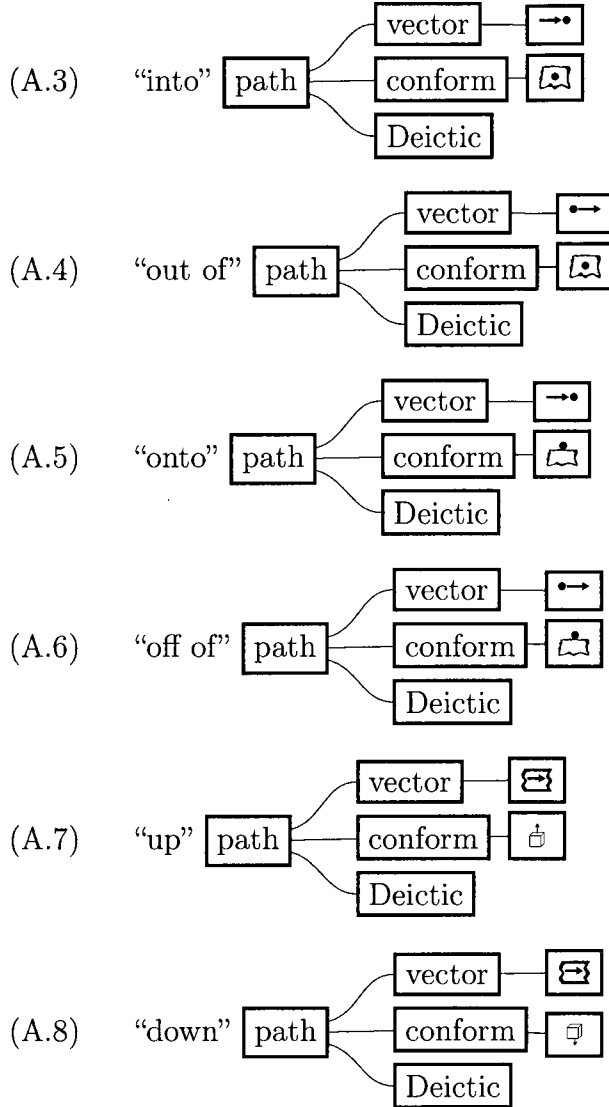
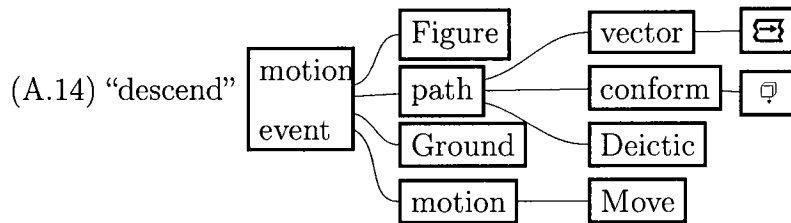
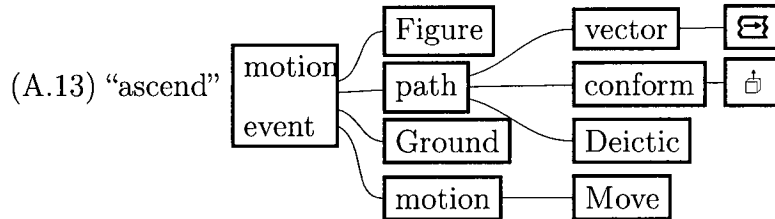
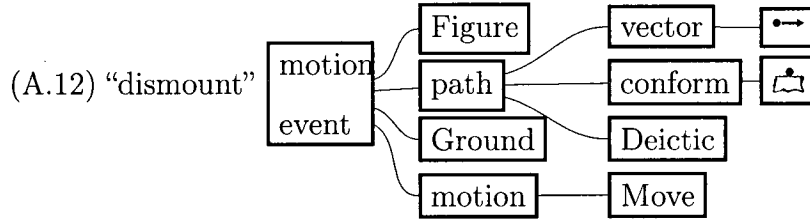
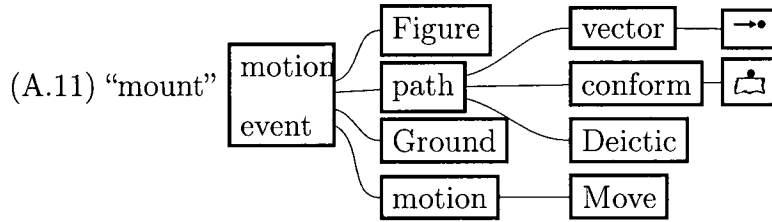
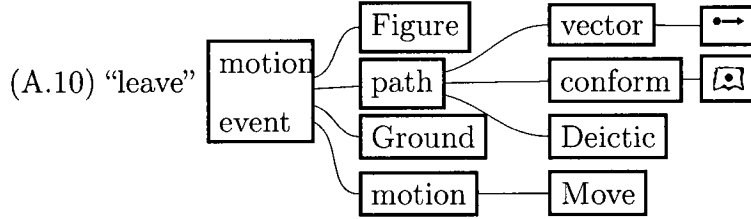
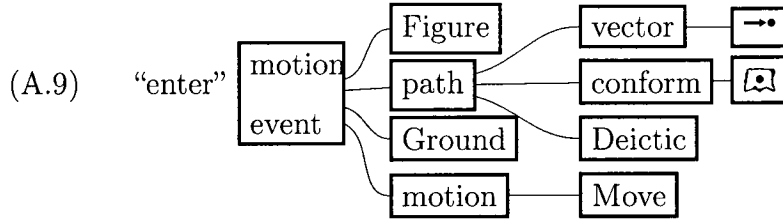
Conform	The schema is ... the Ground
	... a point on the inside of ...
	... a point supported on the surface of ...
	... an extent aligned with the positive vertical axis of ...
	... an extent aligned with the negative vertical axis of ...

Table A.1: A few more Conforms

A.3.1 Prepositions(**Path** \Rightarrow)



A.3.2 Path verbs ($\boxed{S} \Rightarrow$)



BIBLIOGRAPHY

- Carnap, Rudolf (Oct. 1952). “Meaning postulates”. In: *Philosophical Studies* 3.5, pp. 65–73.
- Chomsky, Noam (1956). “Three models for the description of language”. In: *IRE Transactions on Information Theory* 2, pp. 113–124.
- (1963). “Formal Properties of Grammars”. In: *Handbook of Mathematical Psychology*. Ed. by R. D. Luce, R. Bush, and E. Galanter. Vol. 2. New York: Wiley, pp. 323–418.
- (1981). *Lectures in Government and Binding: The Pisa Lectures*. Mouton de Gruyter.
- (2002). *Syntactic Structures*. 2nd ed. New York: Mouton de Gruyter.
- Comon, H. et al. (2007). *Tree Automata Techniques and Applications*. Available on: <http://www.grappa.univ-lille3.fr/tata>. release October, 12th 2007.
- Croft, William et al. (2008). “Revising Talmy’s typological classification of complex events.” Draft, available at <http://www.unm.edu/~wcroft/Papers/TalmyTypology-paper.pdf>.
- De Groote, Philippe and Christian Retoré (Aug. 1996). “On the Semantic Readings of Proof Nets”. In: *Formal Grammar*. Ed. by Geert-Jan Kruijff, Glyn Morrill, and Dick Oehrle. Prague: FoLLI, pp. 57–70.
- Engelfriet, Joost (1980). “Formal language theory; perspectives and open problems”. In: ed. by Ronald V. Book. New York: Academic Press. Chap. Some open questions and recent results on tree transducers and tree languages, pp. 241–286.
- Engelfriet, Joost and Sebastian Maneth (2002a). “Hierarchies of String Languages Generated by Deterministic Tree Transducers”. In: *Developments in Language*

- Theory 5*. Ed. by Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa. Vol. 2295. Lecture Notes in Computer Science. Berlin: Springer, pp. 228–238.
- Engelfriet, Joost and Sebastian Maneth (2002b). “Output String Languages of Compositions of Deterministic Macro Tree Transducers”. In: *Journal of Computer and System Sciences* 64, pp. 350–395.
- Engelfriet, Joost and Jan Joris Vereijken (Oct. 1997). “Context-Free Graph Grammars and Concatenation of Graphs”. In: *Acta Informatica* 34.10, pp. 773–803.
- Engelfriet, Joost and Heiko Vogler (1985). “Macro Tree Transducers”. In: *Journal of Computer and System Sciences* 31, pp. 71–146.
- Feder, Jerome (July 1971). “Plex Languages”. In: *Information Sciences* 3.3, pp. 225–241.
- Fodor, J. A. et al. (1980). “Against definitions”. In: *Cognition* 8.3, pp. 263–367.
- Girard, Jean-Yves (1987). “Linear Logic”. In: *Theoretical Computer Science* 50.1, pp. 1–102.
- Grädel, Erich and Martin Otto (1999). “On logics with two variables”. In: *Theoretical Computer Science* 224, pp. 73–113.
- Guhe, Markus (2007). *Incremental Conceptualization for Language Production*. Mahwah, New Jersey: Lawrence Erlbaum Associates.
- Halliday, Michael Alexander Kirkwood (1970). “Language Structure and Language Use”. In: *New Horizons in Linguistics*. Ed. by John Lyons. Pelican Books. Harmondsworth: Penguin Books. Chap. 7, pp. 173–195.
- (1975). *Learning how to mean: explorations in the development of language*. London: Edward Arnold.
- Iordanskaja, Lidija, Richard Kittredge, and Alain Polguère (1991). “Lexical Selection and Paraphrase in a Meaning-Text Generation Model”. In: *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Ed. by Cécile L. Paris, William R. Swartout, and William C. Mann. The

- Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, pp. 293–312.
- Jackendoff, Ray (1996). “The Architecture of the Linguistic-Spatial Interface”. In: *Language and Space*. Ed. by Paul Bloom et al. MIT Press. Chap. 1, pp. 1–30.
- Jackendoff, Ray and Adele E. Goldberg (Sept. 2004). “The English Resultative as a Family of Constructions”. In: *Language* 80.3, pp. 532–568.
- Jakobson, Roman (1962). “Typological Studies and Their Contribution to Historical Comparative Linguistics”. In: *Selected writings*. Vol. 1. Mouton, pp. 523–532.
- Kirsch, R. (1964). “Computer Interpretation of English Text and Pattern Recognition”. In: *IEEE Transactions on Electronic Computers* 13.
- Kühnemann, Armin (1996). “A pumping lemma for output languages of macro tree transducers”. In: *Trees in Algebra and Programming — CAAP ’96*. Ed. by Hélène Kirchner. Vol. 1059. Lecture Notes in Computer Science. Berlin: Springer, pp. 44–58.
- Lamb, Sydney M. (1966). *Outline of Stratificational Grammar*. Washington, D.C.: Georgetown University Press.
- Lambek, Joachim (Mar. 1958). “The Mathematics of Sentence Structure”. In: *The American Mathematical Monthly* 65.3, pp. 154–170.
- Lecomte, Alain (1993). “Towards efficient parsing with proof-nets”. In: *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, pp. 269–276.
- Levelt, Willem J. M. (1989). *Speaking: From Intention to Articulation*. ACL-MIT Press Series in Natural-Language Processing. Cambridge, Massachusetts: The MIT Press.

- Lutz, Rudi (1996). "Recent Advances in Parsing Technology". In: ed. by H. Bunt and M. Tomita. Kluwer Academic Publishers. Chap. 19, pp. 359–383.
- Maneth, Sebastian (1999). "String Languages Generated by Total Deterministic Macro Tree Transducers". In: *Foundations of Software Science and Computation Structures*. Ed. by Wolfgang Thomas. Vol. 1578. Lecture Notes in Computer Science. Berlin: Springer, pp. 258–272.
- Mel'čuk, Igor (1981). "Meaning-Text Models". In: *Annual Review of Anthropology* 10, pp. 27–62.
- (1984). *Dictionnaire explicatif et combinatoire du français contemporain*. Vol. 1. Recherches Lexico-Semantiques. Les Presses de l'Université de Montréal.
- (1988). "Semantic Description of Lexical Units in an Explanatory Combinatorial Dictionary: Basic Principles and Heuristic Criteria". In: *International Journal of Lexicography* 1.3, pp. 165–188.
- Merenciano, Josep M. and Glyn Morrill (1997). "Generation as deduction on labelled proof nets". In: *Logical Aspects of Computational Linguistics, LACL'96*. Lecture Notes in Artificial Intelligence 1328. Springer, pp. 310–328.
- Miller, W. F. and A. C. Shaw (1968). *Linguistic Methods in Picture Processing — A Survey*. Tech. rep. SLAC-PUB-469. Stanford, California: Stanford Linear Accelerator Center.
- Narasimhan, R. (1962). *A linguistic approach to pattern recognition*. Tech. rep. 21. Urbana: Digital Computer Laborator, University of Illinois.
- (1966). "Syntax-directed interpretation of classes of pictures". In: *Communications of the ACM* 9.3, pp. 166–173.
- Pfaltz, John L. and Azriel Rosenfeld (1969). "Web Grammars". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. Ed. by Donald

- E. Walker and Lewis M. Norton. The MITRE Corporation. Bedford, Massachusetts, pp. 609–619.
- Pogodalla, Sylvain (Apr. 2000). “Generation in the Lambek calculus framework: an approach with semantic proof nets”. In: *Proceedings of the first conference of the North American chapter of the Association for Computational Linguistics*. Vol. 4. ACM International Conference Proceeding Series. Seattle, pp. 70–77.
- Pratt-Hartmann, Ian (2003). “A Two-Variable Fragment of English”. In: *Journal of Logic, Language and Information* 12.1, pp. 13–45.
- (2005). “Complexity of the Two-Variable Fragment with Counting Quantifiers”. In: *Journal of Logic, Language and Information* 14.3, pp. 369–385.
- Prince, Alan and Paul Smolensky (1993). *Optimality Theory: Constraint Interaction in Generative Grammar*. Tech. rep. RuCCS-TR-2; CU-CS-696-93; ROA-537. Rutgers University.
- Reiter, Ehud and Robert Dale (2000). *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.
- Roorda, Dirk (May 1992). “Proof Nets for Lambek Calculus”. In: *Journal of Logic and Computation* 2.2, pp. 211–231.
- Stabler, Edward (1997). “Derivational Minimalism”. In: *Lecture Notes in Computer Science* 1328, pp. 68–95.
- Svenonius, Peter (2006). “The Emergence of Axial Parts”. In: *Nordlyd: Tromsø Working Papers in Linguistics*. Ed. by Peter Svenonius and Marina Pantcheva. Vol. 33. 1. Tromsø: CASTL.
- Talmy, Leonard (2000a). *Concept Structuring Systems*. Vol. 1. Toward a Cognitive Semantics. MIT Press.
- (2000b). *Typology and Process in Concept Structuring*. Vol. 2. Toward a Cognitive Semantics. MIT Press.

- Teich, Elke (1999). *Systemic Functional Grammar in Natural Language Generation: Linguistic Description and Computational Representation*. Communication in Artificial Intelligence. London and New York: Cassell.
- Wadler, Philip (1993). “Mathematical Foundations of Computer Science 1993”. In: vol. 711. Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. Chap. A taste of linear logic, pp. 185–210.
- Zubizarreta, Maria Luisa and Eunjeong Oh (2007). *On the syntactic composition of manner and motion*. Linguistic inquiry monographs. MIT Press.