INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

When *Push* Comes to *Shove*: A Computational Model of the Role of Motor Control in the Acquisition of Action Verbs

David R. Bailey

TR-97-041

October 1997

Abstract

Children learn a variety of verbs for hand actions starting in their second year of life. The semantic distinctions can be subtle, and they vary across languages, yet they are learned quickly. How is this possible? This dissertation explores the hypothesis that to explain the acquisition and use of action verbs, motor control must be taken into account. It presents a model of embodied semantics—based on the principles of neural computation in general and on the human motor system in particular—which takes a set of labelled actions and learns both to label novel actions and to obey verbal commands. A key feature of the model is the *executing schema*, an active controller mechanism which, by actually driving behavior, allows the model to carry out verbal commands. A hard-wired mechanism links the activity of executing schemas to a set of linguistically important features including hand posture, joint motions, force, aspect and goals. The feature set is relatively small and is fixed, helping to make learning tractable. Moreover, the use of traditional feature structures facilitates the use of *model merging*, a Bayesian probabilistic learning algorithm which rapidly learns plausible word meanings, automatically determines an appropriate number of senses for each verb, and can plausibly be mapped to a connectionist *recruitment learning* architecture. The learning algorithm is demonstrated on a handful of English verbs, and also proves capable of making some interesting distinctions found crosslinguistically.

When *Push* Comes to *Shove*: A Computational Model of the Role of Motor Control in the Acquisition of Action Verbs

by

David Robert Bailey

B.S. (Cornell University) 1990

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

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA, BERKELEY

Committee in charge:

Professor Jerome A. Feldman, Chair
Professor George Lakoff
Professor Robert Wilensky

Fall 1997

The dissertation of David Robert Bailey is approved:

Chair Date

Date

Date

University of California, Berkeley

Fall 1997

When *Push* Comes to *Shove*:
A Computational Model of the Role of Motor Control in the Acquisition of Action Verbs

by
David Robert Bailey

Abstract

When *Push* Comes to *Shove*:
A Computational Model of the Role of Motor Control
in the Acquisition of Action Verbs

by

David Robert Bailey
Doctor of Philosophy in Computer Science
University of California, Berkeley

Professor Jerome A. Feldman, Chair

Children learn a variety of verbs for hand actions starting in their second year of life. The semantic distinctions can be subtle, and they vary across languages, yet they are learned quickly. How is this possible? This dissertation explores the hypothesis that to explain the acquisition and use of action verbs, motor control must be taken into account. It presents a model of embodied semantics—based on the principles of neural computation in general and on the human motor system in particular—which takes a set of labelled actions and learns both to label novel actions and to obey verbal commands. A key feature of the model is the *executing schema*, an active controller mechanism which, by actually driving behavior, allows the model to carry out verbal commands. A hard-wired mechanism links the activity of executing schemas to a set of linguistically important features including hand posture, joint motions, force, aspect and goals. The feature set is relatively small and is fixed, helping to make learning tractable. Moreover, the use of traditional feature structures facilitates the use of *model merging*, a Bayesian probabilistic learning algorithm which rapidly learns plausible word meanings, automatically determines an appropriate number of senses for each verb, and can plausibly be mapped to a connectionist *recruitment learning* architecture. The learning algorithm is demonstrated on a handful of English verbs, and also proves capable of making some interesting distinctions found crosslinguistically.

Professor Jerome A. Feldman
Dissertation Committee Chair

Contents

List of Figures

Acknowledgements

As work on this thesis comes to a close, I am reminded of this quote from Terry Regier's dissertation, which really hits the nail on the head:

> I have flipped and flopped a number of times as regards my general outlook on the work reported here. At times I've been frustrated purple by my inability to make any substantive headway through the morass; at other times I was convinced that any idiot could do it. And in the worst of times, these two outlooks co-occurred. More recently, though, I have settled in to something approaching a very modest satisfaction.

Through it all, I've gotten by with a little help from my friends. My greatest debt is owed to my advisor Jerry Feldman, with his remarkable combination of broad knowledge, uncanny intuition, thoughtful guidance, open-mindedness, and considerable patience. Srini Narayanan, fellow graduate student, has been an invaluable resource and a convenient target to bounce ideas off; I hope I've returned the favor adequately. And my appreciation also goes to Terry Regier, whose thesis framework provided the model for my own.

The Neural Theory of Language group at ICSI provided a stimulating working group for the duration my graduate student days, including the above folks as well as George Lakoff (whose linguist's perspective and infectious enthusiasm got me into this territory in the first place), Nancy Chang (from whose amazing editing skills I have greatly benefited), Jonathan Segal, Dean Grannes, Andreas Stolcke, Collin Baker, Lokendra Shastri, Ben Gomes and David Stoutamire. Eve Sweetser and Len Talmy helped refine my necessarily-amateur linguistic analyses.

I also thank my other committee members, Robert Wilensky and Dan Slobin, for taking the time to critique this work from their different perspectives.

Much intuition has come from crosslinguistic data cheerfully provided by Masha Brodsky, James Chan, Nikki Mirghafori, D. R. Mani, and Srini Narayanan. And a special thanks to Carol Bleyle who put much effort into gathering data from her ESL students. Jane Edwards kindly provided me with child data on a number of verbs.

Graduate studentdom is not without its bureaucratic hurdles, and Kathryn Crabtree's talents in this regard, and good humor, have smoothed the bumps quite nicely. She is an underappreciated jewel in the Computer Science Division.

Digging back a little further, I'd like to acknowledge two professors from my Cornell undergraduate days: David Gries taught me rigorous thinking, and Devika Subramanian

kindled my interest in artificial intelligence.

I thank the Center for Human Modeling and Simulation at University of Pennsylvania for making the *Jack* animation package available. Kudos, too, to the rag-tag fleet of hackers around the globe who created the Linux operating system and made it possible for me to work as productively at home as at the office.

Encouragement, some ideas, and constant demonstrations that there is more to life than a thesis, were generously provided by my wife Lisa Leinbaugh, parents Robert and Cheryl Bailey, and in-laws Mary and Dennis Leinbaugh. I'd like to thank them for their patience and remind them that they are my real measure of happiness in this life.

Chapter 1

Overview

"Of the above possible fields the learning of languages would be the most impressive, since it is the most human of these activities. This field seems however to depend rather too much on sense organs and locomotion to be feasible."
—Alan Turing, 1948

How do children learn to use verbs such as *push* and *pull?* They are able to do so after hearing just a few examples, even though different languages classify actions quite differently. The answer to this tantalizing question, we believe, is that the common substrate of the human motor control system drives children's rapid yet flexible acquisition of the lexicon of action verbs in their native language. This hypothesis is explored by building a computational model of motor control and word learning, and testing its acquisition of the relevant vocabulary from a range of the world's languages.

As an interdisciplinary endeavor this dissertation addresses a wide audience, ranging from the artificial intelligence community to linguists, psychologists and neurobiologists. Accordingly, the material has been organized so that each chapter fully discusses one aspect of the model, including motivation, representations and algorithms, connectionist account, cognitive implications, limitations and extensions. For the sake of the computer scientist who wishes to cut to the chase, the core implemented computational ideas are always grouped into a single block within each chapter (usually in a single section). Machine learning experts may wish to skip directly to Chapter 6 and Chapter 8.

Chapter 2 begins by motivating the verb acquisition problem in detail. We see that the English vocabulary for hand actions is quite rich, and furthermore that other

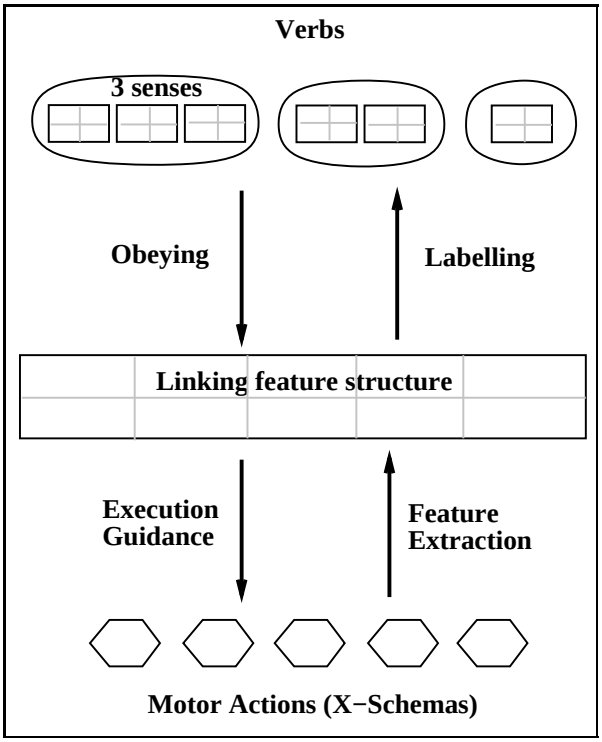


Figure 1.1: Top-level architecture of the verb learning model.

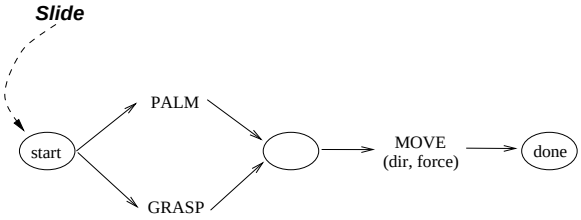
languages of the world classify hand actions in significantly different ways. In that chapter we present our philosophical stance toward the question of how children deal with this variety: bodily grounding of semantics constrains possible meanings. In this framework we pose a specific computational task: Given a set of pairings of actions and verbs, learn the bidirectional mapping between the two so that future actions can be labelled and future commands can be acted out.

Figure 1.1 shows the architecture of our cognitive and computational model of action verb acquisition. As does the dissertation as a whole, the overview will proceed "bottom-up" through this diagram, and then move on to learning. For concreteness, a simplified running example will be developed during the overview.

Given the command-obeying component of the task, it is obvious that our solution must include an active motor control mechanism, not just passive descriptions of actions. **Chapter 3** presents *executing schemas* (*x-schemas* for short, shown at bottom of Figure 1.1), our model of high-level motor control, where synchronization and parameter-

ization of lower-level motor synergies are the key issues. Consequently, the model makes the strong claim that details of lower-level motor control are not linguistically relevant. X-schemas are described using the Petri net formalism (Murata 1989), allowing natural expression of concurrency and asynchrony. Generally there is a 1-to-1 mapping between x-schemas and goals. X-schemas for a variety of object manipulation tasks such as sliding and lifting are developed. For example:

Example: A very simple x-schema for sliding an object on a tabletop might look like this:



This x-schema begins by either grasping the object or placing the palm against it, and then proceeds to move it in a given direction with a given force.

Interfacing the execution of these x-schemas to language (bidirectionally) is accomplished by a set of special features called the *linking feature-structure* (*linking f-struct* for short, shown in center of Figure 1.1), described in **Chapter 4**. The linking f-struct essentially summarizes the execution of an x-schema as a collection of features. A key consequence of the overall architecture is that all linguistically relevant aspects of x-schemas must be represented in these features, which play the crucial role of further restricting the hypothesis space so as to render verb learning tractable (in terms of both computation time and the number of training examples needed). In particular, the summarizing nature of the linking f-struct allows the verb learning algorithm to avoid dealing directly with the time-varying activity of x-schemas. Linking features include the name of the executed x-schema (indicating intention), parameters such as force or direction, control flow patterns such as loop repetition, and perceptual information necessary to guide action. For example:

Example: Linking features, derived from our example x-schema and others like it, might include the following:

schema	posture	direction	force
slide, depress	palm, grasp, index	away, toward, up, down	low, med, high

Each column represents a feature. The upper box names the feature, and the lower box lists possible values. Note that features can range from the x-schema name, to choices of hand posture, to parameters of the primitive synergies such as direction and force.

Next we turn to the semantic representation of verbs, the topic of **Chapter 5**. Since the model must be able to re-create appropriate actions for a given verb, we cannot just represent the minimal abstractions needed to distinguish one verb from another. Rather we need to capture a richer "gestalt" representation, which is accomplished by a conjunction of linking features called a *word sense f-struct*. In a word sense f-struct each feature can be associated with multiple possible values, with varying strengths of association. This permits graded judgments and hence an ability to generalize. By treating these degrees of association as probabilities, well-known statistical techniques can be brought to bear on the problem. Sometimes, the uses of a verb are too varied to sensibly encode as a single (albeit probabilistic) conjunct. For these cases, we employ *multiple senses* for a single word, as shown at the top of Figure 1.1. Labelling an action involves choosing the word sense f-struct which most closely matches the linking f-struct resulting from the action, and emitting the corresponding verb. Conversely, obeying a command involves choosing, for the given verb, a word sense f-struct whose features fit best with the current world state, and then copying the word sense f-struct's features into the linking f-struct in order to guide x-schema execution. Chapter 5 evaluates this model with respect to notions of human categorization including prototype effects, basic-level effects and radial categorization. An example of the word sense representation follows:

Example: Simplified representations for some senses of the verbs *push* and *pull* might look like this:

PUSH: 2 senses

sense 1

schema		posture		direction	
slide	100%	palm	60%	away	50%
touch	0%	grasp	10%	toward	5%
		index	30%	up	15%
				down	30%

sense 2

schema		posture		force	
slide	0%	palm	85%	low	10%
touch	100%	grasp	5%	med	30%
		index	10%	high	60%

PULL: 1 sense

schema		posture		direction	
slide	100%	palm	20%	away	10%
touch	0%	grasp	80%	toward	70%
		index	0%	up	10%
				down	10%

In use, the first sense of *push* generates sliding actions which usually use the palm but in a suitable context might use the index finger, and which tend to be directed away from the body or downward. The second sense generates actions such as pushing on a wall, in which there is no motion but instead a steady application of medium to high force, almost always involving the palm posture. The single *pull* sense shown here generates sliding actions toward the body using a grasp. In recognition mode, an occurrence of one of these three prototypical actions would strongly activate the corresponding sense, leading to production of the appropriate verb. Other actions would weakly activate multiple senses, in which case a verb is produced only if the winner's activation exceeds a threshold.

Finally we arrive at the core of the dissertation: the learning algorithm. Since much of the model's structure (namely the x-schemas and linking features) is specified before learning, the learning process involves only generating an appropriate set of word sense f-structs given the data. **Chapter 6** begins with a review of the lexical acquisition literature, from which three important constraints are taken: children learn to label their *own* actions, do so with little negative evidence, and exhibit fast mapping (learning from as few as one example). This leads to the choice of *Bayesian model merging* for our learning algorithm. One key property of this statistical approach to achieving good generalization ability is the use of a Bayesian criterion which explicitly specifies the trade-off between (1) a preference for a small number of word senses, and (2) the ability of a larger number of senses to more accurately represent the training data. Another key property of model merging is that it captures "fast mapping" because new words are immediately modelled by a word sense which essentially copies the training example.

Example: As instances of pushing and pulling occur, a new sense is initially created for each, closely matching the instance. In accordance with the formulas presented in Chapter 6, similar senses are then merged until only the three senses described above remain. Their probability tables reflect the instances merged to form them. The two senses of *push* do not merge because if they did, important correlations—e.g. that sliding pushes often use the index finger posture but touching pushes rarely do—would be lost.

A sketch of a connectionist network version of the model, including this learning method, is presented in parts of Chapters 3 through 6, though it is mostly unimplemented. Recruitment learning, conjunctive "triangle units" and winner-take-all connectivity provide the mechanisms needed to implement the model.

Chapter 7 presents a way to extend the model to handle *verb complexes*, by which we mean a verb plus any inflections, particles or auxiliaries which help to specify the action. Essentially this involves grouping the word sense f-structs into separate *slots* for each grammatical position, and extending the labelling and obeying algorithms to execute within each slot. When a verb complex is given as a command and its component words specify conflicting feature values, the word with the more selective probability distribution is preferred. Interestingly, the learning algorithm can remain unchanged. However, a useful heuristic is implemented to encourage the system to learn the typically relevant features in each slot to speed learning of new words.

Example: Assume we have several x-schemas for moving objects like the SLIDE x-schema described earlier, and we label actions with two words: a verb and a directional modifier. The resulting representation for a modifier like *up* might look like this:

UP: 1 sense

schema		posture		direction	
slide	10%	palm	35%	away	0%
touch	10%	grasp	35%	toward	0%
depress	10%	index	30%	up	100%
...				down	0%

The word codes very selectively for the upward direction and hence will override any weaker directional correlations in verbs such as those shown earlier. *Push up* therefore generates an action directed upward, not away.

The model has been tested on a variety of verbs from languages such as English, Farsi, and Russian. **Chapter 8** surveys these results, first with English verbs and then crosslinguistically. Sensitivity to the several learning parameters is discussed, as well as

some categories which are not learnable by the model.

Finally, **Chapter 9** discusses the implications of our model, addresses some objections, points out new questions it raises, and discusses related efforts applying the x-schema formalism to other domains. We conclude with some thoughts on real world uses of this work.

Chapter 2

Setting the Stage

"It takes ... a mind debauched by learning to carry the process of making the natural seem strange, so far as to ask for the why *of any instinctive human act."*
— William James

Historically, most of the effort in analyzing language has focused on its generative capacity—how words are combined—treating the meanings of individual words as a comparatively simple problem. This chapter argues that the issue of lexical semantics is itself subtle enough to warrant computational modelling, and proposes a methodology for the particular case of action verbs.

2.1 A Crosslinguistic Conundrum

Have you ever considered the verbs you use to describe actions you perform with your hands? Many people are surprised by the number of such verbs, and the subtle distinctions they make. Consider the following list, which is far from complete:

get, seize, snatch, grab, grasp, pick (up), take, hold, grip, clutch, put, place, lay, drop, slam, release, let go, move, push, pull, shove, yank, slide, bat, flick, tug, nudge, lift, raise, hoist, lower, pass over, lob, toss, throw, fling, whip, chuck, hit, tap, rap, bang, slap, press, poke, punch, rub, shake, pry, turn (over), flip (over), tip (over), rotate, spin, twirl, handle, squeeze, pinch, tie, twist, bend, bounce, scrape, scratch, scrub, smear, crush, smash, shatter, scatter, spread (out/on), cut, slice, clip, wipe, brush, grind, tighten, loosen, open, close, insert, remove, hook, hang, balance, peel, (un)wind, dunk, (un)zip, juggle, knead, dribble, scribble, hand, pass, salute, caress, fondle, pet, pat, stroke, wave, point, hide, stack, touch, feel, reach (for), stop, help, resist, try, bump, slip, knock (over/down)

There are some gross distinctions in meaning (e.g. possession changes *vs.* object movement *vs.* object manipulation) but also considerable variation of a subtler kind which doesn't so easily admit qualitative characterization (e.g. *grab vs. seize*, or *raise vs. lift vs. hoist*, or *fling vs. toss*).

How could children possibly learn all these fine distinctions? Maybe all these concepts are already in the child's mind—either pre-wired, or as a result of maturation or experience—before the child begins to learn language. If so, the verb learning task would amount to a game of mix-and-match between verbs and concepts—a comparatively easy task. This has been proposed by Nelson (1973).

But we will argue in this thesis that this can't be the case. A few examples of some conceptual distinctions made in other languages of the world should convince you. The following examples are from our own informal crosslinguistic survey of languages including Tamil, Cantonese, Farsi, Spanish, Korean, Japanese and Arabic:

- **THALLU & ILU (Tamil):** These correspond roughly to pushing and pulling; however, they connote a sudden action as opposed to continuous application of force and smooth movement. The only way to get this latter meaning is to suffix a directional specifier. Thus there is no way to indicate smooth pushing in an arbitrary direction.

- **HOL-DAADAN & FESHAAR-DAADAN (Farsi):** These correspond to two different senses of *push*. *Hol-daadan* refers to moving an object away from oneself. (It is actually closer to *shove* as it implies high force; there is in fact no word for gentle or continuous pushing, other than the generic *move* verb with a directional specifier.)

In contrast, *feshaar-daadan* refers to applying steady pressure to an unmoving object, e.g. pushing on a wall.[1]

- **PULSAR & PRESIONAR (Spanish):** These verbs correspond to English *press*, but they make a distinction based on hand posture. *Pulsar* refers to pressing with a single finger, while *presionar* refers to pressing with the entire palm.

- **PUDI (Tamil):** *Pudi* covers both obtaining an object, as well as continuing to hold an object. It connotes quickness in the first case, and exertion of force in the second case. It prefers the use of either a cupped palm supporting the object, or else a closed fist. Close English verbs are *catch*, *clutch* and *restrain*.

- **ZADAN (Farsi):** This word refers to a large number of object manipulations whose common character seems to be the use of quick motions. The prototypical *zadan* is a *hit*ting action, though it can also mean to *snatch* (*ghaap zadan*), or to *strum* a guitar (or play any musical instrument, for that matter!).

- **MEET (Cantonese):** This verb covers both pinching and tearing. It seems to connote forceful manipulation by two fingers, yet it is also acceptable for tearing larger items where two full grasps are used.

- **DROP:** Neither Tamil nor Cantonese has a verb for gentle dropping. Both languages instead possess one verb for grip-release (i.e. *let go*) which does not specify whether the released object is otherwise supported, and another verb for throwing down, which connotes use of force.

- **VAIIE & PODU (Tamil):** Both verbs can refer to *put*ting an object down. Carefully executed puts which ensure the object is placed securely use *vaiie*, although this verb is perhaps closer to English *keep*, since its prototypical case refers simply to maintaining an object in a given location, without expenditure of effort. Meanwhile *podu* connotes a careless put—indeed it includes throwing the object down. There does not seem to be an equivalent of *place*, connoting gentleness but focusing on the relocation of the object.

[1] *Daadan* means to give. *Hol* is a noun for an outward movement, but it is not used alone. *Feshaar* is a commonly used noun for pressure.

2.2 It's the Body, Stupid!

We have seen that languages are quite rich in verbs of hand action and also seem to vary widely. Yet children learn those verbs in their native tongue from a modest number of samples and quickly generalize their words correctly (or near correctly). How?

The answer explored in this dissertation is that the potential variety of lexicalized action categories is not infinite, but instead is constrained by virtue of being grounded in the workings of the human motor control system. Properly construed, this grounding greatly restricts the size of the hypothesis space for verb acquisition, rendering it tractable.

In some sense it is an undeniable and obvious claim that language must ultimately be bodily grounded, since it is a human activity. What this dissertation attempts to do is to answer the question of how this grounding is important for computational models of language. As I see it, there are three main aspects to bodily grounding as it applies to action verbs: (1) neural constraints on information processing algorithms; (2) simple facts about the structure of the body (e.g. arm, hand, five fingers, etc.); and (3) organizing principles of the motor control system (e.g. discrete coordination of simple synergies). These topics will be the theme of the dissertation.

Upon reflection, it's not surprising that details of the motor system are implicated in semantics: while abstract representations at the level of "CAUSE(POSSESS(x))" (such as the conceptual dependency representation of Schank (1975)) may be useful for reasoning, they clearly are inadequate for actually *performing* the action via the arm and hand. In this thesis we will look at representations that do support driving actual behavior.

The following excerpt from Webster's Ninth New Collegiate Dictionary corroborates the view that motor control plays a central role in making some of the finer distinctions in English:

> "TAKE, SEIZE, GRASP, CLUTCH, SNATCH, GRAB mean to get hold of by or as if by catching up with the hand. TAKE is a general term applicable to any manner of getting something into one's possession or control; SEIZE implies a sudden and forcible movement in getting hold of something tangible or an apprehending of something fleeting or elusive when intangible; GRASP stresses a laying hold so as to have firmly in possession; CLUTCH suggests avidity or anxiety in seizing or grasping and may imply less success in holding; SNATCH suggests more suddenness or quickness but less force than SEIZE; GRAB implies more roughness or rudeness than SNATCH."

Most of these distinctions could be made based on features such as speed, force, security of grip, and precision of motion. Going back to the crosslinguistic examples from the previous section, it is clear that these distinctions, too, can usually be made in terms of motor control features, broadly construed to include both goals and those aspects of world state which are directly relevant to carrying out actions.

The influence of motor control and intentional activity in general on theories of conceptual representation has a long history. Piaget is perhaps the best-known advocate, having developed a comprehensive theory of the development of abstract concepts via interaction with the world. Pinker, also, points out that children must certainly attend to internal (and hence externally unobservable) variables such as goals in determining word choice, since often two different words are uttered in the same world state. And Landau & Gleitman (1985) show that blind children learn language more or less normally despite the absence of what is often assumed to be the primary semantic source—vision. For example, they found that blind children readily learn the verb *look*. But in their own behavior, it translates to haptic exploration (that is, using the sense of touch). The core meaning of the verb, they suggest, is "explore with the primary sense," certainly an action-oriented meaning. More recently, brain imaging studies (Damasio & Tranel 1993) have made a strong case for the intimate connection between language and sensorimotor areas of the brain: verbs activate motor control regions, while nouns do not.

The study of embodied cognition generally is not a new enterprise. Johnson (1987) argues that human conceptualization is "imaginative" in the sense that our concepts tend to reflect biases resulting from the human condition—whether perceptual or having to do with the kinds of goals which people seek to achieve—rather than purely reflecting an objective structure to the external world. Lakoff (1987) also argues persuasively for a "non-

objectivist" basis for semantics. Ultimately, embodiment must be explained in terms of neural structures. The role of connectionist modelling in this dissertation will be discussed shortly (§2.4).

To be sure, only a subset of human concepts are directly bodily grounded. However, as Lakoff & Johnson (1980) further argue, these bodily concepts frequently underlie more abstract concepts metaphorically. A computational account of metaphor consistent with this thesis has been developed by Srini Narayanan and is discussed in §9.5.

The best-known example of the study of embodied cognition is work on basic color terms, and we mention it here to make clear the kind of story we wish to tell. Berlin & Kay (1969) show that while languages differ considerably in their "basic" terms for colors, there is an underlying pattern. In particular, given the *number* of basic color terms in a language, one can reliably predict what they will be. And for languages with fewer basic color terms (which thus cover wider ranges of the spectrum), the set of prototypes for each color corresponds to the basic color terms of the richer languages. Why should this be? The punchline of the story is that in later work (Kay & McDaniel 1978), the spectral characteristics of some of the prototypes were found to be predictable from the physiology of the visual system, which suggests why they might be so nearly universal. Recently, this work has been addressed in a computational framework by Lammens (1994). Lammens was disturbed that the earlier accounts could not explain non-spectral colors like brown or white (the latter would activate *all* the prototypes!), and built a computational model to investigate learning of colors. A key result of the work was that the correctness of the learned categories depended upon the choice of color-space with which to represent the light collected by the camera. With a cognitively inspired color-space, reasonable learning results were obtained using an optimization procedure which fit a multi-dimensional Gaussian to each color so as to maximize response to examples of the color while minimizing response to examples of other colors.

To avoid confusion, we point out that our notion of embodiment is somewhat different from that of Brooks (1986) and others in the autonomous robotics community, who emphasize the need for physically realizing robots in order to make progress in robotic control. For them, embodiment primarily means confronting the details of the "real world" such as sensor errors and effector failures. These concerns are certainly an important component of embodiment, and indeed the design of our model of motor control is partially

driven by such concerns. However, the perspective on embodiment taken here is that these issues are secondary; the focus is instead upon the details of the structure of the human body and the principles of neural computation.

With this mindset, then, I set out to find a suitable representation for motor control and to see how it partially determines the course of action verb acquisition.

2.3 The Task in Detail

In this study, the actions under consideration are limited to those of a single hand by a person seated at a table, on which there may be zero or one simple geometric objects such as a cube or stick. The verbs studied are limited to those applicable in this world.

The task is to build a computational model which meets the following requirement, for any single natural language:

- **Given:** a training set consisting of pairs, each containing
  - an action (represented by the motor control pattern which generates it)
  - a verb (as an atomic symbol)
- **Produce:** a representation for the verb lexicon which allows
  - appropriate labelling of novel actions
  - appropriate obeying of verbal commands in novel world states

To the extent possible, the trajectory of learning should reflect the child's.

The system should also be able to handle "verb complexes" such as *keep pushing left.*

Since actions are represented as motor control activity, there is a methodological question of how to present them to native speakers in order to collect labels for the training data. Ideally, an animation software package, such as *Jack* from Transom Technologies, would be used to translate model internals into an on-screen depiction of the corresponding action. Informants viewing the animation could then confidently label actions as well as evaluate the obeying of verbal commands. However, due to some difficulties in interfacing *Jack* to the verb-learning software system we have developed, various shortcuts were used in

the experimental work reported here. In some cases, the author's knowledge of the internals of the model was used to physically demonstrate actions for speakers, or to label training data directly. In other cases, informants were familiarized with the internals of the model. For more details, see §8.1.

By its nature, this computational task forces any solution to strike a balance between two extremes. The learning requirement demands that any solution model strong innate biases. Meanwhile, the crosslinguistic requirement guards against solutions with excessive biases which oversimplify the task actually faced by children.

It is critical to understand that our task definition specifies that the child is assumed to be labelling *his own* actions, and therefore has access to his internal state during the performance of the action, including his intentions. This is in contrast to using visual input (i.e. watching a parent perform the action). Certainly it is a simplification to pretend the child never hears a verb in association with someone else's action rather than his own. But it has been shown that the own-action case is indeed the most frequent (Tomasello 1992). Furthermore, there is impressive evidence that even neonates can map others' actions onto their own motor control system (Meltzoff & Moore 1977), so even for the other's-action case, the language-learning child may be inclined to consider motor parameters as the primary semantic component.

Another simplification in our task is the pre-selection of the time window of activity labelled by the verb. This is done to simplify the problem, but Tomasello (1992) provides evidence that children may have help in this regard, too, since most verb labels are uttered immediately preceding the action, and for early verbs, the actions are usually short in duration.

Many other simplifications have been made to render the task manageable. Linguistic context is not modelled, even though it could contribute to learning individual lexical items. (But see Goldberg (1995) for arguments on the separation of verb meaning and grammatical meaning, and the different nature of verb meaning from grammatical meaning.) The social domain is absent, restricting the vocabulary we can address. We avoid using objects with functional significance beyond their simple spatial qualities, to avoid representing those other qualities. We don't deal with deformable, liquid or jointed objects. Nor do we deal with multiple objects or actions which involve both hands, or tools. We only consider actions which do not involve planning, i.e. those whose "plan" is already wired as a motor

control program (so we don't handle verbs like *stack*). Verbs are an open-class grammatical category, making these somewhat arbitrary restrictions necessary.

Lastly, I would like to emphasize that this project explores only the motoric component of the full semantics of action verbs. This reflects a belief that the motor component is central, but in no way does it represent a claim that the motor component is the full story.

2.4 A Connectionist Commitment

The role of connectionism in this work is very much tied up with the notion of bodily grounding. Neural plausibility provides further strong constraints on how concepts may be represented and learned, and thus should inform serious cognitive modelling efforts.

Yet working at the connectionist level can be cumbersome. In this thesis, it has proved useful to define a number of computational tools which can be mapped to connectionist models, but to do most of the work at the higher, computational level. Indeed, our implementation of the verb learning system is done at this higher level. Throughout the thesis, we will provide sketches of connectionist networks which could implement our representations and algorithms. These sketches are intended to convince the knowledgeable connectionist that they are implementable, but the networks have not been simulated in code.

The important question is, what are the biological constraints which one should respect when creating a connectionist network? What are the criteria for evaluating the neural plausibility of an algorithm? For high-level tasks such as language learning, precise neuronal modelling would be hopeless. What we consider are some broad computational constraints imposed by neural structures in general (Feldman & Ballard 1982), including:

- use of many parallel, simple, slow computing units
- no central controller—local rules only
- no passing of structures between computing units (simple messages only)
- substantially less than full connectivity among computing units

One decision in designing a neural network is whether to represent the "concepts" of the problem domain in a *punctate* manner (Feldman & Ballard 1982) where "grandmother cells" are assigned individually to the concepts[2], or in a *distributed* manner where concepts are represented by the activity of many neurons, each of which participates in many such concepts. The advantages of distributed representations (Rumelhart & McClelland 1986) include graceful degradation and the potential to allow learning algorithms to develop new features, and for these reasons they have been the focus of neural network research. Yet, learning algorithms for distributed representations, such as backpropagation (Rumelhart & McClelland 1986) and its variants, virtually always involve gradual adjustment of weights, rendering them useless for tasks in which one-shot learning is desired. Accordingly, we will focus on more punctate representations, which have the advantages of facilitating more structured design and, as we will see, faster (if less flexible) learning.

Within the connectionist framework, a variety of techniques have been developed (Hertz *et al.* 1991). A few selections from this toolbox will prove useful in demonstrating the neural plausibility of the verb learning model developed in this thesis. At a low level, we will use the notion that the activation level of a connectionist unit can represent an approximate probability, or degree of belief, that the concept it represents is currently applicable. We will also make use of the notion of thresholding, in which evidence for a concept must reach a certain level before the associated unit will fire. At a higher level, winner-take-all organization will be used to select units which best fit data. Recruitment learning (Feldman 1982) provides a weight update rule and network pattern which proves useful for one-shot learning. And lastly, the notion of encoding bindings via temporal synchrony of separate units (Shastri & Ajjanagadde 1993) is used in one connectionist implementation of motor schemas.

2.5 Related Efforts

Ideas from a variety of fields have been borrowed in this work. They are acknowledged along the way. This section highlights a small number of projects which have attempted an overall task very similar to this one, and which therefore invite comparison.

[2]It should be stressed that the punctate model does not demand that concepts are represented by a single biological neuron, but allows for a small, functionally distinct cluster of neurons.

This architecture could be considered an implementation of learning *procedural semantics*, as pioneered by Winograd's SHRDLU system (Winograd 1973). SHRDLU was a question-answering system which operated in the time-honored blocks world domain. The user could, for example, request that the system "Pick up the red block," and the appropriate object identification and action would occur. If the request was ambiguous, the system might reply "Which one?" and could handle the simple response, "The big one." Later, if asked "What are you holding?" it would respond "The big red block." The current work differs from SHRDLU in two important ways. First, it pays considerably more attention to the fine-grained motor details which can be involved in verb semantics. Winograd was not concerned with distinguishing *push* from *shove*, only with providing a set of qualitatively distinct actions and a single verb to map to each. It's not clear, for example, that SHRDLU could be straightforwardly recoded for an arbitrary natural language, since his "procedures" might not correspond to action categories encoded by all languages. This was entirely appropriate, for his focus was at the system level, i.e. on demonstrating how simple models of semantics, along with simple models of parsing and inference, could combine to provide a system capable of understanding full discourse when the domain was suitably limited. The second important difference is that this thesis addresses the problem of learning.

Siskind (1992) built a system to learn to recognize action verbs in visual movies. Among the important contributions of this work is the identification of contact, support and attachment relations as key features in understanding the scenes. A logical notion of events was used to discretize movies into "phases," which is not unrelated to the role played by x-schemas in the model proposed here. Yet difficulties arose from the use of necessary and sufficient conditions as a lexical representation: neither defaults nor focus of attention could be expressed, and the system exhibited brittleness in the face of minor timing variations. Furthermore, the input was strictly visual and thus faced an unnecessarily hard problem compared with our model which includes internal state of the actor.

By far, the project most closely related to this thesis is the dissertation work of Regier (1996). Both projects are part of the Neural Theory of Language Project (formally called "L_0") (Feldman *et al.* 1996), and Regier essentially provided the model for the type of research reported here. Regier built a model of the acquisition of spatial terms based upon features derived from the structure of the human visual system. This structure was modelled

as a connectionist network with subnets for computing orientation relations and center-surround relationships. During learning it would develop new features, such as contact or inclusion, that were necessary for the language it was learning. A backpropagation network then categorized these features into spatial terms such as *in*, *over* or *through* in English. The current thesis borrows from Regier the methodology of training a structured network on lexical items from a variety of languages in order to force a balance between innate and learned structures. However, an important limitation of Regier's system is that it can function only as a recognizer; there is no way for it to produce a visual scene corresponding to a given spatial term. This deficiency, it turns out, was the inspiration for my focus on actions—recognizing them, and carrying them out.

Chapter 3

Executing Schemas for Controlling Actions

This chapter presents an active representation for control of actions. The representation is crucial because in our model it provides the foundation of action verb semantics. In order to best understand the rationale behind the representation, we first briefly review some important properties of human motor control which motivated its design.

3.1 Human Motor Control

Before focusing on the neural aspects of motor control, consider the hand and arm and their behavior apart from neural control. All motion is the result of joint rotation. Muscularly, this rotation is accomplished by adjusting tension in opposing muscles, called flexors (e.g. biceps) and extensors (e.g. triceps), attached to each degree of freedom of each joint. The arm has two joints, the shoulder and elbow. The shoulder is a three-degree-of-freedom ball-and-socket joint, while the elbow can either hinge or pivot. The four fingers each contain two hinge joints with one degree of freedom each, and are attached to the metacarpals—the bones within the palm—via a bi-axial joint with two orthogonal degrees of freedom but no pivoting. Meanwhile the thumb, while possessing one fewer joint, enjoys a very flexible saddle joint connecting at the proximal end of the metacarpal, facilitating opposition. Proprioception—the perception of the body's own state—is accomplished by muscle spindles which measure joint positions (the finger muscles have some of the body's most sensitive position sensors (Napier 1993)) and other types of sensors which measure joint velocity. Various receptors embedded in the skin detect contact, pressure and shear.

Controlling many joints at once is of course an exquisitely complex task, but there are biological design principles which manage the complexity. The key idea is the notion of the *motor synergy* (Bernstein 1967), which is a sub-cortical continuous feedback control circuit for a stereotyped motion, which may be modulated by parameters. The simplest example is the stretch reflex, in which the stretching of either the flexors or extensors (or simulated stretching, such as the doctor's tap on the knee) causes those muscles to contract, to counteract the stretch. This feedback loop involves just two neurons, which extend from the spinal column to the limb. The stretch reflex contributes to maintaining posture and is also a building block for higher-level synergies such as walking. Synergies can also operate across modalities. The flexor reflex rapidly retracts a limb that is experiencing pain. More amazingly, the scratch reflex responds immediately to a localized itch by choosing a suitable end effector (hand or foot), moving it to the needed location, and initiating a cyclic scratching motion.[1] Many types of grasps are encapsulated as synergies. Cutkosky & Howe (1990) catalogs these grasps according to their uses, and argues that motion planning involves discrete choices amongst them.

[1] See Kandel *et al.* (1991: Chapters 37–38) for a review of these and other motor reflexes.

The restriction on arbitrary joint movements implicit in the notion of synergies is evident from experiments in bimanual coordination. Franz *et al.* (1991) has shown, for instance, that subjects cannot draw a square with one hand while drawing a circle with the other. When the two hands do manage to engage in different activities, they often bear certain relationships to each other, such as mirror image activity.

One consequence of the existence of such low-level synergies is that cortical control of activity needs to control many fewer degrees of freedom, since it need specify only the "name" of the synergy and its parameters. While the idea remains controversial, so-called *command neurons* which trigger a synergy by their activation have been located. For the simple stretch reflex, activation and parameterization are accomplished by specification of a single threshold. But even more complicated synergies can be modulated with a small number of parameters. Cat locomotion, for example, can be driven by a single labelled line (i.e. axon) from cortex to a central pattern generator (CPG) which not only controls the speed of the cat's gait, but also induces a switch to a different gait (e.g. trot to gallop) as required to achieve the commanded speed (Shik & Orlovsky 1976) (reviewed in Kandel *et al.* (1991)). Parameters seem to be specified separately from the coordinative structure, and often are encoded in an ensemble of neurons. For example, Georgopoulos (1993) has discovered *population coding* of direction and force in motor cortex of behaving monkeys. According to this scheme, a precise parameter value is specified by the sum of the varying activations of an ensemble of neurons which, individually, are only coarsely selective.

This modularization of low-level continuous control loops allows motor cortex to concentrate on higher-level concerns: the coordination of firing of synergies. What types of coordination are required? The list obviously include sequentiality. Evidence that humans construct a sequential motor plan includes the work of Sternberg *et al.* (1978) on delays in starting or stopping typing sequences depending upon the length of the string to be typed. But the motor control system must also coordinate concurrent actions, as demonstrated by Arbib *et al.* (1987) in the context of preshaping the hand during the movement of the hand toward an object to be grasped. Synergy firing can be based not only on current perceptual input but also on internal state. Central pattern generators are the simplest case of this; the effect of perceptual input is modulated by the state (or phase) of the central pattern generator. Thus high-level controllers are not simple percept→synergy maps. The existence of ballistic synergies (for actions which must be performed too quickly to allow for feedback

control) necessitates a looping mechanism at the coordination level to implement successive refinement. And lastly, the uncertainties of the world demand coordination of "emergency" error-correcting actions with the main sequence of actions. Motor representations with these kinds of capabilities are referred to as *motor programs* and their study constitutes a significant subfield of neuroscience (see e.g. (Pearson 1993)). The supplementary motor area of the cortex appears to be implicated, since its activity increases with action complexity; its activity often precedes activity in primary motor cortex and initiation of movement; and cells in this area have been shown to be sensitive to ordering of actions about to be performed (Tanji 1994).

While most of motor cortex is active only when actions are being carried out (or are about to be carried out), the premotor area is active even when actions are only thought about, including mental imagery or viewing another person acting (in which case the phenomenon is called "mirroring" (Gallese *et al.* 1996; Grafton *et al.* 1996)). The accepted view is that this area is involved in planning action sequences. This fact nicely supports related work (see §9.5) employing x-schemas for reasoning.

In summary, then, the key properties of human motor control which we will capture in our representation are (1) synergies for continuous coordination of muscles during simple actions, (2) limited parameterization of these synergies, and (3) serial, concurrent and asynchronous combination of these synergies to compose complex actions. For the curious reader, Wing *et al.* (1996) provides a compendium of neurobiological and psychological research on hand movements. For a lighter survey on hands ranging from their evolution to left-hand taboos see Napier (1993).[2]

3.2 A Petri Net Model

Several constraints, then, drive the representation of actions described in this section. Logical descriptions are ruled out, since the representations must be able to support real-time control of the actions described. Traditional procedural attachment is ruled out, since "black box" controllers would not support the kind of inference about actions which is required in the language task. An inspectable yet active representation is needed, suggesting

[2]In this work we won't address the issue of which portions of motor control are innate, maturational or learned via experience. Huttenlocher *et al.* (1983) and Gopnik (1981) suggest, however, that lexical development can be partly explained in terms of concurrent learning of language and motor competence.

a state machine formalism.[3] In particular, the Petri net formalism has some desirable properties and is our choice for implementing x-schemas.

3.2.1 Synergies as building blocks

The Petri net formalism requires that there be a set of actions at the lowest level which are essentially atomic. In our x-schemas these will consist of actions which are hypothesized to be controlled by motor synergies as described in the previous section.[4] This set has several properties. First, they form a limited set of distinct actions. Second, since they are atomic, the internal implementation of the primitive actions is irrelevant at the Petri net level. Generally these actions would be expected to be implemented as some form of continuous feedback controller. Third, most synergies have a small number of parameters to modulate their function. The following list includes all the synergies used in the examples in this dissertation:

[3] This idea is not new in robot control, e.g. Brooks (1986).

[4] While the "synergies" proposed here are biologically plausible in some ways (as discussed shortly), they are *not* taken directly from the motor control literature. Indeed, a full characterization of human motor synergies remains an elusive goal.

List of Primitive Synergies

MOVE ARM (direction, force, duration): apply force to move the arm in a feedback-controlled manner

MOVE ARM TO (dest, [via]): ballistically move the arm to a target location, passing through the via point if it is specified

PIVOT WRIST (direction, force, duration): pivot the wrist around the axis of the forearm

GRASP: preshape a circular grasp for holding round or cubic objects

WRAP: preshape a grasp for holding long, thin objects

PALM: preshape the palm for flat contact with an object

PLATFORM: preshape the palm to support an object from below

PINCH: preshape grasp for holding objects between the thumb and index finger

APPLY HAND: close the fingers and/or move the palm until they contact an object which is in front of the hand

TIGHTEN GRIP: increment the gripping force of the fingers

RELEASE: open the hand, terminating any kind of grip

POINT: extend index finger while closing other fingers

While the internals of these primitive synergies are not modelled here, a few points are in order. First of all, note that most of the synergies involve moving a body part into a goal position or orientation. We will assume that invocation of one of these synergies when the body is already in the goal position is allowable and simply produces no motion. This assumption is compatible with several theories of motor control, including the spring model (Latash 1993; Jeannerod 1988) which hypothesizes that movements are generated by simply informing the muscles of their desired tension levels and then allowing the system to

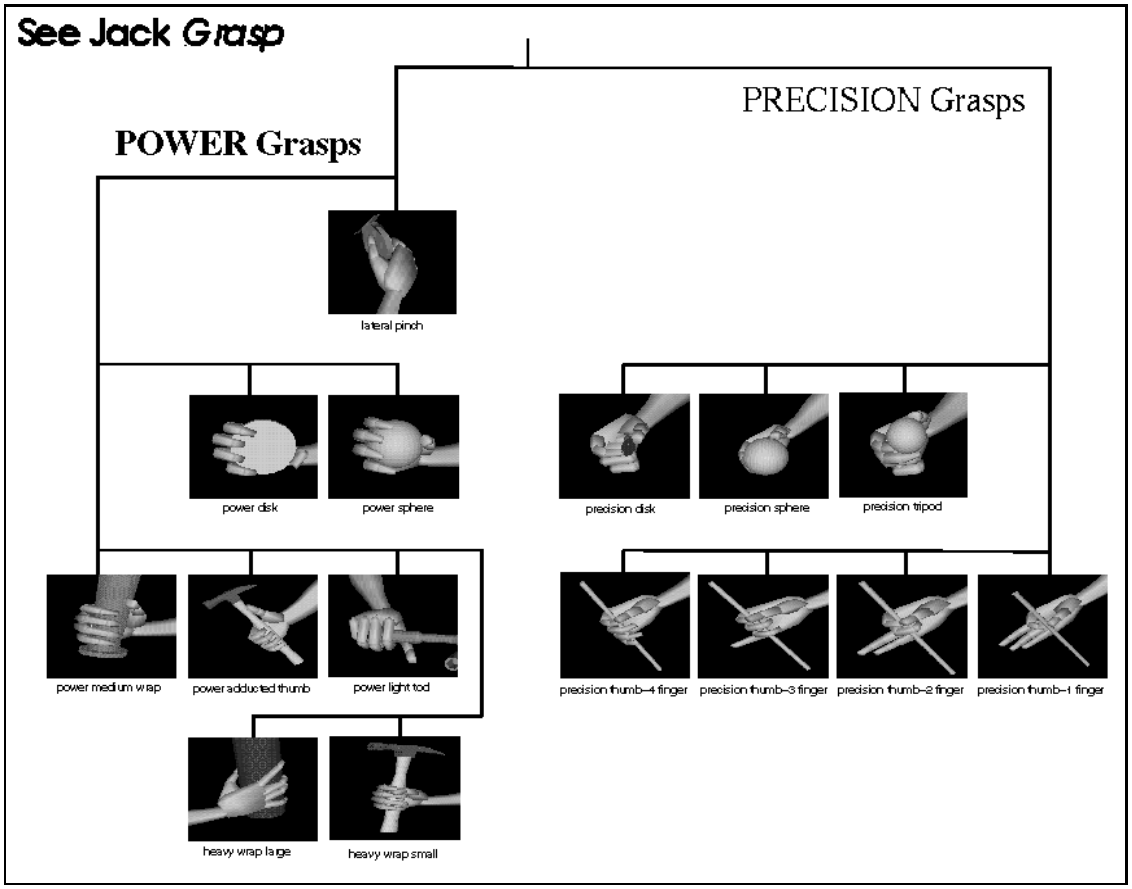


Figure 3.1: A taxonomy of *Jack* grasp synergies. Courtesy of the Center for Human Modeling and Simulation at University of Pennsylvania and Transom Technologies, Inc.

---

relax to this state in accordance with the spring law. It is also compatible with comparator models (Jeannerod 1997: Chapter 6) in which cortex drives the muscles only until (or if) their perceived position differs from the goal position. The MOVE ARM TO synergy also demands a bit of explanation, since it seems to be missing some important parameters such as force or duration. The answer here is that we assume this synergy computes these parameters in accordance with Fitts' Law (Fitts 1954), which is an empirically derived rule relating force and duration to the accuracy requirements of a movement (determined by context or by the specificity of the destination).

Many of these synergies refer to types of grasps. This taxonomy roughly follows that of Cutkosky & Howe (1990) as mentioned in §3.1. It turns out that the *Jack* simulator

also follows this taxonomy, and Figure 3.1 portrays the full taxonomy. Our synergy set uses only a subset of these grasps for simplicity.[5]

Certainly, humans possess many more motor synergies relevant to hand actions. However, this set is large enough to make the points we want to convey.

3.2.2 The Petri net formalism

Executing schemas, or *x-schemas* for short, is the name given to our motor control representation. In the current design they are modeled as Petri nets. Additionally, there is a parameter-passing mechanism which operates in conjunction with the Petri formalism. Each x-schema is designed to achieve a given goal (such as obtaining an object) but may represent multiple ways of achieving the goal, depending on the world state.

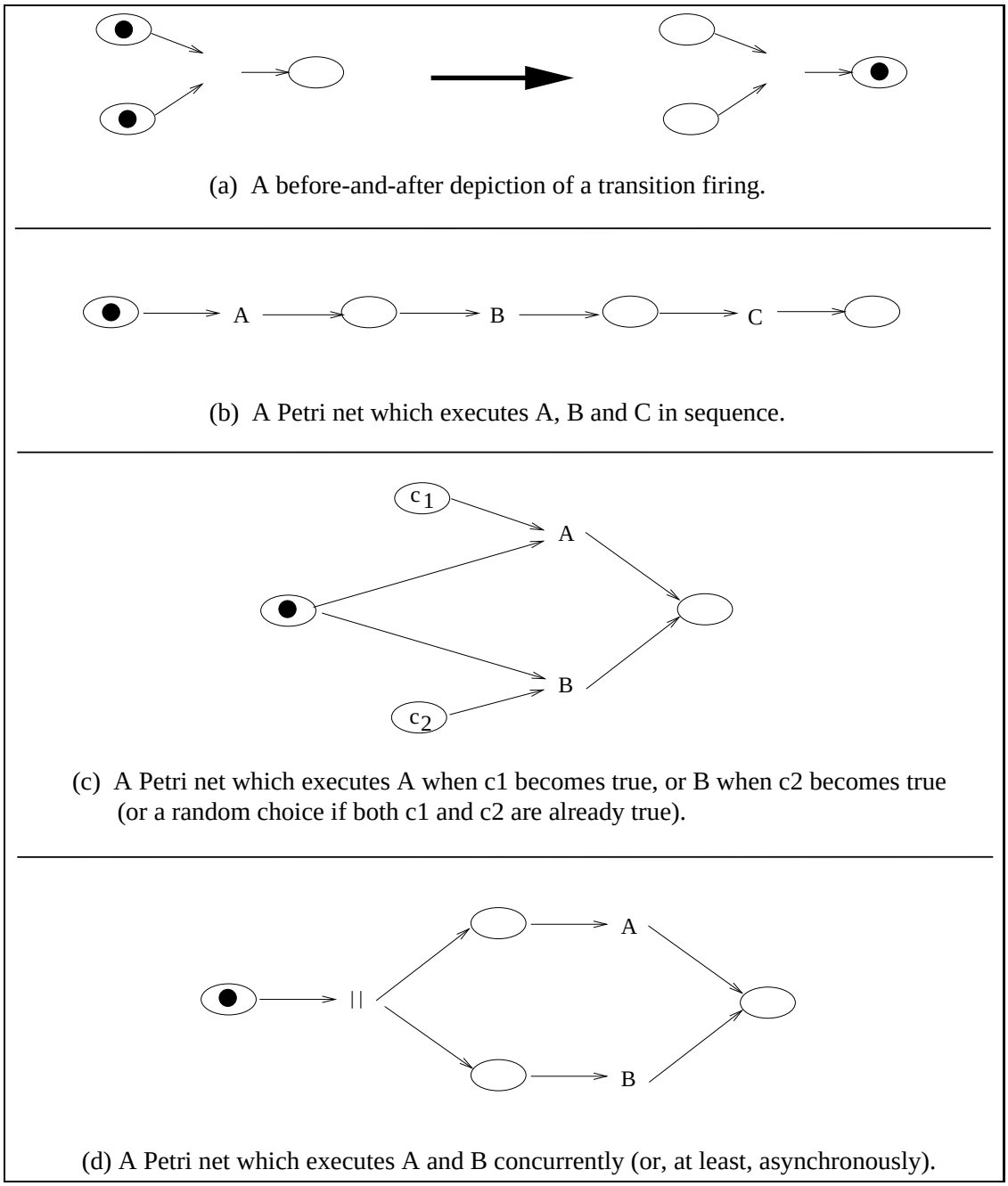
The Petri net formalism (Murata 1989; Reisig 1985) conveniently expresses most of the needed properties for coordination of synergies, including concurrency and asynchrony. A Petri net consists of *places* and *transitions* with directed connections between them. Places may represent either perceived states of the world or internal state, and the current state is indicated by the presence of a *token*. When all of the places with connections to a transition possess tokens[6], the transition is *enabled* and may *fire*, which involves consuming those tokens and then depositing a token in each place with connections from the transition. Figure 3.2(a) shows a before-and-after view of the firing of a single transition. Places are drawn as circles, transitions as rectangles, and tokens as solid dots.

All transitions in a Petri net operate in parallel. There is no global clock, nor do firings get serialized. Each transition fires whenever it becomes enabled. In general, the delay between enablement and firing is unpredictable, although the formalism allows specification of probability distributions on the delays. But we do not use this feature here, and in our x-schemas all delays are assumed to be zero.

In our use of Petri nets, a transition usually represents an action, namely the execution of a primitive motor synergy. The action occurs exactly when the transition fires. These transitions are depicted with the name of the action inside the rectangle. Sometimes,

[5] A more detailed model of grasping could, if necessary, be made. The grasp types above can be decomposed using the opposition space and virtual finger abstractions of MacKenzie & Iberall (1994).

[6] Generally, only one token is required at each input place. Where more than one token is required, the number is indicated next to the incoming arc.



(a) A before-and-after depiction of a transition firing.

(b) A Petri net which executes A, B and C in sequence.

(c) A Petri net which executes A when c1 becomes true, or B when c2 becomes true (or a random choice if both c1 and c2 are already true).

(d) A Petri net which executes A and B concurrently (or, at least, asynchronously).

Figure 3.2: Some common Petri net constructs. (a) shows the simplest case of an enabled transition firing. (b)-(d) show constructs for sequentiality, branching and concurrency.

though, transitions are needed simply to move tokens without any corresponding action. These transitions are left unlabelled.

When assembling these building blocks into working Petri nets, certain patterns of network structure prove particularly useful. For example, sequential actions are a common requirement. The Petri net implementing sequential firing of transitions is depicted in Figure 3.2(b). The places between each pair of transitions serve to pass "control" from one transition to the next. Placing a token in the left-most place, as shown, leads to the action sequence A, B, C and leaves a token in the right-most place. Loops are another common pattern, and are trivial to construct; if the final output arc of Figure 3.2(b) were to connect back to the left-most place, the net would generate the sequence A, B, C, A, B, C, ... Another pattern, branching on perceptual conditions, is slightly more complex and is shown in Figure 3.2(c). Separate places encode the mutually exclusive set of conditions, such as "c1" and "c2". When the token is deposited in the start place, only the transition connected to the currently-true condition fires. If none of the conditions is true, the net suspends until one becomes true. If the conditions are *not* mutually exclusive and more than one is currently true, then one of the two enabled transitions is chosen at random to fire.

Figure 3.2(d) shows how a Petri net can encode concurrency. A transition with no associated action (labelled "||" to indicate concurrency) is used to turn one token at the start place into two tokens. The two tokens simultaneously enable transitions A and B, allowing them to fire simultaneously, or at least in an arbitrary order. Once two tokens arrive in the right-most place, we have a guarantee that both transitions have fired.

Note that some places represent control flow, while others represent input from "outside" the net, i.e. perceptual information, which influences the course of action. This latter type will be discussed shortly.

A simple extension to Petri nets allows weighted connections. A weight on a connection into a transition specifies the number of tokens which must be present in order to enable the transition. A weighted output arc from a transition specifies that multiple tokens are emitted when the transition fires.

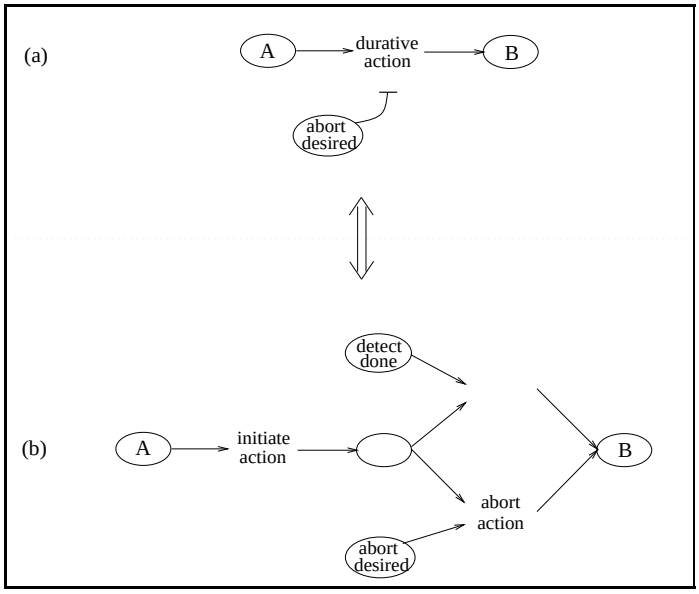


Figure 3.3: Translating durative-action transitions into the standard Petri formalism with instantaneous transitions.

3.2.3 Durative actions

In the Petri formalism, all transition firings are instantaneous, and thus input tokens are effectively consumed at the same time that output tokens are generated. Yet many of the synergies defined in §3.2.1 do not execute instantaneously. In order to represent such *durative actions* as transitions, we must alter the semantics of transitions from the standard Petri model as follows. When a durative transition fires, it consumes its input tokens immediately, but does not deposit its output tokens until its action completes. Furthermore, we allow a special kind of connection from a place to a durative transition which aborts the action-in-progress should a token become available before completion. These connections are drawn with a flat bar at the "tip".

Translating durative transitions back into standard Petri transitions can be accomplished as shown in Figure 3.3. The translation assumes that the durative action can be redescribed as a pair of instantaneous actions which initiate and abort the action, along with a place which detects when the action is done. In some cases the needed detector is already explicitly modelled for other purposes (for example, detecting contact) but in other cases we would need to model new detectors (such as detecting that muscles have reached

their set point). The translated network in Figure 3.3(b) operates as follows. When a token arrives in place A the "initiate" transition fires, invoking the corresponding synergy and *immediately* placing a token in the unlabelled place in the center of the figure which indicates that the action is ongoing. In the usual case, the network then waits until completion of the synergy is detected. At this time the unlabelled transition, "noticing" that the action was ongoing but is now done, fires and deposits a token in place B. However, if there exists an "abort" connection and it receives a token before the synergy completes, then the "stop" transition fires instead, aborting the synergy and depositing a token in place B.

3.2.4 Accessing perceived world state

A central feature of an x-schema is that its execution path can be highly context-dependent, by virtue of having special places which receive tokens from external perceptual sources. In practice, these perceptual places represent fairly high-level properties of the world, as opposed to, say, low-level visual details. Separate, unmodelled perceptual mechanisms are assumed to perform the appropriate computations over perceptual and proprioceptive inputs to generate these high-level properties. Such places have slightly non-standard behavior, in that the token is never depleted; if consumed by a transition, it is instantaneously replaced (unless, of course, the world state changes). These percepts are boolean in nature. They are listed below.

List of Perceptual Places

SMALL object size is small compared to hand

LARGE object size is large compared to hand

RESISTANCE detects a very high force opposing arm motion

ELONGATED object has a large length-to-width ratio

SLIPPING detects a slipping grip

STABLE detects that the object is stably supported

GRIPPING detects that object is already held in grip (any type of grip including palm contact)

AT GOAL detects whether object is at goal location or orientation

Naturally, x-schema transitions can be enabled by either the truth or falsity of these conditions. To allow this, the model has separate places for both the true and false cases. The need for inhibitory connections is thus removed, and the model also gains the capability of representing the "don't know" condition. It is the (unmodelled) perceptual mechanisms' responsibility for ensuring consistency between the two contradictory places.

Not all of the relevant features of the world are boolean, however. X-schemas will need to make calculations, such as computing the force needed to move an object given its weight and a desired acceleration. We posit, therefore, a small working memory where such values are stored. Features contained in this area include:

List of Perceptual Features (Quantitative)

WEIGHT an estimate of the object's weight

OBJLOC a vector indicating object position

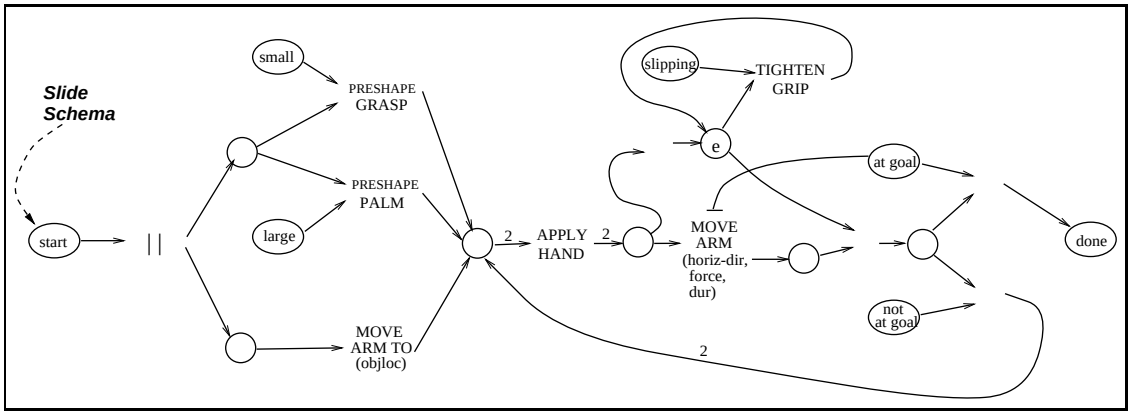


Figure 3.4: The SLIDE x-schema.

3.2.5 The Slide x-schema in detail

Each x-schema corresponds to a single Petri net, and in this section we will examine the SLIDE x-schema in detail. It is shown in Figure 3.4.

The SLIDE x-schema controls actions which move an object across the surface of a tabletop. It begins when a token is deposited in the "start" place at the left-hand side of the figure. Its first transition essentially copies this token into two output places in preparation for carrying out two concurrent synergies (the || symbol is a reminder of this function). The two concurrent synergies are preshaping of the hand and moving the hand to the object. The preshaping step is conditional on the size of the object, choosing a circular five-fingered grasp if the object is small but a flat palm if the object is large.

Only when both actions have completed—enforced by an arc with weight of 2—does the x-schema proceed to the next step, actually applying the hand to the object using the preshaped grip. The APPLY HAND transition outputs two tokens. One of them fires the MOVE ARM transition which engages the arm in the continuous horizontal motion which is the central action of the x-schema. The direction of motion parameter is externally specified. The force of motion may be externally specified, or may be computed from the desired acceleration and the estimated weight of the object. The duration of the movement may be externally specified, but otherwise a value is computed which is likely to land the object near its goal location. However, if the object is observed to reach its goal position before this duration, the movement will be aborted.

As an example of error recovery, note the piece of network above the MOVE ARM transition in the top center of the figure. At the same time that the MOVE ARM synergy is executing, the second token emitted by the APPLY HAND transition makes its way to the place labelled "e". This enables the TIGHTEN GRIP transition to fire should slipping of the grip be detected. This transition returns a token to "e" whenever it executes, so that the x-schema remains prepared to tighten the grip further if more slipping is detected. Importantly, this error-recovery mechanism operates without any other interaction with the rest of the x-schema; it is an "interrupt handler" in computer-systems parlance. Once MOVE ARM is finished, the error-recovery mechanism should be disabled. This is accomplished by the unlabelled transition following MOVE ARM, which consumes the enabling token.

When the arm motion is complete, the x-schema tests whether the object is now in its goal position, and if so terminates by depositing a token in the "done" place. But if the goal has not been reached, the x-schema loops back to re-apply the hand to the object and move it again, with the parameters of motion recomputed for the reduced distance to the goal position.

The above discussion is noncommittal about the source of some of the inputs used to compute synergy parameters. The next chapter will discuss how x-schemas interact with language and reasoning and use *feature structures* for this purpose. But for the remainder of this chapter the source of these inputs will remain implicit.

3.2.6 Other x-schemas

Besides the SLIDE x-schema described above, we have several others.

The LIFT x-schema (Figure 3.5) controls actions in which the vertical position of an object is increased or the object is simply held in the hand. It has a similar structure to SLIDE, but involves some different grasp types. In particular, cylindrical items are picked up using a wrap grasp, while large objects are lifted by supporting them from below using the hand as a platform. The continuous arm motion step calculates its parameters somewhat differently than in the SLIDE x-schema. First of all, the direction is upwards. More interestingly, the force parameter can be calculated to yield either a given upward acceleration of the object, or else zero acceleration. With zero acceleration the x-schema

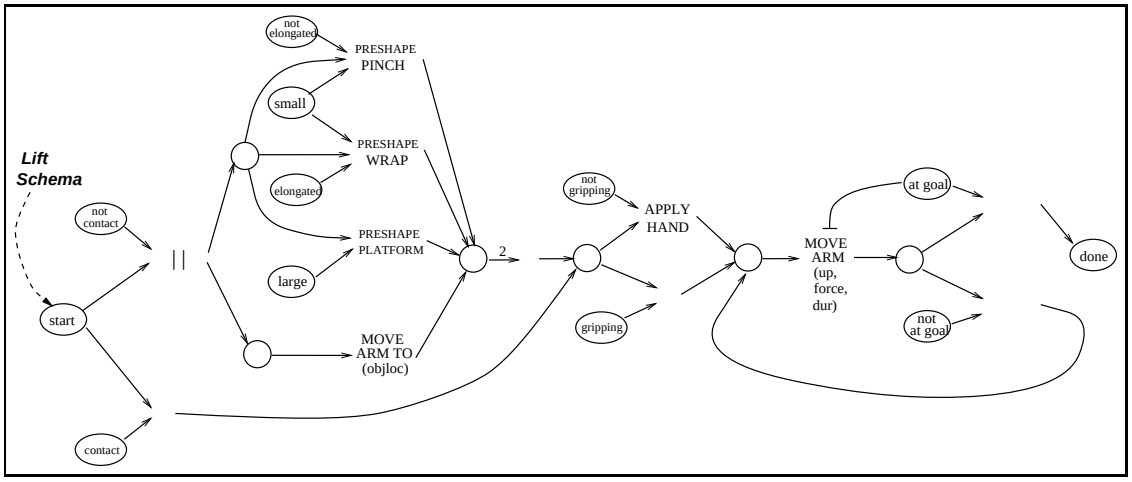


Figure 3.5: The LIFT x-schema.

implements holding an object still against the force of gravity.

The ROTATE x-schema (Figure 3.6) is used to rotate objects. Interesting features include the MOVE ARM TO step, which computes the "opposite" side of the target object with respect to the desired rotation direction, so that the hand will not end up underneath the object during the rotation. Also, the actual rotation step relies upon an abort connection into the main PIVOT WRIST synergy, which terminates the rotation once the object has re-achieved stability. This allows the rotation of, for example, a cube or a flat object. For a round object (or any object rotated around an axis allowing smooth rotation), the duration parameter would instead be used to determine the duration of the pivot synergy.

The DEPRESS x-schema (Figure 3.7) is for use with a button. Hence, it hardcodes the choice of shaping the hand with an extended index finger. The pressing action is implemented by a MOVE ARM synergy which relies on termination by detection of the increase of resistance occurring when the button has hit the end of its travel. The subsequent MOVE ARM step removes the hand from the button by a small distance, allowing potential repetition (modelling the activity of an anxious elevator passenger).

The TOUCH x-schema (Figure 3.8) encompasses a variety of behaviors and varies more in structure from the other x-schemas. At the beginning, the x-schema chooses one of two basic routes (top *vs.* bottom of the figure). The top route is chosen for slower touches. The hand may be shaped so as to use either the index finger or the palm. Once the hand

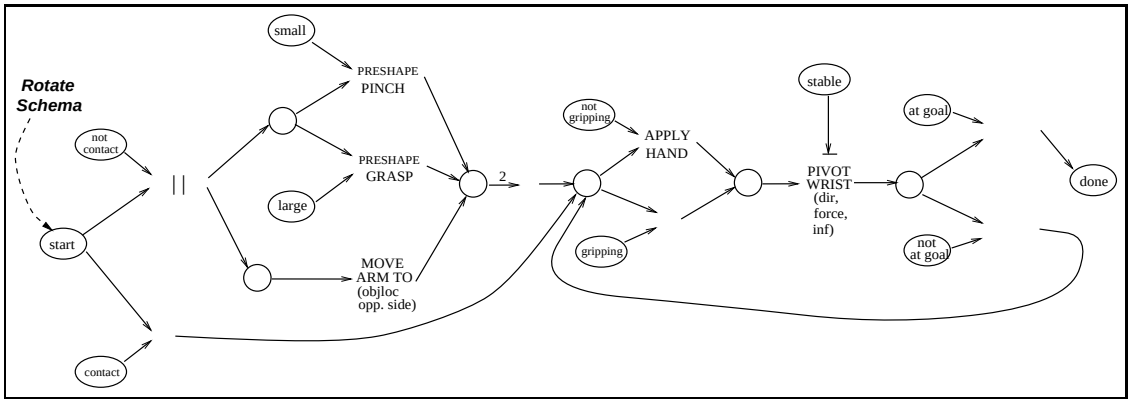


Figure 3.6: The ROTATE x-schema.

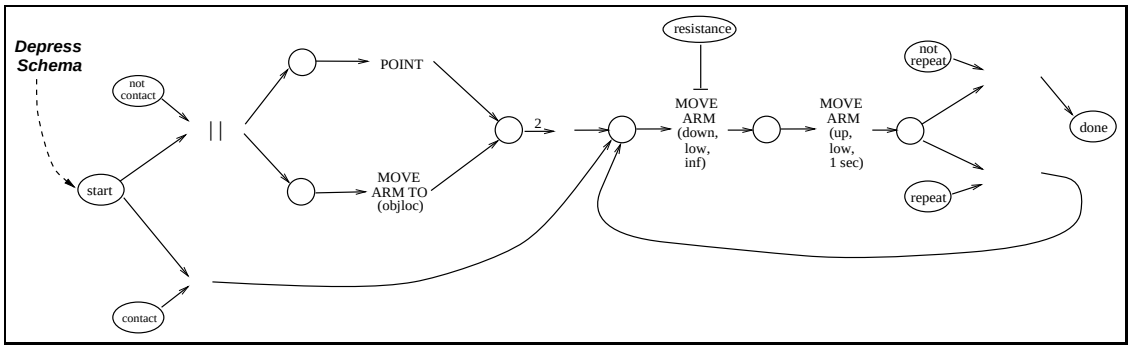


Figure 3.7: The DEPRESS x-schema.

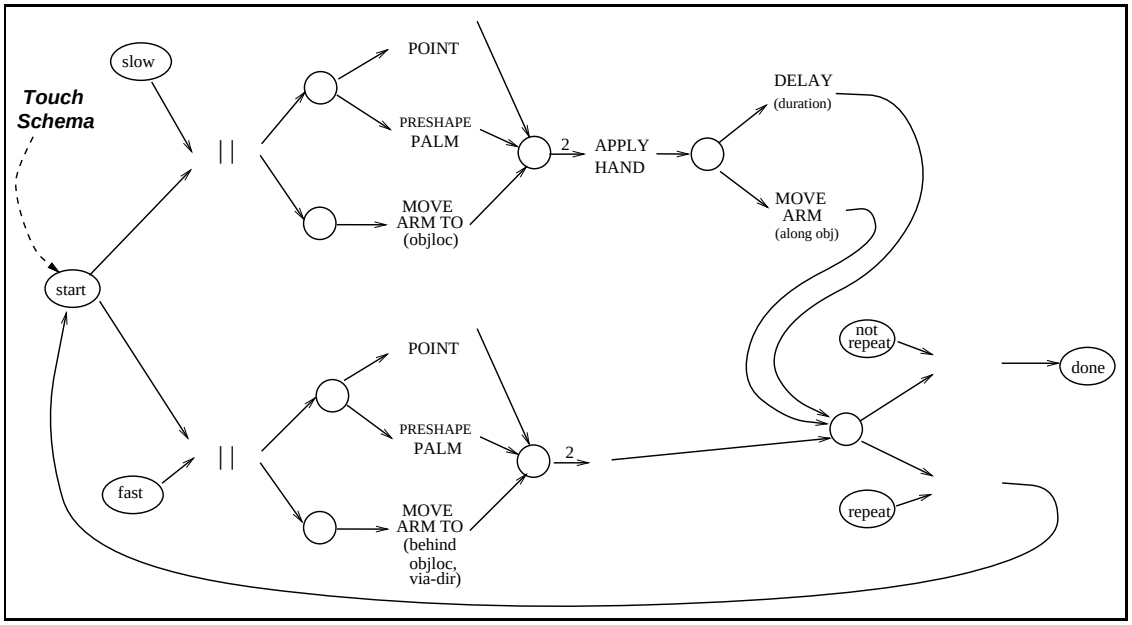


Figure 3.8: The TOUCH x-schema.

arrives at the object and contact is made via the APPLY HAND synergy, the x-schema may either simply delay for a variable duration (momentary *vs.* extended touching) or may move the hand along the surface of the object, crudely implementing a *feel* (the haptic exploration of objects involves a host of strategies (Lederman & Klatzky 1996) which aren't modelled here at all). The bottom route, in contrast, generates a rapid ballistic motion of the arm during its approach to the object, and aims at a location slightly behind the object to ensure that contact is made—resulting in hitting-like behaviors. The path by which the MOVE ARM TO synergy approaches the object can be influenced by specification of a "via point" through which the path is constrained to travel. This allows anything from straight-on hitting to a sideways slap. Whichever route is chosen, the entire action may be repeated.[7]

This set of x-schemas is capable of generating only a small fraction of possible hand actions. However, the intent here is not to build an exhausive set of controllers, but rather just to include a wide enough range of action controllers to make a convincing case for the approach to verb semantics presented in subsequent chapters.

[7] The "repeat" and "not repeat" places are under the control of either linguistic input (see next chapter) or higher-level planning systems not considered here, such as the "aspect graph" of Narayanan (1996).

3.2.7 Multiple entry points

A recurring pattern in the x-schema set is that they effectively have multiple entry points. This is a natural consequence of the goal-oriented nature of x-schemas and the fact that one can start toward a goal from varying distances (as measured in steps). For instance, LIFT will skip the grip-acquiring step if the hand is already gripping the object, in which case its behavior might be described in English as *raise* or *hold* but not *pick up*.

As shown in the LIFT x-schema, as well as the ROTATE and DEPRESS x-schemas, multiple entry points have been implemented using "noticer" transitions which "fast-forward" the control flow past unneeded steps when tokens are present in the appropriate perceptual places (e.g. the "contact" place).

Another strategy is to have true multiple entry points into x-schemas. In other words, we would declare a new x-schema RAISE, which would happen to point to the middle of the LIFT x-schema, yet would be treated as a separate x-schema by the rest of the verb-learning system. Such a strategy would facilitate learning the distinction between verbs like *pick up* and *raise*, while making it more difficult to learn verbs which encompass either starting point (as does *lift* for many speakers).

3.2.8 What *can't* be encoded in x-schemas?

What is the advantage of using a formalism such as Petri nets to encode x-schemas when, seemingly, one could just as well implement them as arbitrary computer programs? The difference is that, unlike an arbitrary procedure, a Petri net is *analyzable*. The types of interactions between primitive actions are limited. Consequently, important properties, such as repetition or interruption, can be "pointed to," i.e. described by their network patterns. This analyzability derives partly from the restrictions the formalism places on the procedures which can be encoded. For example, while a Petri net can encode concurrency, it cannot specify the details of two concurrent actions such as their relative rate or which finishes first. Such representational restrictions render the learning of action categories tractable, and constitute an interesting claim about what kind of details are semantically salient and which are semantically irrelevant.

3.3 Connectionist Account

As a plausibility argument, this section sketches an account of how standard connectionist techniques could be employed to model x-schemas. These ideas have not been implemented. First, we will present a simple approach which captures the control structure of Petri nets. Then we will overview a much more complex architecture which handles parameter bindings as well as control flow.

3.3.1 Petri net control

For the most part, the Petri net formalism is a natural for connectionist implementation. It is a graph structure with limited connectivity. Both transitions and places can be represented as connectionist units. We will first describe the representation and then give an example (which is illustrated in Figure 3.9).

Transitions are the simpler case. Conceptually they can be implemented with single neurons (although for robustness reasons one would ultimately expect a cluster of neurons). The incoming arcs of the transition correspond to the inputs to the neuron. The weights of the neuron are all set to 1, and the threshold is set to the number of inputs. The enablement of the transition corresponds to the simultaneous firing of all the inputs to the neuron. Note that this is the only condition under which the neuron will fire, due to its high threshold. When it does fire, it does so for only a brief period. The output arcs of the transition correspond to multiple branches of the output axon of the neuron.

Places are somewhat trickier since they must be capable of storing tokens indefinitely between being deposited and being consumed. This can be modelled using a group of neurons (minimally two) with mutual reinforcement. Such recurrent connections allow the group to settle into stable states of activity, even after external excitatory or inhibitory inputs have ceased. Using this design, the presence of a token corresponds to sustained activation of the group of neurons, and absence of a token corresponds to sustained inactivity. Depositing a token corresponds to incoming excitation that elevates the group to the active state. Removing a the token corresponds to inhibitory activation that returns the group to the inactive state. Outgoing axons from the group of neurons relay the current activity level to transitions so that they may determine whether to fire.

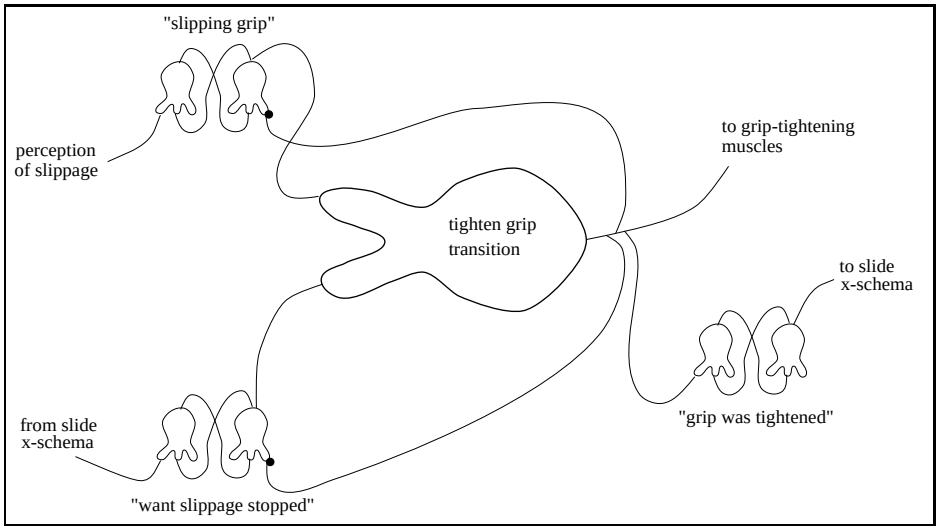


Figure 3.9: An example connectionist implementation of Petri nets including one transition and several places.

When the transition actually fires, a mechanism is required to remove its input tokens. This can be accomplished with backward connections from the transition unit to its place units. The backward connections supply sufficient inhibition to reset the place units to the inactive state. For transitions which fire instantaneously, the output axon of the transition neuron can be used to accomplish the removal of the input tokens. For the case of durative actions, the translation process described in §3.2.3 would be needed to allow this procedure to work. One problem with this design is that there is a potential for *race conditions*. If two transition neurons are competing for a token from a shared input place, it is possible that they will both fire before either gets a chance to remove the token. The probability of this occurrence is reduced if transitions have delay before firing, but is not eliminated. It should be noted that the x-schemas presented above do not use this type of nondeterministic competition for tokens.

An example of these ideas is shown in Figure 3.9. Neurons in this and later figures are drawn with a lobed side representing the dendritic tree, which is the "input" side of the neuron, while the "output" axon is drawn as a line emerging from the rounded side. In the center of Figure 3.9 is a neuron implementing a transition—in particular, the TIGHTEN GRIP transition from the SLIDE x-schema. It accepts input from two groups of neurons, one which implements the "slipping grip" place, and the other which implements the place

signifying that we currently care about stopping any slippage. The output of the transition neuron projects to a third group of neurons which represent a place signifying that the grip has been tightened. The output also projects back to the input places with inhibitory connections, indicated by round-tipped links. Lastly, the output projects to whatever motor machinery actually controls the grip force. Within each group of neurons implementing a place, the recurrent connections are shown. External excitatory connections to and from the three place units are also shown.

3.3.2 Passing parameters

The preceding section addressed the control portion of the x-schema model, but did not address the problem of passing parameters within x-schemas. This section considers some ideas for integrating parameter passing with control flow.

One simple design strategy would be to keep the motor parameters separate from the connectionist Petri net controller. They would connect only to the primitive synergies they parameterize. Indeed, there is some evidence for this design, such as the experiments of Georgopoulos (1993) which suggest that certain areas of motor cortex are dedicated to representing single parameters such as direction, independent of other aspects of the current action. With this design, when a transition fires it would trigger the corresponding motor synergy (whose neural implementation we have not considered) and that synergy would "read" any needed values from the global parameter-storing area.

But it's hard to envision how that solution could scale up to x-schemas in which some parameters might be used by more than one synergy during execution. For example, perhaps the arm must be moved with one force value, while the hand squeezes an object with a different force value. This situation requires *binding* of parameter values to particular transitions in the Petri net.

To solve this binding problem, a connectionist action controller has been developed (Shastri *et al.* 1997) allowing parameters to be (1) passed into an x-schema, (2) propagated with potential modification, and finally (3) used to modulate primitive synergies. The mechanism builds on SHRUTI (Shastri & Ajjanagadde 1993), a connectionist system for logical reasoning.[8] This system is fairly complex and will not be described in detail here.

[8] In fact, the controller mechanism can be integrated with the reasoning system, providing the necessary tools to build a full planning and execution system.

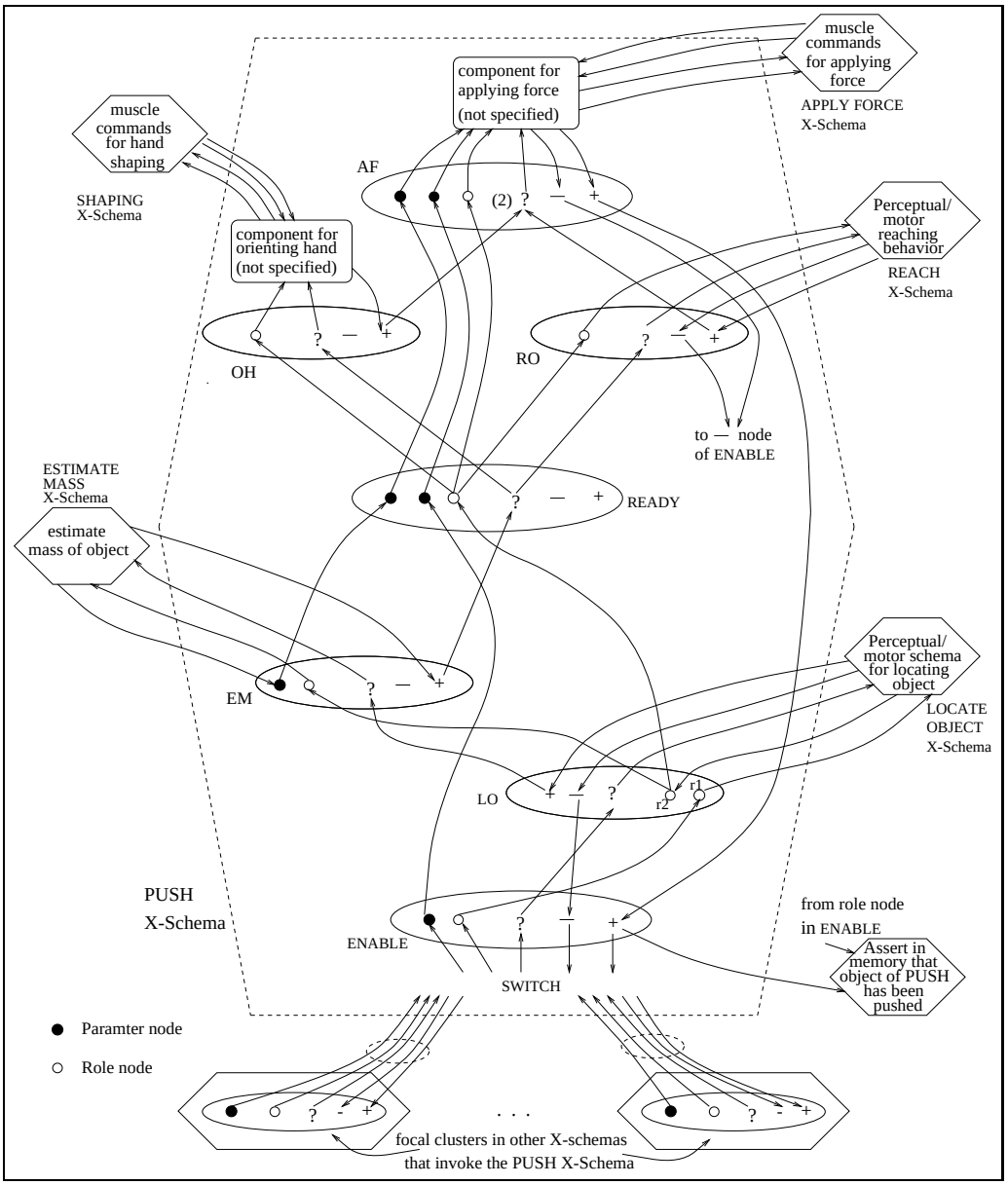


Figure 3.10: A SHRUTI implementation of a PUSH x-schema which is similar in function to the SLIDE x-schema of Figure 3.4. From Shastri *et al.* (1997).

The key idea, though, is that *temporal synchrony* is used to represent bindings. To use a familiar example, the connectionist representation of the predicate `loves(lover,loved)` would include units for the two roles of the predicate. Other units represent atoms such as `John` or `Mary`. To represent the assertion `loves(John,Mary)`, the `lover` unit fires at the same time as the `John` unit, and the `loved` unit fires at the same time as the `Mary` unit. The two synchronizations are kept distinct, and persist in a periodic manner while reasoning is performed. Other units for the `loves` predicate detect whether all bindings are in place and if so may trigger activation of related predicates such as `alive(John)`. The collection of units implementing a predicate are called a *focal cluster*.

The key step in adapting SHRUTI to implement x-schemas is that these focal clusters can be used to represent "stages" of execution of an action (although they do not map exactly to either transitions or places). Figure 3.10 shows a SHRUTI implementation of a PUSH x-schema. Focal clusters are drawn as ovals. A focal cluster is triggered when activation arrives at its "?" unit. Thus, the whole x-schema is invoked by sending activation to the "?" unit of its first focal cluster, shown at the bottom of the large hexagon (drawn with dotted line) in the figure. Along with this start signal, the focal cluster receives parameters by virtue of the firing of its *role units* (open circles in the figure) in synchrony with other units in the overall system which represent the parameter values. Each focal cluster may trigger a primitive synergy or perhaps a sub-schema, which eventually indicates completion by sending activation back to the "+" or "−" units of the focal cluster to indicate success or failure. The focal cluster then redirects this activation to the next stage of execution, perhaps to different stages depending on whether "+" or "−" is activated. The primitive synergy or sub-schema may also create new temporal bindings, thereby manufacturing new parameters for later steps of execution. (For example, in the figure the LOCATE OBJECT sub-schema creates a new binding representing the object to be pushed.) The final step of the x-schema returns activation to the "+" or "−" unit of the initial focal cluster to indicate completion (thus the first focal cluster acts as both the "start" and "done" place).

Although this connectionist representation does not directly mimic the places and transitions of Petri nets, the mechanism is able to encode sequentiality, concurrency and asynchrony, and can capture all of the control patterns used in x-schemas presented in this dissertation. It does suffer, however, from the same race conditions as the simpler design in the preceding section.

The reader is referred to Shastri *et al.* (1997) for a fuller account of this architecture.

3.4 Related Ideas in Artificial Intelligence

The notion of "schemas" has appeared in the AI literature in various guises.

One such notion, which came out of early work in planning, is the *procedural net* (Sacerdoti 1975). Procedural nets are a graphical, flow-chart-like representation of an activity, and to this extent they resemble our x-schemas. A further similarity is that procedural nets include "split" and "join" nodes, allowing expression of parallelism.[9] The main difference between procedural nets and our x-schemas is one of emphasis. While x-schemas are primarily action *controllers*, procedural nets serve primarily as a *working memory* for a nonlinear, hierarchical planner, NOAH. NOAH constucts a plan by beginning with a single node for the goal, and gradually refining portions of the net with more detailed sub-actions amongst which minimal ordering commitments are made. Thus the parallelism allowed by the split and join nodes is not intended for coordinating concurrent actions, but rather to express partial ordering of sub-actions. (Later addition of ordering constraints by various "critics" is a crucial part of the planning algorithm.) Since the net's main role is to represent these ordering constraints, it omits much of the reasoning regarding the actions it represents; a separate logical inference engine is used for this purpose. In contrast, our x-schema model aims to include reasoning within the Petri net formalism in order to achieve the rapid responses needed during action execution. (To be fair, the inferences performed by our x-schemas are much simpler than those required for full-fledged planning—but that is part of the point of using what can be thought of as "compiled plans".) Lastly, since Sacerdoti's work precedes the era of conditional plans and integration of planning and execution, procedural nets do not include a mechanism for branching. The ability to choose execution paths based on world state—and in particular to do so in an asynchronous manner—is the biggest advantage of the Petri net formalism.

In the robotic control world, our x-schemas are similar to the schemas of Arbib *et al.* (1987) in their state-machine-like character as well as their use of parallel mechanisms

[9] On a technical note, it appears that Sacerdoti's system only constructs nets with *nested* parallelism, though the representation is not inherently limited in this way (nor are Petri nets).

for control and for passing parameters. Our x-schema design is also akin to the subsumption architecture of Brooks (1986) in that multiple layers of control may execute concurrently (and also shares the finite-state nature of each controller). Nilsson (1994) offers a slightly different perspective on schemas, in which a production system continuously monitors primitive actions and switches amongst them appropriately as relevant (and discrete) changes in world state occur.

In the learning and sensorimotor representations world, x-schemas also bear a resemblance to the quasi-Piagetian schemas of Drescher (1991) in their use of essentially independent primitive actions connected only by common preconditions and postconditions, which allows for very flexible execution patterns depending on the vagaries of the world.

Schema-like representations have also found application in modelling verb semantics in previous work. As part of the *Jack* project (Badler *et al.* 1993) (see §8.1.1), certain coordinated activities such as grasping or walking are encoded using a construct called *parallel transition networks*. These networks bear a passing resemblance to our Petri net representation. However, practicality, not elegance, was the main design criterion, and as a result they would make a less appealing semantic base than Petri nets. In particular, each state has hooks for including arbitrary Lisp code at each state, and also there are several global mechanisms operating in concert (or cacophony?) with the local code in each state.

Connectionist implementation of x-schema-like representations was the focus of Goddard (1992), which represented various walking and running motions as cyclic sequences of configurations and used these representations to recognize the motions in visual input. The intricate strategies for managing delays would likely prove useful in a strictly connectionist implementation of x-schemas.

3.5 Thoughts on Hierarchical X-Schemas

The limitation of the above x-schemas to fairly simple actions raises the question of how x-schemas for more complex actions could be constructed. More complex actions are obviously necessary from the problem-solving point of view, but also potentially from the lexical-semantic point of view. For example, consider English verbs such as *stack*, *tie* or *sharpen*. In certain languages—serial verb languages in particular—verbs directly code for sequences of simpler actions, and thus clearly require an account of x-schema composition.

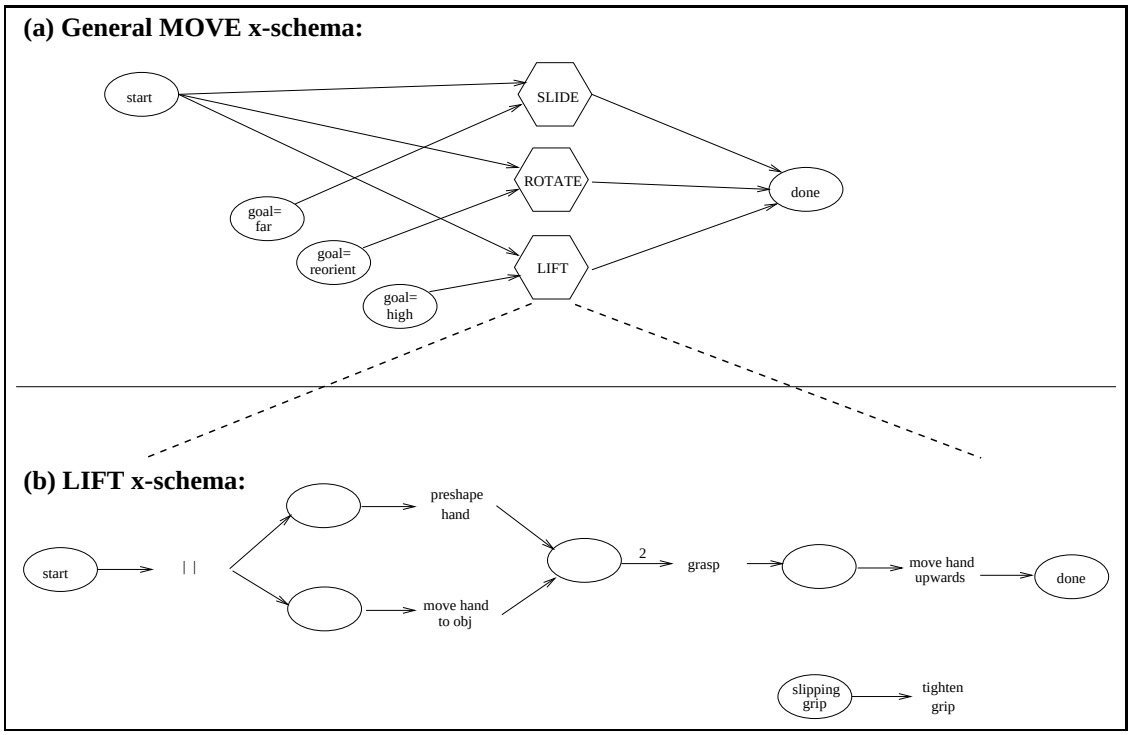


Figure 3.11: Two x-schemas organized hierarchically.

The Petri net formalism can be extended easily to allow for hierarchical networks. An example is shown in Figure 3.11. Hexagons represent transitions which invoke entire sub-schemas rather than primitive synergies. Note how the LIFT x-schema is called as one possible step in the more general MOVE x-schema depicted above it. An "umbrella" x-schema like MOVE, which serves only to choose a more specific x-schema which is best suited to the current goal, would facilitate learning general verbs like *move*. Without it, the system must learn the collection of x-schemas which are associated with *move* (SLIDE, ROTATE and LIFT in this illustration). This statistical determination would require a substantial number of training examples. But if the MOVE umbrella x-schema were present, the verb *move* could simply map to this x-schema—a mapping which could be learned relatively quickly. In effect, umbrella x-schemas could aid learning by pre-specifying collections of specific x-schemas which are likely to be linguistically relevant.

Here are three approaches to handling more complex actions in the context of verb learning. One approach is to manually construct higher-order x-schemas such as the MOVE example just shown. Unfortunately, it would become impractical to design and hardwire into the model such x-schemas as we scale up to ever more complex behaviors (as would be necessary, say, if we were to consider sentential as opposed to lexical meaning) because the range of possible behaviors is too large. A second approach would be to create a special set of "template x-schemas" which essentially provide a catalog of ways in which the lower-level x-schemas may be combined. We might have template x-schemas for executing two x-schemas sequentially, for executing one x-schema conditional on successful execution of another, for repeatedly executing an x-schema, and so forth. The major design problem any template-schema solution would have to solve is how to bind the appropriate sub-schemas to the template x-schema for a particular invocation. The third approach, developed by Narayanan (1996), is similar to the template x-schema idea but proposes a single template which captures the stages inherent in any process, such as enablement, interruption, completion, and so forth.

Chapter 4

Linking Actions and Verbs via Features

This chapter shows how the execution of the x-schemas presented in Chapter 3 interacts with the language portion of the model via a restricted interface, called the *linking feature structure*.

4.1 Cognitive and Linguistic Motivation

The case has been made that some details of motor control matter for verb semantics. Yet it's crucial to see that linguistic expression of motor control is restricted. After all, consider how hard it is to tell someone how to juggle, ride a bike or play hockey. We don't have conscious access to our stretch reflex thresholds or motor unit activation levels. We often don't even have a good idea of our joint positions (Soechting *et al.* 1996). We

certainly don't have conscious awareness of discrete action controllers and all their state changes during behavior! This motivates the search for a limited set of *features* to represent those aspects of x-schema execution which are relevant for distinguishing verbs. These features will be the sole means by which verbs can "connect" to x-schemas.

But which features to choose? One source of motivation is the linguistics literature. Talmy (1985) is the most comprehensive cataloging of features encoded by verbs and their *satellites* (which roughly correspond to the grammatical forms we consider in Chapter 7 and call "verb complexes"). A key result for *motion verbs*—a category that overlaps our domain—is that a given language will tend to encode in its root verbs, along with the fact of motion, only one of (a) manner/cause, (b) path, or (c) figure. Other semantic categories can appear in verb roots as well, and Talmy has compiled a list which, considering only those items relevant to our simplified domain, gives us the following sources of linguistic distinction:[1]

> cause, manner, purpose (goal), figure (for Atsugewi), path (in, out, up, down, past, through), polarity (occurrence *vs.* non-occurrence of key event), phase (start, stop, initiate, finish), aspect (one-way, full-cycle, iterative, continuous, gradient), direction (deictic)

A recurring pattern in action verbs also observed by Talmy is that verbs which express manner tend not to express goal. For example, releasing and obtaining possession are distinct goals and have distinct verbs (*put* and *get*) which don't express manner. If we wish to express that the manner is a twisting motion, we have only the verb *twist*; the goal must be expressed by a satellite, as in *twist in* and *twist out*.

The above catalog tells us what *kinds* of features are semantically salient, but does not tell us the level of detail required. It is likely that the appropriate level is fairly abstract. Talmy (1988) provides an account of how motoric features such as force may be conceptualized in a schematic way for purposes of language. Mandler (1992) presents psychological evidence that such schematizations are present before lexical development begins.

Jeannerod (1994), by way of some ingenious experiments with mental imagery of

[1] The catalog is also important for its list of features which do *not* prove relevant to verb semantics. These include: resulting event subordinate to main event, ground alone, hedging, degree of realization, rate (Talmy claims it appears only in conjunction with other properties), spatial location of speaker or hearer, tense, speech act (e.g. declarative *vs.* imperative).

actions, also demonstrates that only a subset of all the actual motor parameters are directly available to consciousness; the others are "re-created" using typical—though not perfectly reliable—relationships amongst parameters. For example, imagined duration of actions has been shown to be a complex function of amount of force, radius of curvature of motion, and effort expended.

In the next section we consider the projection of the above features onto our restricted domain of hand movements to help determine an appropriate set of linguistic features. In doing so, we must be able to extract the features from the schematic representations developed in the previous chapter.

4.2 The Linking Feature Structure

In order to map to declarative linguistic representations, the execution of an x-schema is summarized in what we will call a *linking feature-structure* (*f-struct* for short). The linking f-struct (and other kinds of f-structs, as we will see later) are drawn as horizontally extended double-boxes, as can be seen at the top of Figure 4.1, which shows a subset of the linking features we will use. Structurally, an f-struct is a list of (feature, value) pairs, and our use of the term "f-struct" is meant to facilitate compatibility with feature structures as conventionally used in linguistics.[2] Each (feature, value) pair occupies a column in the double-box, with the feature name in the top box and the value in the lower box. (In Figure 4.1 several possible values are shown for each feature.)

Crucially, the linking f-struct by virtue of its position in the overall architecture of the model provides a bidirectional interface between x-schemas and language. The features in the linking f-struct are connected to appropriate locations within the x-schema set (as indicated by dashed lines in Figure 4.1 and discussed later) so that when an x-schema executes, appropriate values are placed in the linking f-struct. Similarly, when linguistic input sets particular values in the linking f-struct, these values will then appropriately guide the execution of x-schemas.

The set of features is hardwired into the model, meaning it cannot change during verb learning. Certain features may prove irrelevant for a given language, but no new features can be added. Thus, the set of features represents a strong claim about what kinds

[2]The actual features will, however, differ from those conventionally used in linguistics.

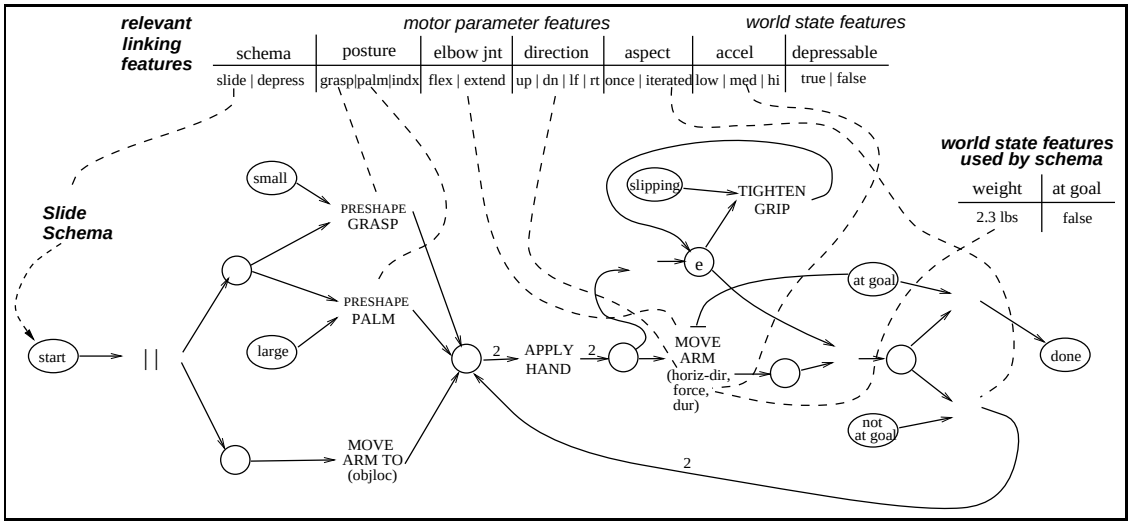


Figure 4.1: A linking feature structure (top) and its connection to the SLIDE x-schema.

of properties of the execution of x-schemas can "matter" for the verbs we consider. In return for this commitment, the small size of the linking f-struct (compared to the full set of possible properties of executing schemas) facilitates rapid learning.

In summary, attempting to bridge the linguistic-motor gap with a simple static feature structure yields some significant simplifications. This separation not only allows natural representations to be used on each side of the divide, but more importantly, it limits the hypothesis space while providing necessary access to active motor control machinery.

4.2.1 The linking feature set

The following list summarizes the set of linking features used in the current model. Remember, the names of these features and their values serve only as identifiers to aid us in discussing them; they have no theoretical significance and derive their meaning solely from their grounding in x-schemas, which will be discussed next. The linking features dealing with motor control are:

- `schema (slide, lift, rotate, depress, touch)`:
  Perhaps the most important linking feature. Specifies the x-schema generating the action. Since x-schemas are generally goal-oriented, this feature also implicitly specifies

the goal.

- `posture (grasp, wrap, pinch, palm, platform, index)`:
  Refers to the configuration of the hand while manipulating object. Usually determined by which hand-preshaping primitive synergy is used during x-schema execution.

- `elbow (flex, extend, fixed)`:
  Qualitatively describes the elbow joint's direction of motion (or lack thereof) during arm motion.

- `force (low, med, high)`:
  Specifies the magnitude of effort applied to the muscles of the arm, discretized into 3 levels. The force may be measured during arm motion (most x-schemas) or during static application of pressure (e.g. the TOUCH x-schema). This feature could easily be generalized to include finger force to capture verbs like *squeeze*.

- `acceleration (zero, low, med, high)`:
  Specifies the acceleration of an arm movement, discretized into three levels plus a zero value.

- `direction (away, toward, up, down, left, right)`:
  Gives the direction of motion, with respect to the body. Also used for rotation, in which case it refers to the direction in which the top of the object moves. Discretized into a small set of salient directions.

- `aspect (once, iterated)`:
  Reflects whether the pattern of state transitions during execution involves repetition of a loop. Loop detection is only a simple example of the kinds of aspectual distinctions which can be given precise specification within the Petri net formalism. The representational ideas have been worked out (Narayanan 1996), and learning such semantics for both inherent aspect and aspectual markers fits the framework of this thesis. However, implementation remains for future work.

- `duration (short, med, long)`:
  Specifies the length of time to carry out a particular salient continuous synergy within the x-schema, discretized into 3 levels.

The linking features dealing with perceived world state are:

- `depressible (true, false)`:
  True only if the object is button-like.
- `elongated (true, false)`:
  True for long, thin objects but not cube-like objects.
- `size (large, small)`:
  A coarse measure of object size.
- `contact (true, false)`:
  True if the hand contacts the object prior to x-schema execution.

4.2.2 Connecting to x-schemas

As mentioned earlier, these features are abstract in that they are wired into multiple x-schemas, and the roles played in different x-schemas can differ. The features also must act as both input to and output from executing schemas.

We can return to Figure 4.1 for a depiction of how several of the linking features connect to the SLIDE x-schema. The `schema` feature is associated with the presence of a token in the start place of the x-schema. The `posture` feature is set according to which preshaping synergy executes—PALM or GRASP. The `direction` feature is associated with the direction parameter of MOVE ARM, while the `acceleration` feature is related to the force of the move via a calculation involving the object's weight. The `aspect` feature, which distinguishes iterated from non-iterated actions, is set according to whether the lower-rightmost transition fires since this is the mechanism by which the x-schema repeats the grip-and-move sequence.

The connections between the linking features and the other x-schemas differ in sometimes subtle ways. Figure 4.2 describes how the above linking feature set is hooked to the full x-schema set presented in the previous chapter. In particular, note that while features like `acceleration` and `direction` are linked to the *object motion* phase for most x-schemas, they are linked to the *hand's approach* to the object for the TOUCH x-schema.

	X-schema:				
Feature:	**SLIDE**	**LIFT**	**ROTATE**	**DEPRESS**	**TOUCH**
`schema`	identifies this x-schema	identifies this x-schema	identifies this x-schema	identifies this x-schema	identifies this x-schema
`posture`	while moving object	while moving object	while rotating object	while pressing	during contact
`elbow`	while moving object	while moving object	N/A	N/A	during approach
`force`	while moving object	while moving object	while rotating object	while pressing	during contact
`accel`	while moving object	while moving object	while rotating object	during approach	during approach
`direction`	motion of object	motion of object	rotation of wrist	N/A	of approach
`aspect`	grasp & move object	move object	grasp & rotate object	press & release	approach & contact object
`duration`	of moving object	of moving object	of rotating wrist	hold-down phase	contact phase

Figure 4.2: The different roles played by the linking features in different x-schemas.

4.2.3 Deriving the feature set

First and foremost, the feature set stands or falls based on its ability to support successful training on words from a variety of languages, and during its development it has often fallen. The current feature set is the product of an iterative experimental process. While this process differs from the analytical style common in linguistics, I feel that it has equal legitimacy, even though our experimental capabilities prevent us from exploring the full richness of human language use. The process has previously proved effective in work on spatial semantics (Regier 1996).

That said, several influences have guided the search for the best feature set. First is the cognitive and linguistic data as reviewed earlier in §4.1.

Another is the work of Agre & Chapman (1987) on so-called "indexical representations". The essential idea is that rather than manage a large number of bindings, there are special slots for the features of the object of interest in the current context, i.e. the action currently being carried out. This served as motivation, for instance, for using a single abstract `force` feature which represents whatever force value is relevant in the current situation, by virtue of its connections to the x-schema set. Recently, Ballard *et al.* (1996) have argued that such representation schemes are used by humans and have their genesis in the fact that bodily motions occur on approximately the same time scales as some reasoning tasks. For example, re-orientations of the hand or eye may be used to avoid having to store in memory certain bindings which can instead simply be perceived. The indexical representation view is also compatible with the neurobiological work cited earlier (Georgopoulos 1993) on population coding of motor parameters, since representations which are spread across large numbers of neurons are not easily copied.

And finally, it must be confessed that the feature set is influenced by what is easily computed from the x-schema set presented in Chapter 3. As a result, it is inevitable that important features are missing whose lack would become evident with a more detailed and biologically accurate x-schema set. Moreover, additional features would doubtless be needed to handle a broader range of actions, especially more complex actions. In the end, I believe the feature set is interesting for its flavor and for the example it sets, and not as a claim about a complete and "true" set of features used by language.

4.2.4 How many features?

In addition to *which* features to use, there is the question of *how many*. Recall that it will be important to have a "moderate" number of features. An overly small set would not be rich enough to represent all the conceptual distinctions found in languages. But an excessively large set (for example, a full trace of all token movements and synergy parameters in the x-schema set) would render learning intractable due to the "relevance problem"—the more features associated with each example of an action, the greater the burden upon the learning algorithm to determine which subset of features are relevant for each category to be learned.

But how to arrive at this mysterious number, "moderate"? The obvious possibilities are (1) guess a large number, then cut unneeded ones until learning becomes fast enough; and (2) start with the empty set, then add features only as necessary until the learning algorithm can draw the necessary distinctions. In practice, I have generally worked with a set of 5 to 15 features at a given time, using fewer features when investigating the learning algorithm's properties but more when attempting to fully model the verbs under study.

A subtle point on this issue of feature set size is that a minimal feature set may not lead to a minimal lexicon. It is likely that some verbs have multiple possible descriptions in terms of the features under consideration, and that some of these descriptions will be shorter than others. For example, perhaps *heave* can be distinguished from *lift* by some combination of hand `posture`, object `size`, and `acceleration`, but the distinction may have a simpler description in terms of just the `force` involved. In this case, omitting `force` from the linking feature set would be possible, but including it would lead to a more compact representation for *heave* and *lift*. So, there is a benefit to having a slightly larger than minimal "menu" of features for the learning algorithm to choose from. The current feature set does indeed include correlated features, such as `force` and `acceleration`, and also `elbow` and `direction`.

On a related topic, it has been suggested that this attempt to find a small number of universal linking features is reminiscent of work on "deep case" in linguistics (Fillmore 1968). The goal of deep case was to delineate a small, fixed set of abstract frame-semantic roles (agent, patient, instrument, experiencer, etc.) which could serve as generalizations of the specific roles found in the various frames in our conceptual system. Then, grammatical

constructions would need to tie syntactic categories such as subject or object only to these deep case roles, rather than to the individual roles in every frame. Moreover, the existence of deep case would be a powerful clue about the structure of our conceptual system. The linking feature structure described in this chapter shares the goal of seeking a small number of universal abstractions. But two important differences exist. First, the work on deep case focuses on the structure of frames and hence is an attempt to arrive at the fundamental structure of our conceptual system. In contrast, I make no claim that action verb semantic features will enjoy such generality. Second, the work on case is, in a sense, "object-oriented" (the fillers of roles tend to be physical entities), while many of the action verb linking features are instead "parameter-oriented" (e.g. `force`, `duration`).

4.2.5 Why a separate linking structure?

The reader may wonder why the linking f-structure deserves to be reified as an actual structure in the model, as opposed to simply using a set of connections from the x-schema set into the representations of individual lexical items. The most important reason for this design choice is that features don't come directly from x-schemas, but rather are *abstractions*.

As we saw in §4.2.2, linking features like `direction` can be bound to the direction of hand motion in one x-schema, or the direction of object motion in another (see Figure 4.2). The abstraction is accomplished by virtue of multiple connections to and from the x-schema set. Importantly, this abstraction should not be performed within the representation for each verb which encodes direction, since this needlessly increases the required neural connectivity. If we have n verbs using an abstraction, and m different "groundings" of the abstraction in the x-schema set, then we would need $n \times m$ connections to implement the model without a linking f-struct. With the linking f-struct, the number of connections is only $n + m$.

A secondary reason for the use of separate structures for holding features is that it will prove useful in scaling up the model, where the need to deal with more than one f-struct at a time will arise.

4.2.6 Static *vs.* dynamic

It is important to realize that, as it is used in the current model, the linking f-struct is a *static* structure. In other words, it summarizes the execution of an x-schema, rather than changing dynamically during execution. This clearly restricts what can be represented about an x-schema execution, without resorting to features such as force-at-time-1, force-at-time-2, etc. But with this restriction comes a tremendous simplification of the verb learning problem: classification of sequences is generally regarded as more difficult than classification of static feature vectors, especially when the sequences vary in length.

And it is not unreasonable to assume that children compute a summary of a temporally extended action. After all, the child normally hears the verb label in advance of the action (Tomasello 1992), so he is alerted to begin storing a representation. Also, early verbs tend to correspond to short-duration actions. From an empirical standpoint, Regier's model (Regier 1996:Section 5.4) successfully learned motion senses of spatial terms using static summaries of feature values. And that was accomplished by averaging over movie frames, with no notion of the key times during the movie. We have an easier time of it, since Petri nets already encode only the key events in the action.

Limitation

Yet it must be admitted that for full modelling of language, a static structure won't suffice. Especially for multiple-sentence discourse understanding, it must be possible to make reference to feature values bound to multiple points in time. A hybrid solution would be to create separate linking f-structs for different points in time.

4.3 Connectionist Account

The connectionist representation of features is not particularly difficult. It turns out that the best strategy (and a biologically plausible one as well) is to use *place coding* (Feldman & Ballard 1982). That is, for each feature, there is a dedicated connectionist unit (i.e. a separate place) for each possible value of the feature. In Figure 4.3, the rectangular box shows an example of place coding to represent the possible values of the `force` feature. For continuous-valued features (which we don't use in our implementation for simplicity, but

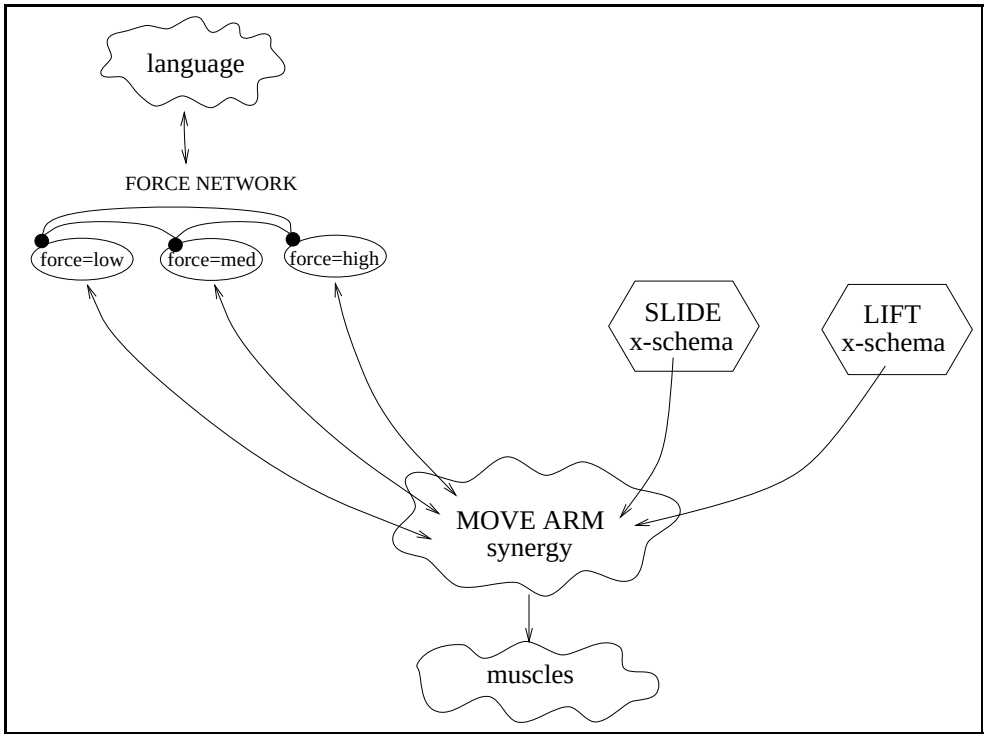


Figure 4.3: The connectionist representation of a linking feature such as `force`, and its connection to motor control.

---

which certainly exist), place coding forces us to discretize the range of the feature; however, by supplying varying degrees of activation to these discrete values we can still represent the continuous range. One fundamental advantage of place coding is that it avoids encoding feature values as activation levels, which is prone to noise sensitivity and is also unnatural for nominal-valued (rather than numerical) features. Another advantage of a place-coded design is that feature values are never passed around. Since they are tied to particular units, feature values can be straightforwardly linked to their grounding by simple connections to the relevant perceptual or motor apparatus, as we will show shortly.

We now turn to connectivity. Within the network representing the possible values of a feature, we desire a winner-take-all (WTA) behavior. That is, only one value unit should be active at a time, once the network has settled. And the active value should be the one which is receiving the strongest evidential support from outside the WTA network. The winner-take-all behavior is achieved by inhibitory connections between each pair of

value units in the network. These are shown by circular-tipped connections in Figure 4.3. Further details will be provided in §5.4 once we've laid out the full requirements which these feature networks must fulfill.

Here, instead, our focus is the connection from the linking feature value units to the model of motor control, i.e. the x-schemas and primitive motor synergies. While there's nothing particularly clever required to do this, we should at least mention the several types of connections which are used.

Some linking features are used to parameterize a variety of primitive synergies. For example, our abstract `force` feature parameterizes synergies such as MOVE ARM and TIGHTEN GRIP. Thus, each value unit for this type of feature is connected to all of the synergies which it can parameterize. This is the situation depicted in Figure 4.3 (although only one synergy—MOVE ARM—is shown). In the event that the currently executing x-schema should trigger the MOVE ARM synergy, the neural circuitry implementing that synergy will detect the currently active force value unit and use it to control the muscles. Other synergies which do not get triggered by the currently executing x-schema (say, TIGHTEN GRIP) effectively ignore any activation from incoming feature value units.

Other features, such as `posture`, serve to choose one primitive synergy versus another. For these features, each value unit is connected to all those locations in the x-schema set which should activate that value (in this case, it's the transitions for executing the corresponding hand posture synergy). The function computed by each value unit is a temporal OR: if any of these these connections is active at any time during the action being labelled, then the linking feature value unit will remain active at the end of execution.

Another interesting case is the `aspect` feature. Its `iterated` value unit is connected to "backwards"-pointing arcs which construct loops in some of our x-schemas. Recall that in any given execution, the loop may or may not be repeated. The `iterated` value unit computes a temporal OR (as described above) of the activity on this arc over the course of x-schema execution. Its peer, the `once` value unit, is wired to fire by default, unless inhibited by the `iterated` value unit.

Chapter 5

Word Senses and their Use

"Nothing has really happened until it's been described."
—Virginia Woolf

In this chapter we develop structures for combining linking features into word meanings and examine the algorithms which employ these word meanings to label actions and to understand and obey verbal commands.

5.1 Polysemy—Why Do Languages Have It?

The model's verb representation (and learning algorithm) will deal centrally with *polysemy*. Polysemy refers to the tendency of lexical items to possess multiple meanings, or *senses*, which are related to one another.[1] Why should words have more than one meaning? The non-linguist may consider this a strange phenomenon and an inefficient way for language to operate. One may even doubt the correctness of this account of meaning. Yet, the linguistic evidence is fairly clear (Lakoff 1987), and moreover is compatible with some convincing evidence from psychology (Rosch 1977).

The underlying issue is the structure of human concepts, and there are two relevant views. Under the traditional set-theoretic view, as typified by mathematical logic, a concept is a simply set of entities, which can be therefore be described by giving the necessary and sufficient conditions on membership. This tradition is prevalent in the field of semantics, which tends to favor concise and abstract meanings in the form of necessary and sufficient conditions. Such efforts are commonly oriented toward finding the minimal discriminant between a word and its contrast set, or determining which syntactic constructions a word can appear in. As it turns out, even for those tasks it is far from clear that this set-theoretical approach will work, due to its brittleness, context insensitivity, and lack of internal category structure.

The contrasting view of conceptual structure takes into account the considerable evidence that some examples of a category are "better" that others. Such examples are called prototypes. This evidence suggests a representation of categories based on these prototypes, rather than on necessary and sufficient conditions, thereby capturing the internal category structure. Often prototypes will be related to each other in stereotyped ways, such as metaphoric or image-schematic transformations. Thus, the full category representation involves the prototypes as well as the typed links connecting them. (See further discussion in §5.5, particularly §5.5.2.)

The arguments on this issue are rather involved, but within the context of our hand action verb learning task the difference can be made rather clear. The key point is that necessary and sufficient conditions, since they must be true of all examples of a category,

[1] When the meanings have no relation, such as the oft-cited example of *river bank vs. financial bank*, the phenomenon is called homonymy, and is not of interest here.

are necessarily rather abstract. Prototypes, on the other hand, are free to include specific details, since generalization is achieved by other means (e.g. by measuring similarities to prototypes). And for our verb learning task, in which we must interface our verb representation to actual grounded activity, we need a more prototype-like representation which includes the "full picture"—i.e. a conjunction of many features—especially for carrying out commands. To actually drive behavior given a command, we must know all the relevant motor parameters, not just those which distinguish the command from other verbs.

And why does this lead to polysemy, i.e. multiple word senses? Some degree of generalization can be achieved *within* a prototype by using graded response and/or removing features with multiple allowed values. But this kind of generalization must be sharply limited because it is at odds with the very specificity which allows prototypes to successfully drive motor behavior. Thus, to fully achieve the needed range of generality exhibited by verbs, one needs to employ several such senses—i.e. the verb's meaning is (roughly) the disjunction of the senses. The richness of each sense and its inclusion of relevant world state leads to strong context sensitivity which proves useful in determining the appropriate sense of a verb when this is needed.

This is not to say that necessary-and-suffficient-conditions descriptions of the verbs modelled here would be impossible to derive. Moreover, such descriptions—even if only approximately correct—may well prove important for abstract reasoning. While this dissertation does not include an account of such abstract concepts, nor of how they might be learned from the collection of prototypes we do model, such an enterprise would be worth pursuing. The important point for the current work, though, is that such abstract descriptions *alone* would not suffice for our task. Each language lexicalizes some x-schema parameters and not others. Language-independent pragmatic rules are useful for filling in unspecified parameters, but cannot predict the linguistically coded parameters. In other words, the specific prototypes which are needed for obeying commands would not derivable from the abstract descriptions plus pragmatic rules, because these prototypes depend on the vagaries of the particular language being learned.

5.2 Structure of a Word Sense

Our model represents verbs by a set of *word sense f-structs*. Like any f-struct, a word sense f-struct is a list of features; however, in this case the feature values are probability distributions. The implemented system uses multinomial (i.e. discrete) distributions (even for quantitative features like `force`). Each word sense f-struct also contains a measure of its frequency of occurrence during learning. Word sense f-structs are drawn the same way as linking f-structs are drawn, except the lower box shows probabilities on the values. For space reasons we often show only the *mode* value (the most probable value) and its probability. A verb's collection of senses is drawn in an oval. See the top of Figure 5.2 for examples of two verbs totalling three senses.

A word sense f-struct can be thought of as a conjunction of features, but more precisely it is a probability distribution over possible linking f-struct settings that is restricted in form by an assumption that the features are *independent*. The probability assigned by a word sense f-struct to a given linking f-struct can always be increased by choosing a higher-probability value for any individual feature (assuming such a choice is available). Consequently, the highest-probability linking f-struct is the one which contains the mode value for each feature. The probability values provide a form of graded membership which facilitates choosing the best-matching verb when none matches precisely. And the highest-probability linking f-struct acts as the prototypical example of the word sense—one can think of the word sense as a cluster around this point in "linking f-struct space". This ability to construct a prototypical example from a set of probability distributions is the key to the model's ability to perform "reverse" mappings from verbs to linking features (and ultimately actions).

Due to the independence assumption, a single word sense f-struct is incapable of representing concepts such as "SLIDE with `direction = away` or else DEPRESS with `direction = down`" in which correlations exist between features. In these cases the concept can be represented only by multiple word sense f-structs (cf. *push* in Figure 5.2). Critically, since language never identifies which sense of a word is being used, the choice of sense is a hidden variable which all algorithms must deal with.

5.2.1 On the use of probabilities

At this point a few comments are in order on the use of probabilities in our model of semantics. I would like to dispel any misconceptions that by using probabilities this work is offering a so-called objectivist account of semantics, in which categories are presumed to be properties of the world and not of the mind. I would also like to argue more generally that numerical measures are essential in semantics.

First of all, let us review why there are numbers in the model in the first place. The key point is that we want all the language processes we model (labelling, obeying, and learning) to be *evidential* in nature. That is, they must respond to context and to prior experience in a graded manner; they must "weigh the evidence". This evidence comes in the form of associations, whose strengths are represented numerically. These strengths are partially determined by frequencies of events, and to this extent they may seem "objectivist". But it is important to understand that the "events" in our model are described solely in terms of linking features which, through their connection to x-schemas and perception, represent a bodily-grounded construal of the world. Moreover, the association strengths in our model are also partially determined by various internal biases and expectations, removing them even further from the objectivity of simple frequencies.

In any neural implementation of the model, these numbers would be represented by activation levels and weights, and this will be considered in later sections. But neural or otherwise, the heart of any evidential system is the update rules it provides specifying how the numbers get used and how they change during learning. In general, it is hard to discover good update rules, or even to understand the properties which a given set of update rules will exhibit. One way to think about these numbers is to treat them as probabilities, as we do here. What this means is that we restrict the update rules we will consider to those which obey the laws of probability. In return, we get certain guarantees. For example, we are guaranteed that the numbers will remain in a bounded range (normally taken to be [0, 1]). We also get access to a body of probabilistic literature which includes update rules with well-understood behavior.

It is not claimed that any particular neural systems obey the laws of probability.[2] Rather, probability theory is simply something to leverage in the design of provisional

[2]Nor is it clear whether human reasoning in general obeys the laws of probability. Tversky & Kahneman (1974) argue that it does not.

models—models which will ultimately be evaluated by their fit to the data, not by their philosophical commitments. Indeed, an important question is what kinds of approximations are necessary to map probabilistically formal algorithms to known connectionist architectures.

Aside from the objectivist *vs.* subjectivist issue, there is a more general question about the status of any numerical measure such as probability in semantics. When we say that, e.g., *push* is associated with `palm posture` with probability 75%, what does this mean? Is all of this information part of the "semantics" of *push*, or is only the feature value `palm` part of the "semantics," while the 75% is just auxiliary information for use in "pragmatics"? My own view, similar to that of Wu (1992), is that the situation becomes clearer if we adopt the perspective of language *use* instead of language *description*. From this view, language is a situated activity in which decisions must be made under real-time constraints. Situated language models may well not divide naturally into "semantic" and "pragmatic" components.

Certainly, there is an intuitive sense in which it is unsatisfying to say that *push* means "use the `palm posture`, probably". We know that there are certain circumstances where the `palm posture` is called for and other circumstances where it is not, and an ideal representation should encode those conditions. However, the language learning child may not always be privy to the appropriate conditions and thus may be forced to simply record statistics until he better understands his environment. And full understanding may be a long time coming. In general, the tremendous difficulty of verifying or even representing all the preconditions of real-world rules is known as the "qualification problem" (McCarthy 1977). Other approaches to this problem include default logic and nonmonotonic reasoning, but these mechanisms enjoy neither the formal simplicity nor the evidence-weighing ability of probabilistic inference.

A convincing case study of the power of probability theory in language modelling is Jurafsky's (1996) model of full-fledged parsing. Short-term memory allows only a limited number of parses to be considered in parallel. A probabilistic account can be given specifying when constructions will be accessed, and when potential partial parses will be discarded. Such an account can explain certain psychologically observable effects such as the lengths of "garden paths" in sentences which use infrequent constructions.

PUSH: 2 senses

sense 1

schema		posture		direction	
slide	100%	palm	60%	away	50%
touch	0%	grasp	10%	toward	5%
		index	30%	up	15%
				down	30%

commonness 0.2

sense 2

schema		posture		force	
slide	0%	palm	85%	low	10%
touch	100%	grasp	5%	med	30%
		index	10%	high	60%

commonness 0.1

Figure 5.1: Two senses of push with full specification of their probability distributions.

5.2.2 An illustration of two senses of *push*

This section works through a simple illustration of word sense f-structs, to help ensure that readers understand the representation before we move on to how it supports labelling and obeying and how it is learned. Figure 5.1 shows two senses of *push* as word sense f-structs (these are copied from the Overview chapter).[3]

First let's look at sense 1. We see that three linking features are involved: `schema`, `posture` and `direction`. To get a feel for this word sense f-struct, it is helpful to determine its prototypical example. By choosing the highest-probability value for each feature, we see that the prototypical example is an execution of the SLIDE x-schema using a flat `palm posture` to move an object `away` from the body. If the model were commanded to push an object and this sense were chosen, this is the action that would result. The probability of this prototypical example can be computed from the distributions: $100\% \times 60\% \times 50\% = 30\%$. Changing any of the feature values will produce less-probable examples. Changing the `schema` to TOUCH would yield zero probability. Changing `posture` or `direction` would lower the probability in a more gradual manner. Observe that while sense 1 *prefers* `palm posture` over the other possible `postures`, it doesn't insist. `Index` finger posture is quite acceptable. If we were designing word sense f-structs for Spanish, we would not have a word sense like sense 1, because in Spanish, separate verbs are used for sliding using the `palm` *vs.* the `index` finger (*presionar vs. pulsar*).

Now let's turn to sense 2. By examining the feature distributions, we can observe that it represents the "apply pressure" sense of push, rather than the "move object" sense just described. Most importantly, sense 2 codes for a different x-schema: TOUCH. And it

[3] Note that these are not quite the same senses portrayed in Figure 5.2.

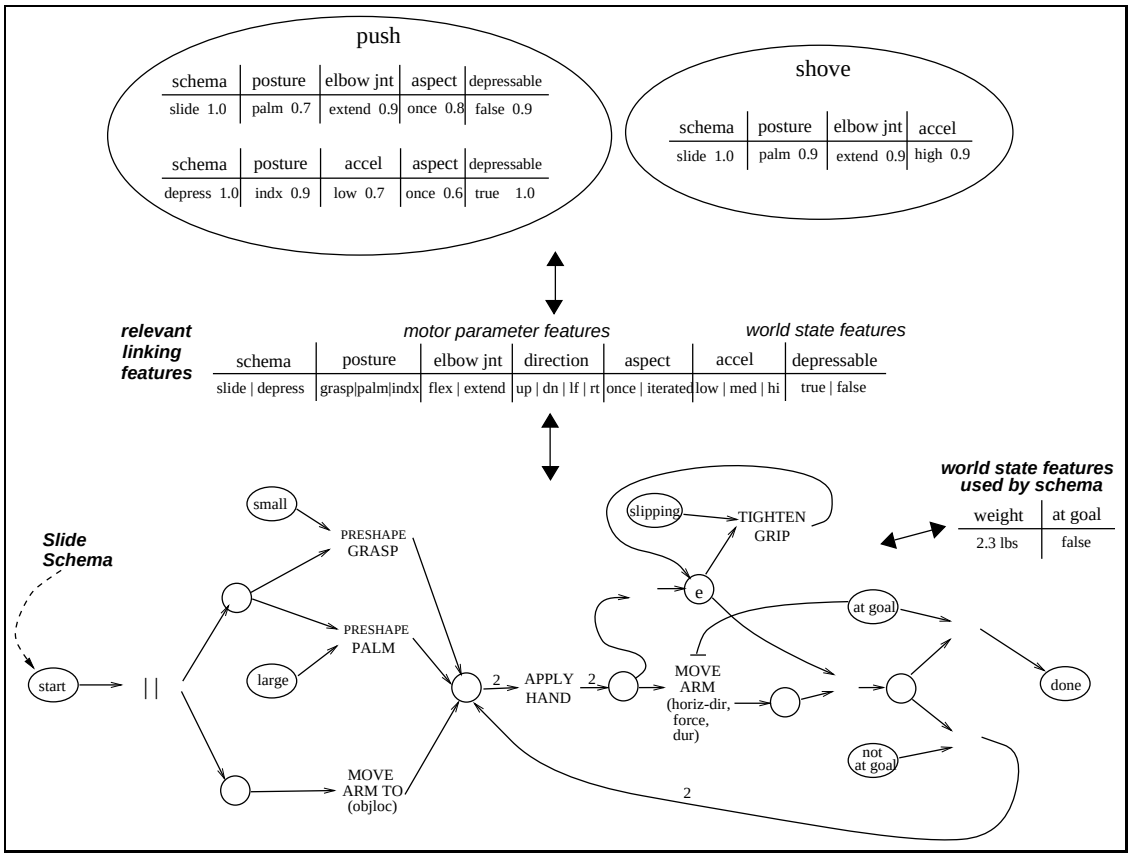


Figure 5.2: The full model as originally depicted in Figure 1.1 but filled in with the SLIDE x-schema, several linking features and two verb representations.

does so absolutely, just like sense 1. Typically in our model, different x-schemas will lead to separate senses. (Moreover, this is usually a good indication of a conceptual distinction that will be marked by separate verbs in some other language. In Farsi, sense 1 and sense 2 correspond to *hol-daadan* and *feshaar-daadan*, respectively.) Like sense 1, sense 2 prefers the `palm posture`, but its preference is stronger than that of sense 1. Additionally, sense 2 codes for the `force` feature, preferring a `medium` to `high` value.

Why are these senses kept separate, besides intuitively being distinct? If we were to use just one sense, we would lose information. For instance, the probability distribution for `schema` would become SLIDE 50%, TOUCH 50%; and the `posture` distribution would become `palm` 73%, `grasp` 8%, `index` 20%. We would be hiding the important fact that, for the verb *push*, `index posture` is reasonably compatible (30%) with SLIDE but not very compatible (10%) with TOUCH.

Lastly, note that the commonness of sense 1 (amongst all senses in the lexicon) is double that of sense 2. This plays a role when these *push* senses are competing against senses of other verbs to label a new action. Due to its greater commonness, sense 1 need not match the new action as closely as sense 2 would need to in order to beat its competitors.

5.3 Labelling and Obeying Algorithms

The word sense f-struct representation for verbs supports the two basic requirements for use of the verbs, namely *labelling* (in which a linking f-struct summarizing an x-schema execution must be labelled with the best possible verb) and *obeying* (in which a verb must be translated into an appropriate linking f-struct that can guide x-schema execution). These two processes are represented by the upper two arrows in the architectural overview in Figure 1.1. For both cases the mappings between the linking f-struct and linguistic utterances can be given a clean probabilistic account.

5.3.1 Labelling algorithm

For labelling, the goal is to find the verb v with the highest probability, given the trained model m and the linking f-struct l resulting from execution of an x-schema. That is, we seek

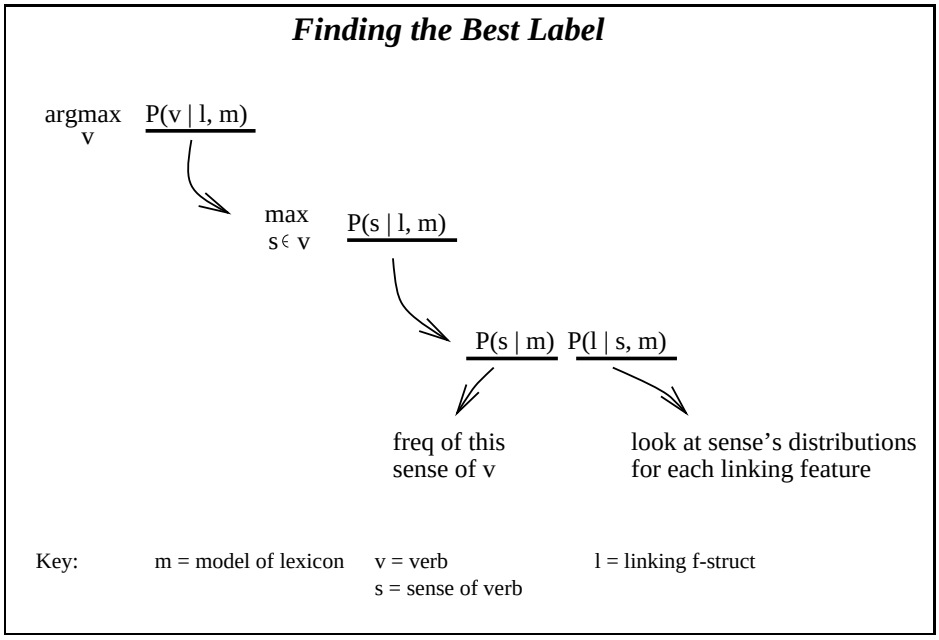


Figure 5.3: Formulas used to determine the best label v for a given linking f-structure l.

$$\operatorname*{argmax}_{v} P(v \mid l, m) \tag{5.1}$$

The computation of this probability is now described and is summarized in Figure 5.3. A simplification is employed in calculating the probability for a verb: the algorithm does not *sum* the probabilities of all the verb's senses s, but instead chooses the probability of the maximum-probability sense. This is a kind of *Viterbi approximation* (Viterbi 1967) and is meant to reflect the notion that only one sense is mentally "settled upon" in response to a stimulus (an action in this case), so low probability on two senses is not equivalent to high probability on one sense. The approximation is

$$P(v \mid l, m) \approx \max_{s \in v} P(s \mid l, m) \tag{5.2}$$

To compute a value for $P(s \mid l, m)$, we apply *Bayes' rule*. Bayes' rule is a simple tautology that follows from the laws of probability, which expresses a conditional probability such as $P(H \mid D)$ in terms of its converse, $P(D \mid H)$. In practice one of these probabilities is often available but the other is desired; hence the rule's tremendous utility. It will be used

extensively in this dissertation, especially for learning. Bayes' rule is:

$$P(H \mid D) = \frac{P(H)\ P(D \mid H)}{P(D)} \tag{5.3}$$

Often H may range over multiple hypotheses while D is the available data and hence is fixed, in which case the denominator is constant and can be ignored, yielding

$$P(H \mid D) \propto P(H)\ P(D \mid H) \tag{5.4}$$

The probability $P(H)$ is called a *prior* and often is determined subjectively. $P(D \mid H)$ is called a *likelihood* and $P(H \mid D)$ is called a *posterior*. Using Bayes' rule we can rewrite $P(s \mid l, m)$ as follows:

$$P(s \mid l, m) \propto P(s \mid m)\ P(l \mid s, m) \tag{5.5}$$

Finally, we have arrived at expressions which can be directly calculated from the statistics contained in the word sense f-struct. $P(s \mid m)$ is simply the frequency value stored in the word sense as described in §5.2. $P(l \mid s, m)$ is calculated directly from the per-feature probability distributions comprising the word sense, by multiplying together the probabilities of the individual feature values in l (this is the independence assumption at work). In a nutshell, the probability for a given word sense is the product of its commonness and its fit to the current action.

These calculations are incorporated into the LABEL algorithm presented below in pseudo-code form:[4]

[4] "Pseudo-code," a somewhat abstract level of algorithm description which is not tied to any particular programming language, will be used to describe all our algorithms. The $\leftarrow$ symbol stands for assignment of the value on the right-hand side to the variable on the left-hand side.

```
LABEL(linking f-struct l, model m) returns a verb
  for each sense s of each verb v in m:
    prior ← relative frequency of s amongst all senses
    likelihood ← product over each feature f in s of:
                    s_f's likelihood of generating l_f
    posterior ← prior × likelihood
  endfor
  if (some sense's posterior ≥ MinLabel)
    then return v corresponding to s with highest posterior
    else return nothing
  endif
end
```

The main loop of the algorithm computes $P(s \mid l, m)$ for every sense of every verb in the lexicon, according to the formulas just described. The if-statement then chooses the sense with the highest probability and returns the verb associated with it—but only if the probability exceeds $MinLabel$, a tunable threshold. Note that since a word sense provides no sharp boundaries on the category it represents, any action will have some residual probability for any verb. So if $MinLabel$ is set to 0, the LABEL algorithm is "forced choice"—that is, it is required to provide an answer no matter how bad it may be. Non-zero values for $MinLabel$ provide a means for leaving unclassifiable actions unlabelled.[5]

5.3.2 Obeying algorithm

Our word sense representation reveals an important advantage when we turn to obeying commands, which involves the reverse mapping from linguistic input to action. The first step of this process, which we discuss here, involves finding the maximum probability motor-parameter linking features p given the command verb v, the current world state linking features w and the lexicon model m. That is, we seek

$$\operatorname*{argmax}_p P(p \mid v, w, m) \tag{5.6}$$

The computation of this probability is now described and is summarized in Figure 5.4. We begin with a simplification similar to the one made in LABEL: we choose not to

[5]The $MinLabel$ parameter will take on more significance when multi-word labels are considered in Chapter 7.

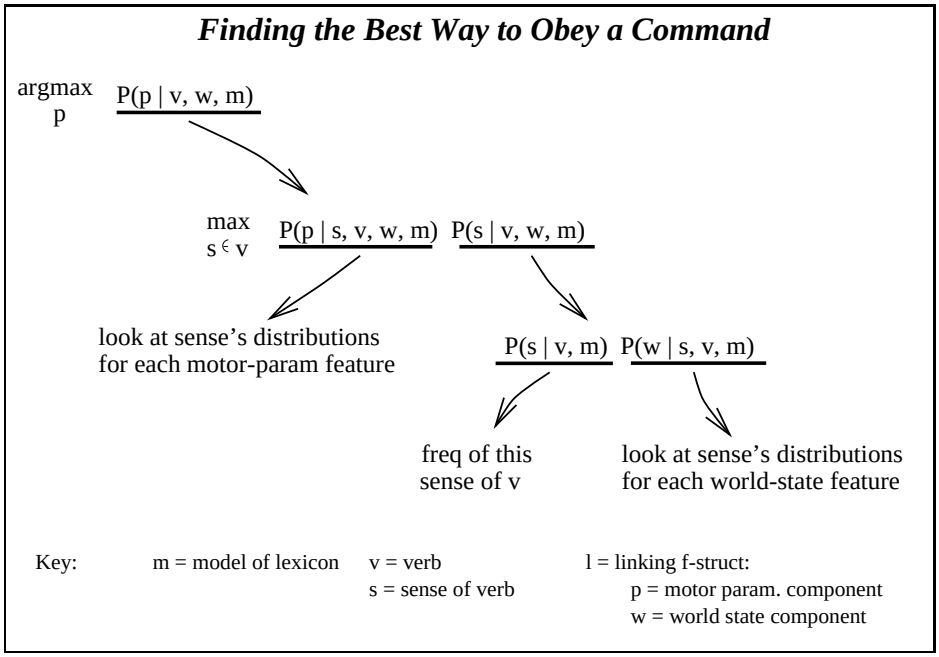


Figure 5.4: Formulas used to determine the best motor parameters p for obeying a command v given initial world state w.

---

sum motor-parameter linking feature probabilities over all senses of v, but instead commit fully to a single sense—namely the sense which best fits w, the initial world state. That is, we choose the sense s as follows:

$$\underset{s \in v}{\text{argmax}}\, P(s \mid v, w, m) \tag{5.7}$$

This expression is transformed via Bayes' rule:

$$P(s \mid v, w, m) \propto P(s \mid v, m)\ P(w \mid s, v, m) \tag{5.8}$$

At this point the computation of the prior can be performed by consulting s's frequency count relative to the total frequency count of all senses of verb v, and the likelihood of the world-state features can be computed from the per-feature probability distributions in word sense s as was done for labelling.

But our goal was p, not the maximal s. That is, we really want to find

$$\operatorname*{argmax}_{p} P(p \mid s, v, w, m) \tag{5.9}$$

This expression is maximized by choosing, for each motor-parameter feature distribution in s, the mode value. However, there are reasons not to retain the full set of motor-parameter features, as will be seen shortly when we review the algorithm.

Pseudo-code for the OBEY algorithm follows:

```
OBEY(verb v, initial world state w, model m)
returns motor-parameter linking features
  for each sense s of v:
    prior ← relative frequency of s amongst senses of v
    likelihood ← product over each world-state feature f in s of:
                   s_f's likelihood of generating w_f
    posterior ← prior × likelihood
  endfor
  if (some sense's posterior ≥ MinObey)
    then let s be the sense with highest posterior
    else return nothing
  endif
  create an empty f-struct p
  for each motor-parameter feature f:
    if (s_f's peakedness ≥ MinSetFeature)
      then p_f ← s_f's mode value
  endfor
  return p
end
```

The first phase of the algorithm can be thought of as a variant of the first step of the LABEL algorithm: we find the sense s (of verb v) which gives the highest probability to the *partial* linking f-struct w containing the current world state. The effect of this procedure is to find the sense of the command verb which best fits the current world state. As with LABEL, a tunable threshold is now employed: $MinObey$. If set to 0 the algorithm will be forced to attempt the action most compatible with the world state even if bad conflicts with the world state are apparent, while a greater than 0 setting allows the algorithm to say "Sorry, Dave, I can't do that."[6]

[6] With apologies to Arthur C. Clarke.

Assuming this test is passed, the next step is to extract the "prototypical" motor-parameter linking features from sense s; that is, for each motor-parameter linking feature, extract its mode value. However, an additional thresholding parameter is introduced here. The sense s may code strongly for some motor parameters, but weakly for others, and we don't want to set features in the linking f-struct if s codes only weakly for them. We will define a measure of *peakedness* of probability distributions, and only those features of s with peakedness exceeding $MinSetFeature$ will be returned from OBEY to be set in the linking f-struct.

The peakedness measure used is the relative probability of the mode value (the highest-probability value) and the *runner-up* (the second-highest-probability value). That is,

$$\text{peakedness} = \frac{P(\text{mode})}{P(\text{runner} - \text{up})} \tag{5.10}$$

This definition of peakedness is obviously only a heuristic since it involves only two of the values in the distribution. It was chosen for simplicity after two other simple definitions failed.

5.4 Connectionist Account

Word senses can be fairly directly mapped into connectionist networks, by using a more localist style of encoding than that found in PDP-style representations. This section sketches such a connectionist architecture. The intent is to demonstrate the plausibility of the architecture. Full specification of the architecture including precise activation functions is not given and remains for future work.

5.4.1 Triangle units

The essential building block is the *triangle unit* (Feldman & Ballard 1982; Diederich 1988), shown in Figure 5.5(a). A triangle unit is an abstraction of a neural circuit which effects a three-way *binding*. In the figure, the units A, B and C represent arbitrary "concepts" which are bound by the triangle unit. All connections shown are bidirectional and excitatory. The activation function of a triangle unit is such that activation on any two of

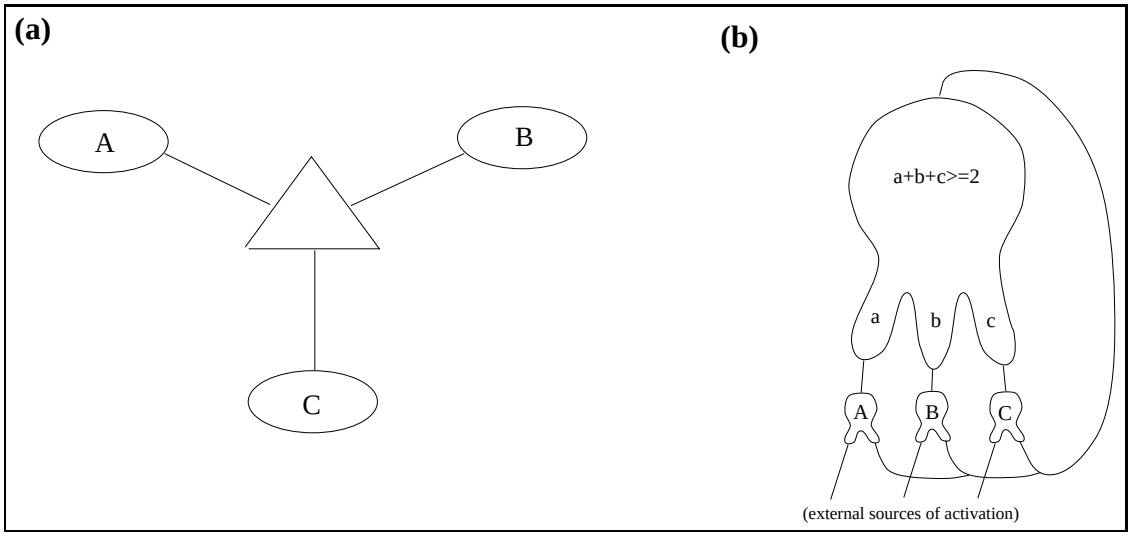


Figure 5.5: (a) A simple triangle unit which binds A, B and C. (b) One possible neural realization.

---

its incoming connections causes an excitatory signal to be sent out over all three outgoing connections. Consequently, the triangle unit allows activation of A and B to trigger C, or activation of A and C to trigger B, etc.

Triangle units will be used here as abstract building blocks, but Figure 5.5(b) illustrates one possible neural realization. A single neuron is employed to implement the binding, and each concept neuron projects onto it. Concept neurons are assumed to fire at a uniform high rate when active and all weights into the main neuron are equal. As a result, each input site of the triangle neuron can be thought of as producing a single 0-or-1 value (shown as lower-case a, b and c) indicating whether its corresponding input neuron is active. The body of the binding neuron then just compares the sum of these three values to the threshold of 2. If the threshold is met, the neuron fires. Its axon projects to all three concept neurons, and the connections are strong enough to activate all of the concept neurons, even those receiving no external input.

A particularly useful type of three-way binding consists of an entity, a feature, and a value for the feature, as shown in Figure 5.6. With this arrangement, if `posture` and `palm` are active, then `"push"` will be activated—a primitive version of the labelling process. Alternatively, if `"push"` and `posture` are active, then `palm` will be activated—a primitive

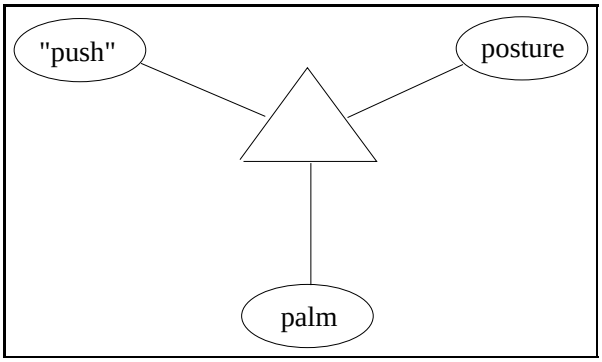


Figure 5.6: Using a triangle unit to represent the value (`palm`) of a feature (`posture`) for an entity (`"push"`).

---

version of obeying. (But note that our final version of labelling and obeying will be more complex than this.)

Generally, a set of these triangle units is connected in a winner-take-all fashion to ensure that only the appropriate binding reaches an activation level sufficiently high to excite its third member. We will soon show how to do that for our language task.

5.4.2 Complex triangle units

But first, we must introduce an extended version of triangle units. In the extended version, multiple connections are allowed on each side of the triangle. Furthermore, these connections can have varying strengths (in both directions). And lastly, these extended triangle units can have graded outputs rather than simple on/off behavior. An example is shown in Figure 5.7(a), in which the B side has three connections and the C side has two. The variable weights are each indicated by a 'w.' The extended triangle unit is useful when more than three entities must be bound, but there is a natural partitioning of the entities into three groups. The function of the extended triangle unit depends on the weights; several usages can be delineated.

One usage is where the desired binding is *conjunctive* over all the connected units (e.g. A, B1, B2, B3, C1 and C2), and the minimum condition for triggering the binding is that *all* the units on *any two* sides of the triangle are active (e.g. A, B1, B2 and B3; or A, C1 and C2). This functionality can be implemented by setting the incoming weights on each

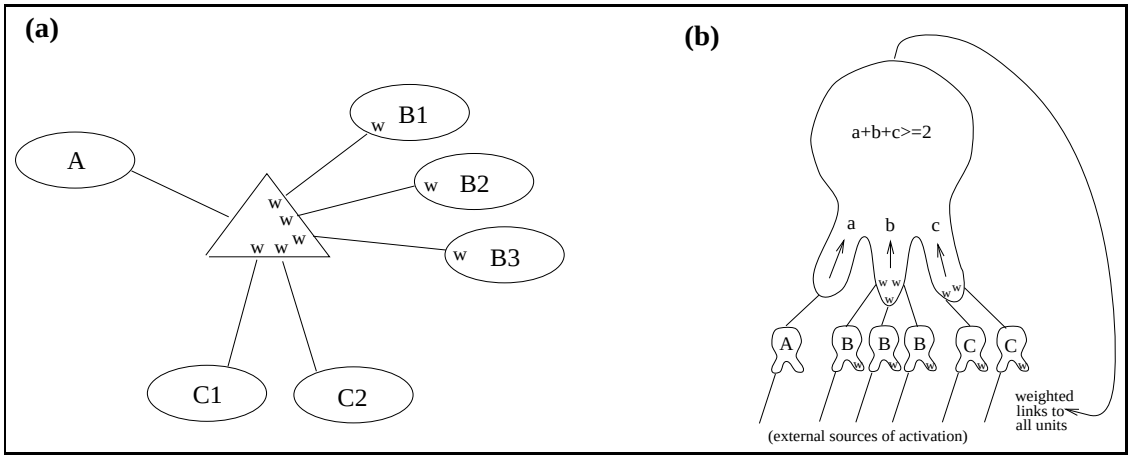


Figure 5.7: (a) A complex triangle unit with multiple weighted connections per side. (b) One possible neural realization.

side to $1/n$, where n is the number of connections on the side, and adding a threshold of 1 locally to each side of the triangle. The central activation function then remains identical to that of the simple triangle unit. A "softer" version of the conjunctive triangle unit can also be implemented, in which one or two missing concept units might lead to weak—but non-zero—activation. Or, if the activation of concept units is variable, then the conjunctive triangle unit can implement a trade-off between the number of active inputs to a side, and their strength of activation. This kind of unit will be used in our connectionist architecture to implement the collection of features in word sense f-structs.

Another usage is more *disjunctive* in nature: at any moment in time, we expect only one concept unit per side to be active, but the activation of the triangle unit should be in proportion to the weights on the links to the active units. (Naturally, in this case we expect the weights to differ from one another.) For example, suppose that B1 and C1 have strong weights and B2, B3 and C2 have weaker weights. Then, should B1 and C1 become active, A will be strongly activated. But if instead B1 and C2 should become active, then A will be only weakly activated. The weighted connections also work in the outgoing direction: if A and C1 should become active, then activation will be sent to all of the B units, although B1 will receive more activation than B2 and B3. This type of triangle unit will be used in our connectionist architecture to implement probabilistic feature values.

Once again, we are not focusing on neural realization here, but an example realiza-

tion of complex triangle units is shown in Figure 5.7(b). We retain the single-neuron design used for the simple triangle unit, but each of the three "lobes" of the neuron, corresponding to the sides of the triangle, must be considered as a full-fledged *site* (Feldman & Ballard 1982).[7] A site is a portion of a neuron's dendritic tree which computes its own local function and passes only the result upstream to the neuron body. These local functions are indicated by the 'w' markings and the arrows pointing to the 'a', 'b' and 'c' quantities summarizing the response of each site. Another difference from the simple case is that the projections of the output axon back to the concept units involves storing another copy of the weights "w" *in the concept units themselves.* This is needed to implement graded activation of the concept units for the disjunctive triangle units just described.

5.4.3 Network architecture

We will now turn to the construction of a network architecture which implements (approximately) our multiple-sense verb representation and its associated algorithms for labelling and obeying. The architecture is shown in Figure 5.8, whose layout is intended to be reminiscent of the upper half of Figure 5.2.

On the top is a "vocabulary" subnetwork containing a unit for each known verb. Each verb is associated with a collection of phonological and morphological details, whose connectionist representation is not considered here but is indicated by the topmost "blob" in the figure. Each verb unit can be thought of as a binding unit which ties together such information. The verb units are connected in a winner-take-all fashion to facilitate choosing the best verb for a given situation.

On the bottom is a collection of subnetworks, one for each linking feature. The collection is divided into two groups. One group—the motor-parameter features—is bidirectionally connected to the motor control system, as described in §4.3 but shown here as a blob for simplicity. The other group—the world-state features—receives connections from the perceptual system, which is not modelled here and is indicated by the bottom-right blob. Each feature subnetwork consists of one unit for each possible value. Within each feature subnetwork, units are connected in a winner-take-all fashion. A separate unit represents each feature itself, apart from its possible values.

[7] If the local thresholding of the conjunctive case is needed, then it may be more appropriate to use separate neurons rather than sites.

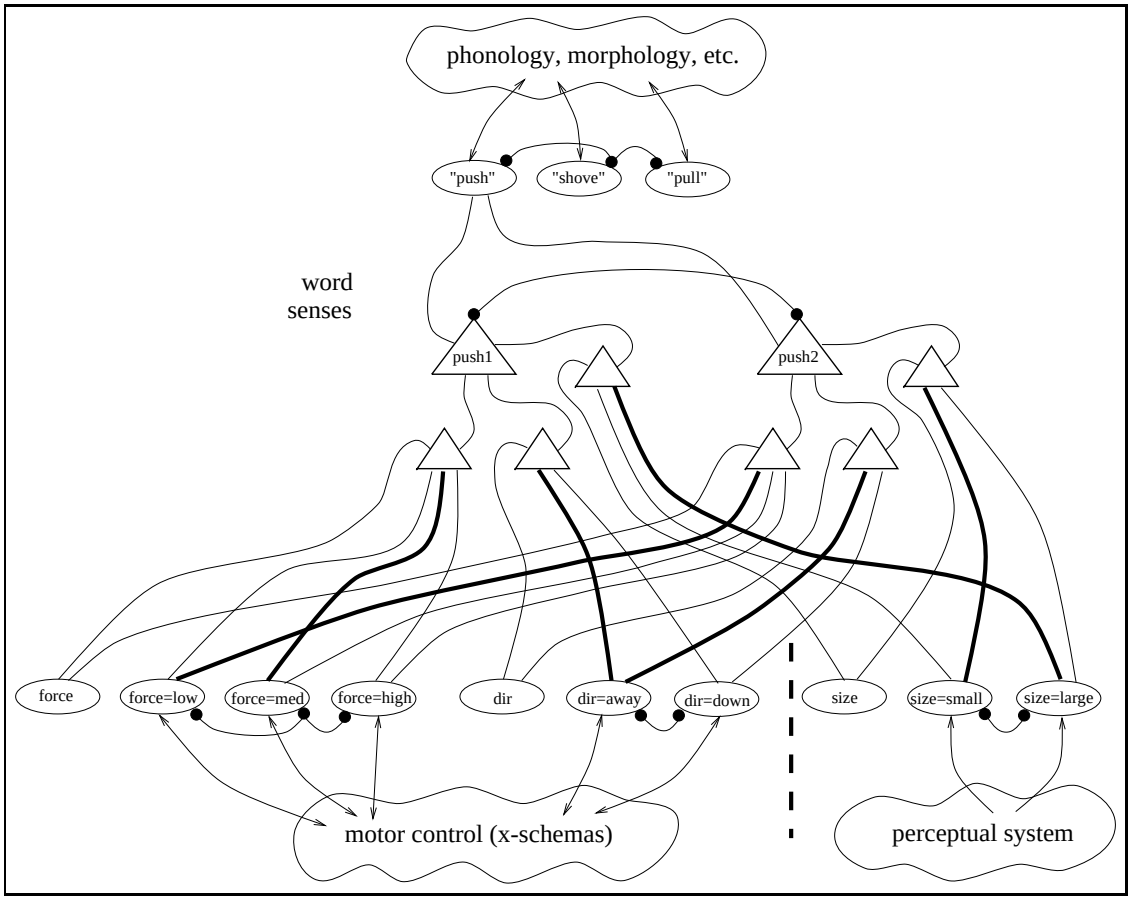


Figure 5.8: A connectionist version of the model, using a collection of triangle units for each word sense.

The most interesting part of the architecture is the circuitry connecting the verb units to the feature units. In the central portion of Figure 5.8 the connectionist representation of two senses of *push* are shown,[8] each demarcated by a box. Each sense requires several triangle units with specialized functions.

One triangle unit for each sense can be thought of as *primary*; these are drawn larger and labelled "push1" and "push2". These units are of the soft conjunctive type described earlier, and serve to integrate information across the features which the sense is concerned about. Their left side connects to the associated verb unit. Their right side has multiple connections to a set of *subsidiary* triangle units, one for each world-state feature (although only one is shown in the figure). The lower side of the primary triangle unit works similarly, but for the motor-parameter features (two are shown in the figure). Recall that, within each side, the weights are equal. However, their magnitude is set to reflect the frequency of the word sense: higher weights are used for more frequent senses, allowing them to be activated more easily.

Each subsidiary triangle unit is of the disjunctive type described earlier, and represents its word sense's probability distribution for a single feature. Its right side connects to the primary triangle unit. Its left side connects to the unit representing the feature itself (e.g. "force"). Most importantly, its lower side connects to each of the value units for the feature. These connections have variable weights representing the preference of the word sense for certain values over others. The response of a subsidiary unit is therefore graded in response to the currently active value; and in the reverse direction, it is also capable of activating value units to differing degrees in accordance with the probability distribution embodied in the weights. In the figure, high-probability links are indicated by thicker lines. Thus it can be seen that sense "push1" corresponds to pushing away a large object with medium force, while sense "push2" corresponds to pushing away a small object with low force. (Clearly these ought to be combined into a single sense; this will be illustrated in the next chapter.)

Lastly, note that the primary triangle units are connected into a lexicon-wide winner-take-all network.

[8] Naturally, the figure illustrates only a subset of the actual features involved in *push*.

5.4.4 Labelling and obeying

We can now illustrate how the network performs labelling and obeying. Essentially, these processes involve providing strong input to two of the three sides of some word sense's primary triangle unit, resulting in activation of the third side.

For labelling, the process begins when x-schema execution and the perceptual system activate the appropriate feature and value units in the lower portion of Figure 5.8. In response—and in parallel—every subsidiary triangle unit connected to an active feature unit weighs the suitability of the currently active value unit according to its learned connection strengths. In turn, these graded responses are delivered to the lower and right-hand sides of each word sense's primary triangle unit. The triangles units become active to varying degrees, depending on the number of activated subsidiary units and their degrees of activation. The winner-take-all mechanism ensures that only one primary unit dominates, and when that occurs the winning primary unit turns on its associated verb unit.

For obeying, we assume one verb unit has been activated (say, by the auditory system) and the appropriate world-state feature and value units have been activated (by the perceptual system). As a result, the only primary triangle units receiving activation on more than one side will be those connected to the command verb. This precipitates a competition amongst those senses to see which has the most strongly active world-state subsidiary triangle units—that is, which sense is most applicable to the current situation. The winner-take-all mechanism boosts the winner and suppresses the others. When the winner's activation peaks, it sends activation to its motor-parameter subsidiary triangle units. These, in turn, will activate the motor-parameter value units in accordance with the learned connection strengths. Commonly this will result in partial activation on multiple values for some features. The winner-take-all mechanism within each feature subnetwork chooses a winner. (Alternatively, we might prefer to preserve the distributed activation pattern for use by smarter x-schemas which can reason over probabilistic specification of parameters. E.g., if all the force value units are weakly active, the x-schema knows it can choose any suitable amount of force.)

5.5 Some Cognitive Linguistics Issues Considered

Now that we have fully specified a model of verb representation and use, it is time to take stock of how the model relates to some important ideas in linguistics. For example, evidence from cognitive linguistics (Lakoff 1987) and also from psychology (Rosch 1977; Rosch *et al.* 1976) suggests that human categories exhibit internal structure that would not be captured by the classical representation using necessary and sufficient conditions. Human categories exhibit prototype effects, radial structure and basic-level effects. We will see that our model captures some of these phenomena, but falls short of a full account, suggesting some avenues for future work. On another front, the traditional linguistics literature decomposes meaning into pragmatics and semantics, each with its own properties. We will see that this distinction has a nice interpretation in terms of our model.

5.5.1 Prototype effects

One characteristic of human categorization is that it exhibits *prototype effects*. With one kind of prototype—graded prototypes—certain members of the category are marked as especially "good" examples of the category and other members are judged by how much they differ from the prototypical members. (In the psychology literature, degree of prototypicality is measured by explicit ratings, by frequencies with which a member is listed as belonging to a category, or by reaction times for judging membership.) Our word sense representation exhibits several of the main graded prototype effects. Each sense implicitly includes a prototype, namely the f-struct with maximum probability according to the distributions in the word sense. The potential for multiple senses corresponds to multiple-prototype categories. Next, the probability distributions give a measure of the degree of goodness of a non-prototypical f-struct. Lastly, the connectionist implementation involves a winner-take-all step which will converge more slowly for lower-probability f-structs than for higher-probability (i.e. prototypical) f-structs during categorization, in accordance with the slower reaction times observed for classifying non-prototypical examples.

5.5.2 Radial categories

According to radial category theory, conceptual representations consist of not only a set of graded prototypes, but also a structure connecting the various prototypes. The central tenet of radial category theory is that the multiple prototypes of a category are not arbitrary, but are related to each other in particular ways. These relations include image-schematic transformations, metaphorical links and metonymic links. If there is one central sense plus several others which each derive from the central sense by one such transformation, the category structure is radial, hence the term. However, other topologies are possible.

Beyond transformations relating verbal senses of a word, there is a wider range of extensions which include changes in part of speech, and these also exhibit various regularities (Wilensky 1991). For example, part of the meaning of *pocket* as a noun carries over to its interpretation as a verb. Presumably there is a rule to the effect that a noun referring to "X" may also be used as a verb to denote some kind of action involving "X," subject to some restrictions. In this example, combined with appropriate world knowledge, this rule leads to the verb meaning "INSERT-INTO(POCKET)". This is an example of a metonymic transformation.

The current model *doesn't* model such category structures, in that we have included no account of these connections. For our relatively limited modelling task, we must ask what benefit is to be gained from representing the category structure as opposed to merely a catalog of possible uses.[9] The primary benefit may be greater compactness. One could, for example, represent the central sense just the way we have been doing in our model, but *delta code* the other senses, i.e. represent them by the (presumably comparatively short) list of features on which they differ from the central sense. This strategy would require significant changes to our learning algorithm (Chapter 6) which would then be charged with the task of identifying which sense is central.

A potentially more interesting version of radial category representation would be to explicitly represent a fixed set of transformations (at the feature level) motivated by linguistic analysis. In such a model, each non-central sense would be represented simply by

[9]Generativity is one potential benefit. If the applicability conditions of the various transformations could be learned, then novel word extensions could be generated on demand. However this may not be feasible, since extended senses, while usually motivated, are not fully predictable.

a link to the sense it is derived from, plus the identity of the relevant transformation. This is effectively a procedural representation of the non-central senses, requiring the indicated transformation to be applied each time the sense is needed. The learning task in this case is a very open question.

Also open is the question of what sorts of transformations apply to the action domain. Metaphorical and metonymic links are quite numerous and often involve reasoning outside the action domain, and thus are hard to incorporate into a learning story at the present time. But we hope that, within the action domain, a relatively small list of *motor-schematic transformations* (analogues to the image-schematic transformations such as end-point focus, mass-multiplex duality, etc. (Lakoff 1987)) could be identified by future linguistic research and used to construct a radial-category-based learning algorithm as just described.

5.5.3 Basic-level effects

Another proposed structural characteristic of human categorization is that in the hierarchy of specific to general conceptual levels, there exists a privileged level called the *basic level* (Rosch *et al.* 1976). Categories at the basic level (the standard example is "chair") enjoy a privileged position in that they are conducive to mental imagery yet are abstract enough to be useful for common reasoning tasks. The basic level is not the lowest level of the concept hierarchy—categories at the more specific "subordinate levels" exhibit a fine level of detail which is not essential for most reasoning (e.g. "ottoman"). Nor is the basic level the highest level of the hierarchy—categories at these "superordinate levels" are too abstract for mental imagery (e.g. "furniture"). Basic-level categories are more easily learned and are more commonly used in reflexive reasoning.

There tends to be disagreement about exactly where the basic level lies in the various domains of human experience. This suggests that perhaps the basic level is not entirely universal, and thus not tied intimately to biology. Revisions to the original theory should allow for the basic level to be determined relative to experience. For an antique furniture salesperson, "ottoman" may be at the basic level. Such revised theories are still at a preliminary stage; nevertheless, it is worth considering how the current model of actions can be made to fit into the story.

First, consider how our model represents categories at varying levels of abstraction. The simplest categories are those which simply code for a specific x-schema, such as *lift*. These, we argue, correspond to the basic level. Choosing x-schemas appropriately to achieve your goals is the kind of everyday reasoning which characterizes the basic level. Furthermore, if we assume that mental imagery or gestalt perception of actions consists of "mental" x-schema executions (i.e. disconnected from the muscles), then specifying an x-schema with no further constraints is the highest level at which such imagery could occur. Lastly, such categories are relatively easy to learn in our model, since only one feature—the x-schema name—is involved, and a single value is called for. (The learning algorithm is usually initialized with a predisposition toward peaked distributions for the `schema` feature but broader distributions for the other motor-parameter features. See the next chapter for details.)

At the subordinate level, we have categories which code for both a specific x-schema and some parameter values, such as *heave* which specifies `schema = LIFT` and `force = high`. Such categories are somewhat slower to learn, since probability distributions must be learned over multiple features, and motor-parameter features such as `force` are initially biased toward broad probability distributions (see next chapter). And the parameter probability distributions may be more complicated than simply selecting a single value; if we had a finer resolution in our `force` feature, we might have to learn that *heave* calls for "`force = 6 to 10, with 8 best`" or somesuch.

At the superordinate level, our model can learn categories which map to *sets* of x-schemas. For example, *move* maps to any invocation of SLIDE, LIFT, DROP, etc. Such categories are slightly harder to learn than basic-level categories since a broad probability distribution for the `schema` feature must be learned (again, this is against the normal bias toward selecting a single x-schema). They become even more cumbersome, however, if the category specifies parameter settings which differ for the different x-schemas, for in this case the category can be represented only by multiple word senses.

So far, we've been assuming that there are separate hierarchies for objects and for actions, each with its own basic level. Yet it may be fruitful to consider (action, object) pairings as basic, since actions and objects each clearly exert an influence on how the other is conceptualized. Such an interactionist account of basic-level phenomena, incorporating an x-schematic model of actions connected to some sort of affordances-based model of objects,

would be an intriguing modelling endeavor.

5.5.4 Pragmatics

It has been argued that many apparent problems of semantics become simpler if the role of pragmatics is considered. Pragmatics refers to the role in interpretation played by context—both linguistic and external. The model we have presented offers an illustration of how world-state context effects can be separated from semantics, simplifying the latter.

The key observation is that x-schemas can be heavily branched, allowing them to function in a wide variety of world states, yet most of this does not appear in the linking f-struct and hence is not directly available linguistically. In the extreme case, a word like *slide* can be represented merely by a pointer to the SLIDE x-schema; when carrying out a *slide* command all the movements, grip choices, obstacles avoided and other such decisions are handled down at the x-schema level. Thus, a wide array of possible behaviors may be associated with the verb *slide* while retaining a very simple semantic representation for the verb.

5.6 Limitations of the Model

The labelling and obeying algorithms assume independence amongst the linking features. Such independence assumptions are commonly needed to make probabilistic models tractable. Fortunately independence often holds, and when it doesn't the model can compensate by having a larger number of word senses. Yet clearly there is a cost to the system's inability to, e.g., represent in a single sense that both `posture` and `size` have broad distributions but are highly correlated (e.g. `small` objects tend to be `pinch`ed but `large` objects tend to be `grasp`ed).

So the limitation is not so much the independence assumption but the representation itself, i.e. flat attribute-value lists. Such a representation is vastly less expressive than, for instance, predicate calculus, since relationships amongst entities cannot be represented. The ability to represent something like "force is proportional to acceleration times object weight" would allow richer concepts to be (naturally) represented.

A different limitation arising from the architecture of the system is that any er-

rors made by the OBEY algorithm (which, remember, chooses all linking features before beginning execution)—perhaps due to incorrect estimates of the world state—cannot be undone during execution. While x-schemas include branches for handling some anomalous conditions, there is currently no mechanism for discovering early in the x-schema execution that the chosen verb sense is unimplementable and a different sense should be chosen.

With a richer representation system, the OBEY routine's very simple conflict resolution strategy could be replaced with an algorithm capable of general planning, which is indeed necessary even in interpreting simple verb phrases. Work in this area by Levison (1995) has investigated the role of objects in choosing among alternate realizations of a verb (e.g. *open the door* compared with *open the jar* or *open the box* (Levison 1993)). Levison's Object Specific Reasoner demonstrates the importance of the planning level, thereby pointing out some limitations to an approach such as ours with its restricted focus on the motor level.

5.7 Continuous Distributions

Characterizing probability distributions by multinomials over each possible value becomes unwieldy for inherently continuous-valued features when the granularity of discretization is increased. In this case, it becomes highly desirable to find a functional form for the distribution with a small number of parameters. When a distribution is modelled as a Gaussian, we no longer need worry that a particular value will, upon being absent in the training data, receive low probability despite the presence of high probability on nearby values.

Continuous distributions are not included in the current version of the model, but were included in an earlier version. The types of distributions that were found to be useful were Gaussians (for linear features such as `force`) and circular Gaussians (for features such as `direction`).

In a continuous probability distribution, the probability of any particular value is zero; probabilities are meaningful only over an interval. Consequently, when calculating the probability of a particular feature structure given a word sense, we use a standardized interval for all the continuous features. In other words, a force value of 6.24 is interpreted as $6.24 \pm \delta$, where δ is fixed in advance. While the resulting probabilities will depend on

the arguably arbitrary choice of δ, all *relative* probabilities are still legitimate, so the choice of the best label for an action will not be affected. However, in LABEL and OBEY, the $MinLabel$ and $MinObey$ thresholds must be adjusted whenever the standard interval is adjusted, since they represent actual—not relative—probabilities.

An approximation to Gaussians can be added to the connectionist account of word senses by modifying the connectionist network in §5.4 as follows. For each numerical feature, the value units are connected in a topographic fashion. That is, excitatory connections link value units which represent quantities of similar magnitude, with stronger connections for values which are closer together. As a result, when activation is present on one value unit, nearby values will receive partial activation. This structure is needed because word sense triangle units can *not* act like radial basis function (RBF) units (Moody & Darken 1988) commonly used in neural modelling to represent ellipsoidal regions in continuous spaces. RBF units rely on each continuous feature being represented by degree of activation rather than place coding. Since, with place coding, there is no way for word sense triangle units to "know" about the distances between the quantities represented by each feature value, we need the excitatory connections amongst value units to represent those relationships.

Some pilot training experiments with Gaussian distributions revealed an important limitation imposed by such restrictions on the form of probability distributions. The word in question was *sideways* (see Chapter 7 on including directional specifiers in the model). The relevant feature was `direction`. Unlike *away* or *toward*, which prefer a single direction with some amount of tolerance on each side, *sideways* has two best directions—left and right—but does not apply to intermediate directions. Such a bimodal distribution cannot be modelled as a Gaussian. The result, during learning, was two separate senses, one for left and one for right. While this may be acceptable for toy examples, it will not scale well, particularly if many features exhibit such bimodality. Simple mathematical models such as Gaussians are convenient to design algorithms for, but don't capture the complexities of language.

*****

In this chapter we have developed the idea that probability distributions over linking features derived from x-schema execution can capture the needed functionality for both labelling actions and carrying out verbal commands. But we were able to do so only

by positing multiple, conjunctive senses for each verb. In the next chapter, we will turn to the question of learning these senses, including the important subproblem of determining *how many* senses are called for.

Chapter 6

Verb Learning

This chapter is the core of the thesis. It reviews some facts about children's lexical development and then incorporates them into a Bayesian model merging algorithm which learns an appropriate set of word senses from a set of labelled actions.

6.1 Children's Verb Acquisition

Lexical acquisition has been studied extensively in psychology. Yet most work has focused on acquisition of nouns to the exclusion of other categories such as verbs. Why? Children do acquire verbs and other parts of speech early in their lexical development (Nelson 1973). A recent volume (Tomasello 1995a) proposes some answers to this question

and then reviews recent efforts to borrow and adapt ideas from noun acquisition (e.g. Markman's principle of mutual exclusivity (Markman 1989)) to explain acquisition of other parts of speech. One reason cited for the lack of research on acquiring verb semantics is that verbs are seen as the stepping stone to acquisition of grammar, which of course has long fascinated the research community and presents its own issues (such as understanding argument structure) which overshadow the inherent semantics of the verbs themselves.

Another reason, though, is that the verb acquisition problem just seems harder than noun acquisition. One difficulty arises from the temporal nature of actions. The referent of an action verb is usually offset in time from hearing the verb, unlike nouns where the parent can point to the object while vocalizing its label. Actions are also fleeting, not available for extended contemplation the way a just-labelled object is. Another difficulty is that action verbs involve not only perceptually available information (when watching someone else act) but also proprioception of internal states (when acting oneself). In particular, many verbs imply goal-directedness and thus can't be properly learned until goal-directed behavior begins! And lastly, in languages like English verbs are phonologically less salient than nouns because they are buried in the middle of sentences rather than appearing at the end. (Verbs appear to be learned earlier in verb-final languages like Korean (Gopnik & Choi 1995).)

How does the child surmount these obstacles? On the issue of temporal difficulties, it has been found that in the formative second year of life children most often hear verbs from their mothers *before* the corresponding action (Tomasello 1992). Indeed, learning is slower if this condition is experimentally altered. Early verbs also typically label short-duration actions. These two facts simplify the task of segmenting the proper time window for the verb and alert the child to attend to important features of the action.[1] The need to attend simultaneously to both action and speech is also relieved. Our model incorporates this simplification by using a static linking f-struct which (1) *summarizes* (i.e. remembers) an action, (2) implicitly provides the proper segmentation, and (3) does not encode the timing of the label relative to the action. On the issue of perception *vs.* proprioception, it turns out that a disproportionate fraction of children's early exposures to verbs refer to their own activities rather than someone else's (Tomasello 1992; Huttenlocher *et al.* 1983).

[1] But see Tomasello (1995b) for further pragmatic and social factors in determining proper reference for actions.

Thus the difficult correspondence problem can be delayed until later.[2]

But there are other significant obstacles that cannot be so easily explained in terms of restrictions on the task itself. These must be addressed by any proposed learning algorithm if it is to be psychologically plausible. One such obstacle is the accepted observation that children generally receive negligible negative evidence during learning (for a review see e.g. Marcus (1993)). That is, they hear examples of words used properly, but don't hear improperly used words and aren't corrected when they improperly use a word themselves. Basic results in computational learning theory (Gold 1967; Mitchell 1980) demonstrate that learning in this circumstance is impossible without a bias specifying preferences amongst those concepts consistent with the data.

Another obstacle—or challenge rather—is *fast mapping*, the ability of children to understand and use a word in a reasonable way after as few as one exposure to it from the parent (Carey 1978). (See Heibeck & Markman (1987) for a more recent treatment or Flavell *et al.* (1993: pages 301-302) for a thoughtful overview.) The key point of fast mapping is that the child hazards a guess as to the relevant features of the single example rather than waiting to collect statistics across multiple examples. The guess may be based on linguistic context ("no it's not purple, it's *mauve*"), parental cues (pointing) or, as modelled here, innate biases regarding what is linguistically relevant.[3]

These constraints together motivate the probabilistic model merging algorithm presented next for the learning of action verbs.

6.2 Learning Word Senses via Model Merging

Recall that the learning task is to construct a model lexicon—that is, an appropriate set of word senses for each verb—from a training set consisting of labelled actions. Each training example consists of a verb and a linking f-struct summarizing an action. The verb representation consists of probability distributions, and thus we frame the problem of verb acquisition in terms of probabilistic model inference. The basic idea is to start with a lot of very specific senses for each verb, and then gradually merge them together to form a

[2]The correspondence is also likely to be partially innate. Neonates are capable of mimicking facial expressions (Meltzoff & Moore 1977).

[3]It should be noted that while fast mapping is well-established for a few domains, it has not been carefully studied for the case of verbs.

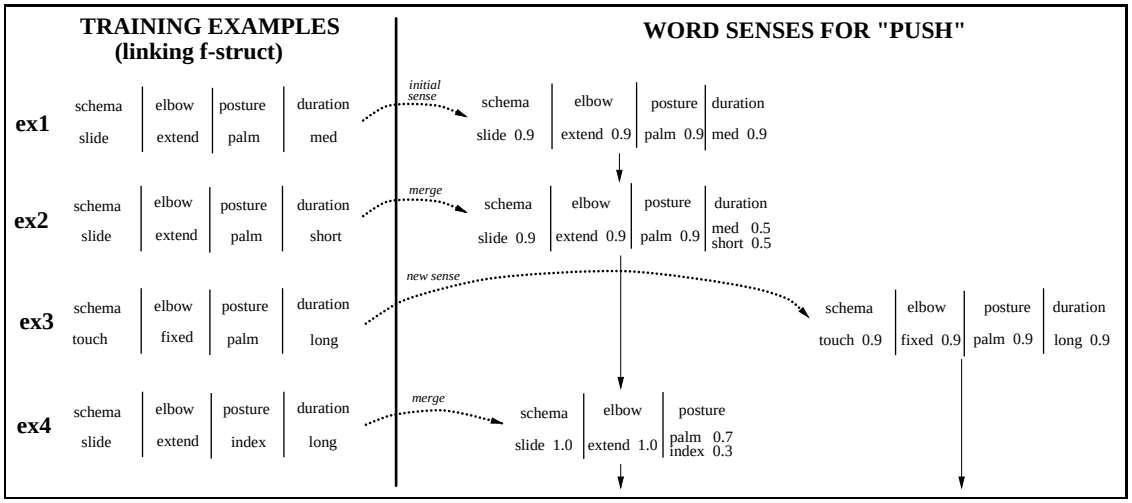


Figure 6.1: Learning two senses of *push*.

smaller set of more general senses.

6.2.1 An illustration of merging

We will introduce the procedure by illustrating a hypothetical training run. Figure 6.1 depicts how the learning of two word senses for *push* might proceed. Importantly, the figure illustrates *online* learning: each training example is incorporated into the model as soon as it occurs. In contrast, the algorithm description in the following sections is oriented toward the offline case, where all training examples are collected before any merging occurs (the description is cleaner this way), but the algorithm can operate in an online fashion.[4]

The left-hand side of each row in Figure 6.1 shows a linking f-struct for a new example labelled *push*. The right-hand side shows the modified semantic representation for *push* after incorporation of the example.

The first training example (which might correspond to pushing a cube across a table) necessarily entails creation of an initial word sense. This initial sense closely reflects the example itself, except that the feature values are probabilistic. Each feature value observed in the example f-struct is assigned probability *slightly less* than 1.0 (they are

[4] In particular, the fully-online case corresponds to setting the algorithm's *BatchSize* parameter to 1.

shown as 0.9 in the figure). The remaining probability is divided amongst the unobserved possible values of the feature. A very modest amount of generalization has thus occurred.

The second example of *push* is quite similar to the first, differing on only one feature—`duration`. So, it is merged into the existing word sense, thereby generalizing that sense's `duration` probability distribution.

The third example corresponds to a different kind of *push*ing, perhaps pushing against a wall. This example differs from the current word sense on almost every feature, and thus is deemed too different for merging. Instead a new word sense is generated (shown in the right-most column).

Finally, the fourth example arrives. It is a sliding motion like the first two examples, but involves the `index` finger `posture`. It is compared against *both* existing word senses and is judged most similar to the left-hand sense. It is then merged with that sense. Since the `schema` and `elbow` features have been consistent for all three examples incorporated into this left-hand word sense, the probabilities on these feature values are now virtually 1.0. The `posture` feature, however, has been generalized so that it favors `palm` but allows `index`. `Duration`, has proved not to be criterial for this word sense and has been dropped.

6.2.2 A Bayesian criterion

We now turn to a more formal treatment of the learning algorithm. The learning task is an optimization problem, in that we seek, amongst all possible lexicons, the "best" one given the training set. First off, then, we must precisely define "best": we wish to find the lexicon model m which is most probable given the training data t. That is, we seek

$$\operatorname*{argmax}_m P(m \mid t) \tag{6.1}$$

The probability being maximized is the *a posteriori* probability of the model, and our algorithm is a "maximum a posteriori (MAP) estimator" in statistics parlance. The fundamental insight of Bayesian learning is that this quantity can be decomposed using Bayes' rule into components which separate the fit to the training data and an *a priori* preference for certain models over others. The calculations which follow are summarized in Figure 6.2. The first transformation is:

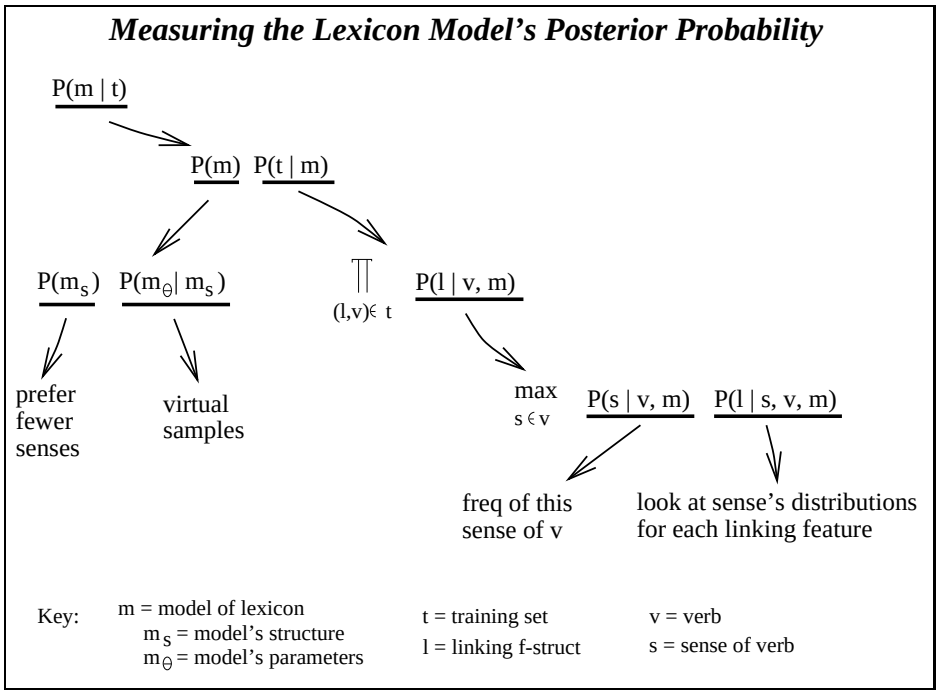


Figure 6.2: Formulas used to determine the posterior probability of a lexicon model m.

$$P(m \mid t) \propto P(m)\ P(t \mid m) \tag{6.2}$$

Let's begin with the second term, $P(t \mid m)$. This term, the *likelihood*, is the probability of the model generating the data. It is a measure of the degree of fit of the model to the training data. To gain intuition for this, one should imagine a word sense f-struct with its probability distributions as a random generator of linking f-structs. The likelihood computes the probability of the word senses in m generating exactly the set of linking f-structs found in the training set t. Thus, we can rewrite the likelihood as follows:

$$P(t \mid m) = \prod_{(l,v) \in t} P(l \mid v, m) \tag{6.3}$$

The individual probabilities are computed using the same Viterbi-like approximation as used for labelling and obeying; that is,

$$P(l \mid v, m) \approx \max_{s \in v} P(s \mid v, m)\ P(l \mid s, v, m) \tag{6.4}$$

and as before, the first term is computed from the sense s's stored frequency count and the second term is computed by consulting the probability tables in s for each feature in l and multiplying.

Now back to the first term in Equation 6.2, $P(m)$. This term, the *prior*, is how subjective preferences are hooked into the learning procedure. While any kind of probability distribution over models can be used in the Bayesian framework, the current work will use only one kind of prior. Our prior has two parts, one dealing with the structure of the lexicon model (m_s) and one dealing with the probability distributions within each word sense, given a particular structure (m_θ):

$$P(m) = P(m_s)\ P(m_\theta \mid m_s) \tag{6.5}$$

The structural component is used to express the subjective preference that, all else being equal, we should prefer a model with fewer senses per verb. We use:

$$P(m_s) \propto\ e^{-ModelPriorWeight \times |m|} \tag{6.6}$$

where $\mid m \mid$ denotes the number of senses in the lexicon model m and $ModelPriorWeight$ is a tunable parameter which adjusts the strength of this bias toward fewer senses.[5] To turn the proportionality into an equality, a normalization must be included to ensure that the sum over all possible model structures equals 1. This normalization factor is

$$e^{ModelPriorWeight}(1 - e^{-ModelPriorWeight})$$

Because of its exponential form, this structural prior has the characteristic that the ratio of the prior on an $(i-1)$-sense model compared to an i-sense model is independent of i and is always equal to $e^{ModelPriorWeight}$. This value effectively specifies the maximum amount by which the likelihood is allowed to drop during each merge.

The m_θ component takes the form of a Dirichlet prior[6], a common choice of prior for multinomial distributions because it is mathematically convenient and has an intuitive interpretation. The intuition behind the Dirichlet prior is that it acts as if a number of *virtual samples* had been observed prior to the actual observed samples. Typically the virtual samples are distributed over the possible values of the multinomial for the purpose of avoiding "drastic" conclusions from limited data. The Dirichlet prior is parameterized, and the parameters effectively specify the number of virtual samples for each possible value. The greater the number of virtual samples, the more observations will be needed before the MAP estimate will converge to the frequencies observed in the data. Consider an example: we would not want to conclude that a coin is unfair (probability of heads equal to 1) based on a single flip yielding heads! Instead, suppose we assign 5 virtual samples to each of heads and tails. This constitutes an initial belief that the coin is fair (probability of heads equal to 5/10). A single flip yielding heads would then lead to an estimated probability of heads equal to 6/11, i.e. we are still more or less convinced the coin is fair. But if 100 consecutive flips yield heads then this probability increases to 105/110 and we have pretty much concluded the coin is unfair.

The above calculations are summed up in the following pseudo-code for the subroutine COMPUTE_MODEL_POSTERIOR:

[5] This prior can also be thought of as a description length prior (Rissanen 1984) and the full algorithm can be thought of as minimizing the combined description length of the lexicon model and the training set.

[6] See Stolcke (1994:p. 23) for a well-written full definition of the Dirichlet distribution and discussion of its properties.

```
COMPUTE_MODEL_POSTERIOR(model m) returns a probability
  prior ← e^{-ModelPriorWeight × |m|}
  likelihood ← product over all examples (l, v) of P(l | v, m)
  return prior × likelihood
end
```

In summary, we have provided the groundwork for biased estimation of the training set labeller's lexicon, in accordance with a specified subjective prior. The prior has a structural component that will be employed by the learning algorithm to make discrete changes in the model structure during search.

6.2.3 Model merging

So we know what we must maximize; what is our search procedure? We use model merging (Omohundro 1992), a hill-climbing algorithm which has proven useful in learning stochastic grammars and Hidden Markov Models. The idea is to start with a model which generates exactly and only the training data, hence maximizing the likelihood. Then, the model is transformed in discrete steps which increase the posterior probability, i.e. the product of the prior and the likelihood. Usually, each such step raises the prior but lowers the likelihood. Eventually a step will occur where the increase in the prior is more than offset by the decrease in likelihood, at which time the algorithm terminates.

In general terms, the algorithm is:

Model merging algorithm:

1. Create a simple model for each example in the training set.
2. Repeat the following until the posterior probability decreases:
   (a) Find the best candidate pair of models to merge.
   (b) Merge the two models to form a possibly more complex model, and remove the original models.

In our case, "model" in the name "model merging" refers to an individual word sense f-struct. Our learning algorithm creates a separate word sense for every occurrence of

a word, and then merges these word sense f-structs so long as the reduction in the number of word senses outweighs the loss of training-set likelihood resulting from the merge.

A major advantage of the model merging algorithm is that it is one-shot. After a single training example for a new verb, the system is capable of using the verb in a meaningful, albeit limited, way. Omohundro (1992) argues for the cognitive plausibility of this learning strategy, which essentially calls for memorizing individual experiences when they are few, followed by a gradual transition to postulating generalities as enough experiences accumulate to warrant it. Model merging is also relatively efficient since it does not backtrack. Yet it often successfully avoids poor local minima because its bottom-up rather than top-down strategy is less likely to make premature irreversible commitments.

One might ask, why merge at all? Why not just keep the initial set of word sense f-structs? There are several answers. One is generalization; without the broadened probability distributions resulting from merging, the lexicon model would have no way of extending its use of verbs (for both obeying and labelling) beyond the set of situations in which it has previously observed the verb used. Another is memory limitations; one cannot store every instance of every word one hears! And lastly, concise representations are more suitable for reasoning since they are easier to "inspect".

6.2.4 Algorithm details

We now turn to details of our implementation of the model merging algorithm described in general terms in the previous section.

Creating initial word sense f-structs

We begin with the first step, which creates initial models corresponding to each training example. The following pseudo-code describes how this is done:

```
CREATE_NEW_SENSE(linking f-struct l, virtual samples table virt) returns a sense
  create an "empty" sense s
  example-count of s ← 1
  for each feature f of l:
    create a probability distribution d_f
    set each of d_f's value counts to virt_f
    add 1 to d_f's value count for the value of l_f
    insert d_f into s
  endfor
  return s
end
```

A very important point is that the initial word sense f-struct we create is not exactly the maximum likelihood sense for the observed linking f-struct. Instead, the initial sense incorporates a number of virtual samples for every value of every feature, effectively "blurring" the probability distributions. The virtual sample values are specified in a table *virt* which has an entry for each feature. The *virt* table is specified externally and thus provides another way to tune our algorithms. For most training runs, we use a small value for the `schema` feature, since verbs often correspond to one particular x-schema. Larger values are used for the other features.

Two advantages follow from this step, which essentially inserts a little generalization right off the bat. The basic point is that by adding the virtual samples the algorithm avoids zero probabilities on unseen feature values, and thereby allows probabilities over entire f-structs to be a useful metric of their similarity. When labelling, this metric makes it possible for a new verb supported only by such an initial sense to be chosen as the label for a linking f-struct even if the match is not perfect. In this way, fast mapping with modest generalization is achieved. And as we will see shortly, the ability to define a cheaply computed similarity metric between word sense f-structs can speed up merging.

Finding the best merge candidate

Next, we consider how the algorithm will choose at each iteration which pair of word senses to merge. Merging can be performed only between senses of the same verb. For a given verb, we generate all possible pairs of senses and then call the routine FIND_BEST_CANDIDATE_MERGE with this set:

```
FIND_BEST_CANDIDATE_MERGE(set of senses S) returns two senses
  for each pair of senses a, b in S:
    for each feature f:
      compute ChiSquare_f(a_f, b_f)
    endfor
    similarity_ab ← e^(−Σ_f ChiSquare_f)
  endfor
  if (highest observed similarity exceeds MinMerge)
    then return a, b with highest similarity_ab
    else return nothing
  endif
end
```

A couple points deserve comment. First, the routine uses a measure of *similarity* to choose the best merge pair. Ideally, we would calculate the new posterior which would result from each potential merge, since that is the quantity we wish to maximize. However, this is prohibitively expensive. The similarity heuristic is intended to be a cheaper way to estimate the improvement in the posterior for a given candidate merge. Since the structural prior depends solely on the number of word senses, any merge will increase it by the same amount. Thus, the similarity heuristic is designed simply to minimize the loss in likelihood.

The similarity measure used here is based on the *chi-square* statistic. The chi-square statistic of two distributions R and S is defined as

$$ChiSquare = \sum_i \frac{(R_i - S_i)^2}{R_i + S_i} \tag{6.7}$$

where i ranges over possible values of the distributions and R_i and S_i represent the occurrence counts for each value in the two distributions. This measure is similar to a least squares error function. A problem with this measure is that it is sensitive to differences in the total number of samples in the two distributions. Often we will want to merge two word sense f-structs which "cover" very different numbers of examples, such as when a new sense has just been created for a new training example and the algorithm is attempting to merge it into an already mature set of senses. So we cannot use the statistic in this form, but instead modify it so that R_i and S_i represent probabilities of each value rather than counts.

The modified chi-square statistic is applied to each individual feature. To achieve an overall similarity statistic for two word senses, these measures for each feature are summed, which is a reasonable combination function since the measure is additive in nature. Finally, the sum is pushed through the negative exponential function in order to yield a measure which tends toward 1 for identical distributions and toward 0 for totally dissimilar distributions.

Another noteworthy feature of FIND_BEST_CANDIDATE_MERGE is that it employs a similarity threshold, $MinMerge$. If the best candidate merge exhibits similarity lower than this threshold, then no candidate will be returned and merging will stop. This thresholding heuristic can be turned off by setting $MinMerge$ to 0 but often it proves useful to set it to a small but positive value to filter undesirable merges.

The computation of similarities for every pair of senses is time-consuming. Accordingly, the implemented version of FIND_BEST_CANDIDATE_MERGE employs a priority queue to store these potential merges, ordered by their similarity, so that they needn't be recomputed each time merging is performed. The priority queue implementation has an additional routine to update the queue after a merge: some old potential merges must be deleted, and some new potential merges added.

Performing the merge

Once a candidate merge is selected, the actual merge operation goes as follows:

```
PERFORM_MERGE(model m, senses s1 and s2) modifies m
  create an "empty" sense s12
  example-count of s12 ← sum of the example-counts of s1 and s2
  for each feature f of s1 and s2:
    create a probability distribution df
    for each possible value v of f:
      df's value count for v ← virtf + sum of s1's and s2's
                               observed value counts for v
    endfor
    insert df into s12
  endfor
  remove s1 and s2 and add s12 to model m
end
```

Essentially, we just sum the counts on each value for each feature (and also sum the counts of the number of examples incorporated into each sense). It is important to note, though, that it is only *observed* counts that are summed, not virtual counts, since otherwise merged models could never generate more-peaked probability distributions than occur in the initial senses. In the implementation this is accomplished by not actually storing the virtual samples within the data structure for each distribution but rather by referring to the global *virt* table as well as the local observed counts whenever probability calculations must be made.

Top-level control and online learning

We finally present the top-level control of the model merging algorithm:

```
INCORPORATE_EXAMPLE(linking f-struct l, verb v, model m) modifies m
  CREATE_NEW_SENSE(l, m's virtual samples table virt)
  if (BatchSize senses have been created for v since last merge) then
    loop:
      s1, s2 ← FIND_BEST_CANDIDATE_MERGE(m's senses for v)
      if (s1 and s2 are null) then terminate loop
      old posterior ← COMPUTE_MODEL_POSTERIOR(m)
      PERFORM_MERGE(m, s1, s2)
      new posterior ← COMPUTE_MODEL_POSTERIOR(m)
      if (new posterior is lower) then undo merge and terminate loop
    endloop
  endif
end
```

While the general model merging algorithm appears to require the full data set to be present initially, this is not necessary. As can be seen above, our top-level algorithm waits only until $BatchSize$ training examples have accumulated for a given verb before entering the main loop which performs iterative merging. Model merging has historically proven robust to being used in a nearly online fashion, i.e. by using batch sizes of approximately ten (Stolcke 1994).

The main merging loop continually checks the model's posterior probability after each merge, and stops as soon as a merge reduces the posterior. This last merge is then "undone". This stopping criterion is not particular robust, since a single small "dip" in the

posterior might terminate merging, even if many more merges could be performed which would increase the prior. An alternate version of the algorithm considers other potential merges when the best candidate merge fails. In practice, though, it was found that the similarity heuristic is actually quite good, and this extra level of robustness was unnecessary.

6.2.5 Computational complexity

First we consider the worst-case performance of model merging for the offline case, i.e. where all training data is available initially. In this case, the running time is of complexity $O(n^3)$ where n is the number of training examples. The reasoning is straightforward: at most $O(n)$ merges will be performed, and at each merge step there are $O(n^2)$ potential pairs of senses to evaluate as potential merges.

The online case presents the potential for considerable improvement in efficiency, since n above is replaced by the batch size b which is presumably much lower. This is assuming the number of senses in the lexicon (measured after each completion of the merge loop for each batch) converges to a small constant. Intuitively, the algorithm will be faster in the online case because we expect to never accumulate *too* many word senses at any given time, since a moderate amount of merging is expected after every batch of b training examples is processed.

6.2.6 Updating the virtual sample priors

An optional final step has been added to the INCORPORATE_EXAMPLE routine to speed learning of new verbs once a modest vocabulary has already been learned. The key insight here is that each language tends to code for certain features in its verbs. For example, English verbs commonly code for manner (implicating features such as `posture`) while Korean or Spanish verbs usually code for more spatial features such as `direction`. Children pick up on this sort of language pattern quite early (Choi & Bowerman 1991).

These patterns are captured by modifying the *virt* virtual sample table. Such modifications affect the amount of generalization performed for each linking feature when a new verb is encountered and its initial word sense f-struct is created. Decreased generalization is appropriate for those features which have often proven relevant in previously learned verbs. It is achieved by lowering the number of virtual samples for such features, thereby

causing a new verb's initial word sense to stick more closely to the value found in its training example. Conversely, increased generalization is appropriate for features which have previously proven irrelevant. It is achieved by increasing the number of virtual samples—in the limit effectively ignoring the specific feature values found in future training examples for a new verb.

To summarize: as a result of such modifications to *virt* during learning, new words in a slot will more quickly converge on their relevant features, so long as they fit the established pattern.

The heuristic used to adapt the virtual sample count $virt_f$ for a given feature f is as follows. We first define a function which maps probability distributions d_f over f to an "effective" virtual sample count. Intuitively, the broader the distribution, the greater the number of effective virtual samples. The formula used is

$$v_{d_f} = \frac{1 - P_{d_f}(\text{mode})}{n_f P_{d_f}(\text{mode}) - 1} \tag{6.8}$$

where n_f is the number of possible values of the feature f, and "mode" is the most probable value of d_f. Why is this a reasonable formula? The equation can be rewritten as

$$P_{d_f}(\text{mode}) = \frac{v_{d_f} + 1}{n_f v_{d_f} + 1} \tag{6.9}$$

Observe that this latter formula reflects the algorithm used when forming each distribution in an initial word sense for a new verb. The bottom line, in other words, is that Equation 6.8 computes the number of virtual samples which, if used in creating a new distribution, would assign the mode of the new distribution (i.e. the observed value) the same probability which it has in d_f.

We then compute the average of these effective virtual sample counts for feature f across all verbs, weighted by the number of training examples supporting each verb. This average also includes the current value of $virt_f$, weighted by a tunable parameter $VirtualInertia$ (which thus specifies the rate of adaptation of the *virt* table as a whole). The resulting value replaces the current value in $virt_f$.

A difficulty with this heuristic is that for a perfectly uniform probability distribution, the effective virtual sample count is infinite. To prevent setting $virt_f$ to infinity, each

effective virtual sample count is artificially limited to $MaxVirtuals$, yet another tunable algorithm parameter.

This procedure is summarized below as the routine ADAPT_VIRTUALS:

```
ADAPT_VIRTUALS(model m) modifies virt
  for each linking feature f:
    wsum ← 0
    for each word sense s in m:
      wsum ← wsum + (min(1−P(mode) / nP(mode)−1, MaxVirtuals) × s's example-count)
    endfor
    wsum ← wsum + (virt_f × VirtualIntertia)
    wsum ← wsum ÷ (sum of example-counts + VirtualIntertia)
    virt_f ← wsum
  endfor
end
```

The routine is run after each merging loop during training.

6.2.7 Summary of algorithm parameters

At this point we have concluded the presentation of all algorithms for verb use and learning. Since a fairly large number of algorithm parameters have been employed, a summary is in order. The full set of parameters, each with its range of allowed values, is summarized in Figure 6.3. Training results are reported in Chapter 8. The parameter settings for the main English training run in that chapter are shown in Figure 6.3 in the column labelled "Typical".

6.3 Alternatives to Model Merging

We've seen Bayesian model merging and its advantages, but we haven't yet discussed other relevant learning algorithms and why they weren't chosen.

Since ours is a connectionist enterprise, an obvious question is why we did not choose backpropagation (Rumelhart *et al.* 1986)—the *de facto* neural network learning algorithm—or one of its many variants. The primary difficulty lies with the network struc-

Parameter	Range	Typical	Description
$MinLabel$	[0, 1]	10^{-7}	Minimum $P(word \mid action)$ required to emit *word*
$MinExplain$	[0, 1]	0.5	Minimum $P(feature \mid sense)$ required to explain away $feature$
$MinObey$	[0, 1]	10^{-7}	Minimum $P(sense \mid worldstate)$ required to use *sense* in setting linking f-struct
$MinSetFeature$	[1, ∞]	1.5	Minimum "peakedness" of distribution required to set feature in linking f-struct
$MinMerge$	[0, 1]	0	Minimum similarity required between two senses if they are to be merged
$ModelPriorWeight$	[0, ∞]	4.5	Magnitude of preference for merging relative to preserving likelihood
$BatchSize$	1, 2, ...	∞	Number of new instances of a word to accumulate between merging episodes
$TrainingPasses$	1, 2, ...	1	Replicate training data this many times
$AdaptVirtuals$	true, false	true	Whether to adapt Dirichlet prior in each slot during learning
$VirtualInertia$	[0, ∞]	50	Effective number of samples supporting current virtuals when adapting them
$MaxVirtual$	[0, ∞]	10	Prevent uniform distributions from causing infinite virtuals
virt table	[0, ∞]	`schema`: 0.05 all others: 1.0	Number of virtual samples to add to each value of each feature in each word sense

Figure 6.3: Summary of parameters of the labelling, obeying and learning algorithms and their settings for the English training run.

ture which backpropagation presupposes. Hidden-node representations of one-way mappings, such as those produced by backpropagation, are very difficult to reverse (as is needed for command-obeying in our task). The network structure does not include any reverse connections which could represent the reverse mapping. If such connections were added, their weights would have to be learned independently of the forward connections. And since the hidden nodes do not correspond to any previously identified features, it is impossible to include pre-wired network structure to "interpret" them.

Nonetheless, there is one way to attempt to reverse such mappings, called *back-propagation through the inputs* (Thrun 1992). Bailey (1992) attempted to use this technique to model the mapping from a spatial term to a mental image of its prototypical case, in the context of Regier's (1996) system. The technique involves applying a random input vector (i.e. image) to the trained network, computing the corresponding (bogus) output vector, and comparing it to the desired output (in which only the node for the desired spatial term is active). The error propagation formulas are then applied, but rather than treating the inputs as fixed and the weights as variable, we do just the opposite: the weights are treated as fixed, and the input is modified to reduce the error. The process is then repeated with this slightly-improved input vector. Eventually the network will converge on an input vector which minimizes the output error. If this error is low enough, we can conclude that the inputs now represent a good "mental image" of the spatial term. However, the success of this project was limited (the non-continuous features in the structured portion of Regier's network could not be handled) and the cognitive plausibility of such an iterative procedure is rather lacking.

Another reason to avoid backpropagation learning is that the trajectory of learning is typically quite slow, since weights are adjusted in small increments to avoid "hopping" over the correct solution. Indeed, typically backpropagation training involves many repeated presentations of the training set. So backpropagation is not capable of modelling fast mapping.

Our multiple-word-sense representation can be viewed as a mixture model. A common algorithm for learning mixture distributions is the "Expectation-Maximization" or "EM" algorithm (Dempster *et al.* 1977). But the EM algorithm focuses on estimation of parameters and not learning of the structure which gives rise to the hidden mixture parameters in the first place. Furthermore, implicit in EM is the notion that an observation

may be "caused" by any of the mixture elements, so that the probability of an observation is the sum of its probability of being generated by each mixture element. And in adjusting each mixture element, each observation is used (in proportion to its probability given the current parameters of the mixture element). In contrast, in our language task, we assume that cognitively one commits to a single sense when categorizing an observation, and hence we do not want to sum the probabilities across all senses of a word, but rather we should look only at the highest-probability sense.

6.4 Connectionist Account

We now turn to the issue of implementing the model merging algorithm in a connectionist manner, so that we will have a unified connectionist story for the entire system. The learning mechanism operates within the architecture described in §5.4, which should be understood before proceeding. And, just as was the case in §5.4, the ideas here are intended as a plausibility argument and have not been worked out in full detail.

Recall the neural plausibility criteria from §2.4. At first glance, the model merging algorithm of §6.2 does not appear particularly connectionist. Two properties cause trouble. First, the algorithm is constructivist. That is, new pieces of representation (word senses) need to be built, as opposed to merely gradually changing existing structures. Second, the criterion for merging is a global one, rather than depending on local properties of word senses. Nevertheless, we have a proposed connectionist solution employing a learning technique known as recruitment learning.

6.4.1 Recruitment learning

Recruitment learning (Feldman 1982; Shastri 1988) assumes a localist representation of bindings such as the triangle unit described in §5.4, and provides a rapid-weight-change algorithm for forming such "effective circuits" from previously unused connectionist units.

Figure 6.4 illustrates recruitment with an example. Recall that a set of triangle units is usually connected in a winner-take-all (WTA) fashion to ensure that only one binding reaches an activation level sufficiently high to excite its third member. For recruitment

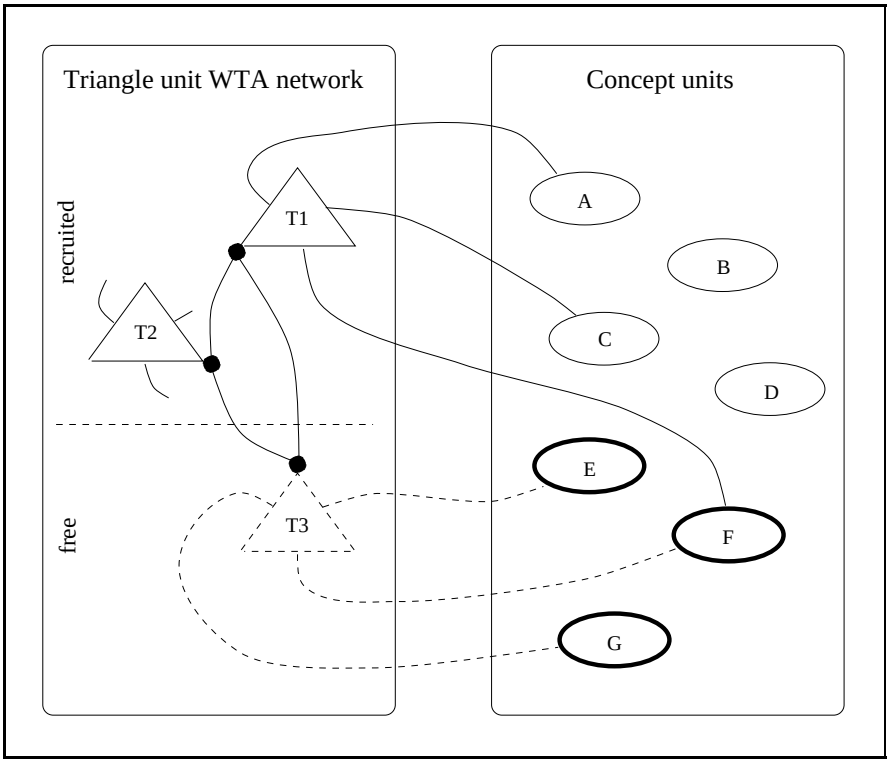


Figure 6.4: Recruitment of triangle unit T3 to represent the binding E–F–G.

learning, we further posit that there is a pool of "free" triangle units which also take part in the WTA competition. The units are free in that they have low, random weights to the various "concept units" amongst which bindings can occur. Crucially, though, they *do* have connections to these concept units. But the low weights prevent these free units from playing an active role in representing existing bindings.

This architecture facilitates the learning of new bindings as follows. Suppose, as in Figure 6.4, several triangle units already represent several bindings, such as T1, which represents the binding of A, C and F. (The bindings for T2 are not shown.) Suppose further that concept units E, F and G are currently active, and the WTA network of triangle units is instructed (e.g. by a chemical mechanism) that this binding must be represented. If there already exists a triangle unit representing the binding, it will be activated by the firing of E, F and G, and that will be that. But if none of the already-recruited triangle units represents the binding, then it becomes possible for one of the free triangle units (e.g.

T3)—whose low, random weights happen to slightly bias it toward this new binding—to become weakly active. The WTA mechanism selects this unit and increases its activation, which then serves as a signal to the unit to rapidly strengthen its connections to the active concept units.[7] It thereby joins the pool of *recruited* triangle units.

As described, the technique seems to require full connectivity and enough unrecruited triangle units for all possible conjunctions. Often, though, the overall architecture of a neural system provides constraints which greatly reduce the number of possible bindings, compared to the number possible if the pool of concept units is considered as an undifferentiated whole. For example, in our connectionist word sense architecture, it is reasonable to assume that the initial neural wiring is predisposed toward binding words to features—not words to words, or feature units to value units of a different feature. The view that the brain starts out with appropriate connectivity between regions on a coarse level is bolstered by the imaging studies of Damasio & Tranel (1993) which show, for example, different localization patterns for motor verbs (nearer the motor areas) *vs.* other kinds of verbs.

Still, the number of potential bindings and connections may be daunting. It turns out, though, that sparse random connection patterns can alleviate this apparent problem (Feldman 1982). The key idea is to use a multi-layered scheme for representing bindings, in which each binding is represented by *paths* amongst the to-be-bound units rather than direct connections. The existence of such paths can be shown to have high probability even in sparse networks, for reasonable problem sizes. These issues are not considered further in this discussion.

Complex triangle units, with multiple connections per side (see §5.4), present some special difficulties for recruitment learning. Consider first the case of the conjunctive style of complex triangle unit, which must not only bind together a set of concept units, but must also partition them into three groups in the desired way. Since all the concept units are equally active at the time of recruitment, there is no way for a potential recruit to "know" if it is wired for the desired partitioning. In this case, we must rely on the overall structure of the net to restrict the potential partitionings to those which are reasonable. For example, in our connectionist word sense architecture, this is easy to specify because each side of the

[7] This kind of rapid and permanent weight change, often called *long term potentiation* or LTP, has been documented in the nervous system. It is a characteristic of the NMDA receptor, but may not be exclusive to it. It is hypothesized to be implicated in memory formation. See Lynch & Granger (1992) for details on the neurobiology, or Shastri (1997) for a more detailed connectionist model of LTP in memory formation.

conjunctive triangle units used to represent word senses has a quite different function.

Next, consider the case of the disjunctive style of complex triangle unit. The difficulty here is in how the variable weights are set, at the neural level. The incoming weights are not problematic; we can employ an update rule which sets each weight to be proportional to the degree of activation of its input. The hitch is the outgoing weights, for they are stored in—and hence must be updated by—the "concept" neurons, not the triangle unit's own neuron. We must, therefore, posit a non-local mechanism (e.g. chemical) for informing all concept neurons that learning is taking place, and that if any of them should receive input from a (newly recruited) triangle neuron, they should set their weight on that synapse in proportion to *their own* (not the triangle unit's) degree of activation.

6.4.2 Merging via recruitment

The techniques of recruitment learning can be put to use to create the word sense circuitry shown earlier in Figure 5.8. The connectionist learning procedure does not exactly mimic the algorithm given above (more on that later), but is most similar to the online case ($BatchSize = 1$) illustrated back in §6.2.1.

To illustrate our connectionist learning procedure, we will assume that the two senses of *push* shown in Figure 5.8 have already been learned, and a new training example has just occurred. That is, the "push" unit has just become active, as have some of the feature value units reflecting the just-executed action.

The first key observation is that when a training example occurs, external activation arrives at a verb unit, motor-parameter feature value units, *and* world-state feature value units. This three-way input is the local cue to the various triangle units that adaptation should occur—labelling and obeying never produce such three-way external input to the triangle units. Depending on the circumstances, there are three possible courses of action the net may take:

- **Case 1: The training example's features closely match those of an existing word sense.** This case is detected by activation of the primary triangle unit of the matching sense—strong enough activation to dominate the winner-take-all competition.

  In this case, an abbreviated version of merging occurs. Rather than create a full-

fledged initial word sense for the new example, only to merge it into the winning sense, the network simply "tweaks" the winning sense to accommodate the current example's features. Conveniently, the winning sense's primary triangle unit can detect this situation using locally available information, namely: (1) it is highly active; and (2) it is receiving activation on all three sides. The tweaking itself is a version of Hebb's Rule (Hebb 1949): the weights on connections to *active* value units are incrementally strengthened. With an appropriate weight update rule, this strategy can mimic the probability distributions learned by the model merging algorithm.

- **Case 2: The training example's features do not closely match any existing sense.** This case is detected by failure of the winner take all mechanism to elevate any word sense above a threshold level.

  In this case, standard recruitment learning is employed. Pools of unrecruited triangle units are assumed to exist, pre-wired to function as either primary or subsidiary units in future word senses. After the winner-take-all process fails to produce a winner from the previously-recruited set of triangle units, recruitment of a single new primary triangle unit and a set of new subsidiary units occurs. The choice will depend on the connectivity and initial weights of the subsidiary units to the feature value units, but will also depend on the connections amongst the new units which are needed for the new sense to cohere. Once chosen, these units' weights are quickly set to reflect the *currently active linking feature values*, thereby forming a new word sense which essentially is a copy of the training example.

- **Case 3: The training example's features are a moderate match to two (or more) existing word senses.** This case is detected by a protracted struggle between the two partially active senses which cannot be resolved by the winner-take-all mechanism. Figure 6.5 depicts this case. As indicated by the darkened ovals, the training example is labelled "push" but involved `medium force` applied to a `small size` object—a combination which doesn't quite match either existing sense.

  This case triggers recruitment of triangle units to form a new sense as described for case 2, but with an interesting twist. The difference is that the weights of the new subsidiary triangle units will reflect not only the linking features of the current training example, *but also the distribution of values represented in the partially active senses.*

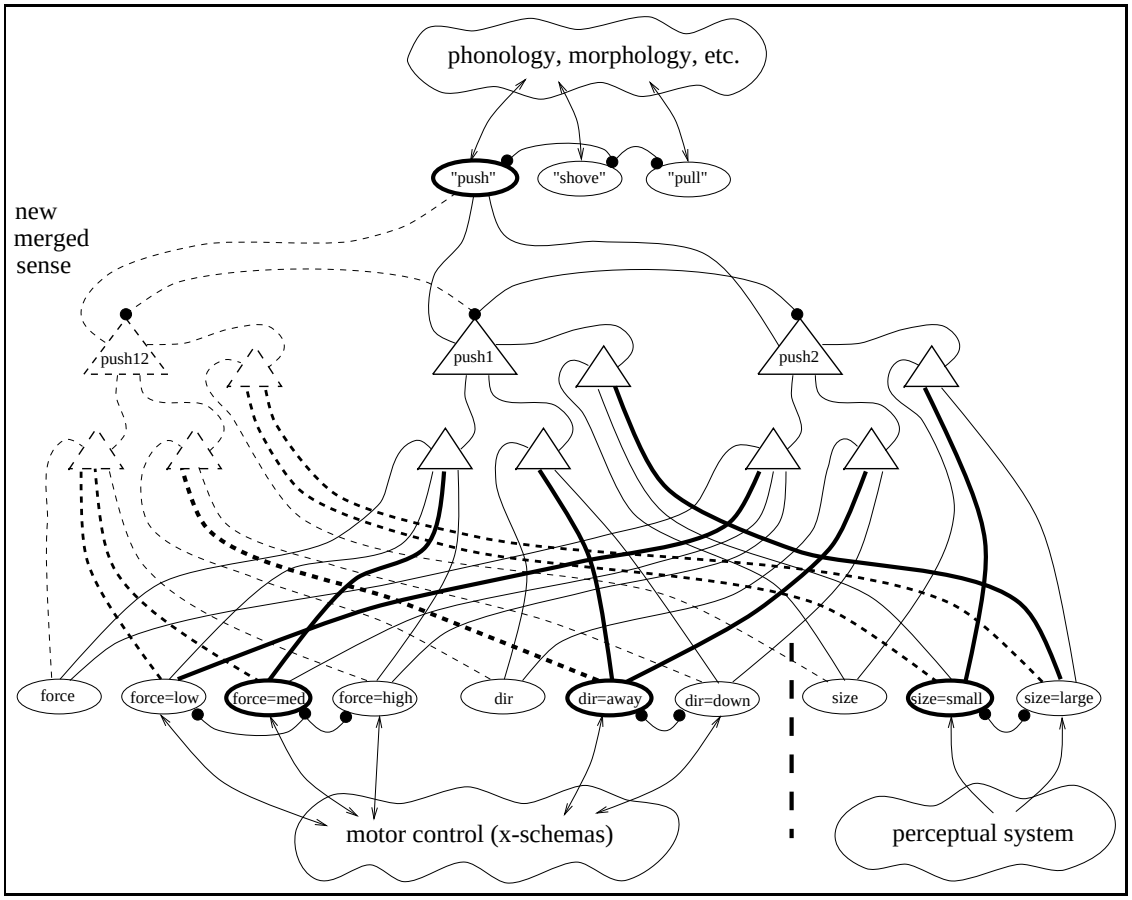


Figure 6.5: Connectionist merging of two word senses via recruitment of a new triangle unit circuit.

Thus, the newly recruited sense will be a true merge of the two existing senses (as well as the new training example). Figure 6.5 illustrates this outcome by the varying thicknesses on the connections to the value units. If you inspect these closely you will see that the new sense "push12" encodes broader correlations with the `force` and `size` features than those of the previous senses "push1" and "push2". In other words, "push12" basically codes for `dir = away, force not high`.

How can this transfer of information be accomplished, since there are no connections from the partially active senses to the newly recruited sense? The trick is to use indirect activation via the feature value units. The partially active senses, due to their partial activation, will deliver some activation to the value units—in proportion to their outgoing weights. Each value unit adds any such input from the various senses which connect to it. Consequently, each feature subnetwork will exhibit a distributed activation pattern reflecting an average of the distributions in the two partially active senses (plus extra activation for the value associated with the current action). This distribution will then be effectively copied into the weights in the newly recruited triangle units, using the usual weight update rule for those units.

A final detail for case 3: to properly implement merging, the two original senses must be removed from the network and returned to the pool of unrecruited units. If they were not removed, the network would quickly accumulate an implausible number of word senses. After all, part of the purpose of merging is to produce a compact model of each verb's semantics. But there is another reason to remove the original senses. The new sense will typically be more general than its predecessors. If the original senses were kept, they would tend to "block" the new sense by virtue of their greater specificity (i.e. more peaked distributions). The new sense would rarely get a chance to become active, and its weights would weaken until it slipped back into unrecruited status. So to force the model to use the new generalization, the original senses must be removed. Fortunately, the cue for removal is available locally to these senses' triangle units: the protracted period of partial activation, so useful for synthesizing the new sense, can serve double duty as a signal to these triangle units to greatly weaken their own weights, thus returning them to the unrecruited pool.

Again, the foregoing description is only a sketch, and activation functions have not been fully worked out. It is possible, for example, that the threshold distinguishing case 2

from case 3 could prove too delicate to set reliably for different languages. These issues are left for future work.

Nonetheless, several consequences of this particular connectionist realization of a model-merging-like algorithm are apparent. First, the strategy requires *presentation of an intermediate example* to trigger merging of two existing senses. The architecture does not suddenly "notice" that two existing senses are similar and merge them.[8] It is conceivable that one could test for the presence of such a strategy in children. In other words, it might be instructive to try to test whether leaps in generalization ability tend to occur shortly after presentation of intermediate usages of a word, as opposed to occurring after a "contemplative" phase, or in response to repeated occurrence of the original usages of the word.

Another consequence of the architecture is that it never performs a series of merges as a "batch". There is no "merge loop" as described in §6.2.4. On the other hand, the architecture does, in principle, allow each merge operation to combine more than two existing senses at a time. Indeed, technically speaking, the example illustrated in Figure 6.5 is a three-way merge of "push1," "push2" and the current training example. The question of the relative merits of these two strategies is left unexplored.

In summary, we have shown that the two seemingly connectionist-unfriendly aspects of model merging—its constructiveness and its use of a global optimization criterion—can be overcome by using recruitment learning and a modified winner-take-all mechanism.

6.5 Overgeneralization and Contrast Sets

Now that we have presented the full learning algorithm, we return to a few psycholinguistic issues and how they are reflected in the model.

A common pattern in early lexical development is *overgeneralization*, in which word meanings initially extend beyond the bounds of the adult meaning. The reader may object that our merging algorithm is a poor psychological model because it handles presentation of an initial example of a new word by creating a rather specific sense strongly reflecting the training example. The objection has some legitimacy for the case of comprehension

[8] It is possible that an "imagination mode" could be implemented, in which the current senses would be used to generate imagined linking f-structs which then might trigger merging.

(i.e. obeying commands), although it should be pointed out that early *undergeneralization* patterns have also been observed (Kay & Anglin 1982) and are harder to detect. It turns out, however, that our model *can* capture early overgeneralization for the case of production (i.e. labelling actions), which is indeed where they are most prevalent. LABEL can emit a verb (the best-matching one, of course) even if its probability is arbitrarily low, so long as the $MinLabel$ parameter is set to a low value. Thus if the vocabulary—and hence the contrast set (Clark 1987) for the new verb—is small, the verb may indeed exhibit plausible overgeneralization in production, assuming the $MinLabel$ parameter is tuned appropriately.

Extension

This behavior suggests that it may be possible to extend our learning algorithm to take advantage of the current lexicon in learning new verbs. For example, suppose the model has successfully learned *push* along with contrasting verbs such as *pull* or *lift*. When an example of a new verb, *shove*, is presented to the model, the algorithm might:

1. Run the LABEL routine to determine that *push* is the closest existing verb and the match is pretty good;
2. Inspect the features of *push* and compare them to the new training example;
3. Find the biggest differences between them, probably the `acceleration` feature in this case;
4. Conclude that *shove* must code for those differences by the principle of contrast (Clark 1987).

In general, then, the reversibility of the features-to-verbs mapping opens the possibility of discovering relationships between verbs (e.g. generalization, specialization, opposition, etc.). Such information could then be leveraged to provide more specific negative evidence than was possible in Regier's framework, in which each label was taken as mild negative evidence for all other labels.

Chapter 7

Adding Verb Satellites

So far we've considered verbs in isolation. In this chapter, we provisionally extend our model to handle a wider range of linguistic forms. "Verb complexes" consist of a verb root plus any associated affixes, auxiliaries or particles (collectively called *satellites* by Talmy (1985)) which together specify the action, such as the four-component verb complex *keep pick-ing up*. These are handled by adding multiple "slots" to the model.[1]

7.1 The Nature of the Problem

First off, note that we *could* simply treat an entire verb complex as a single "word" and thereby continue to use our algorithms from the preceding chapters. But in doing so, we would forfeit all the benefits which can follow from exploiting the compositional structure of verb complexes. Those benefits include, first of all, compactness. In a compositional account the lexicon will be much smaller. A language with n verbs and m particles will

[1] Full verb phrases are not considered since our focus is on linguistic elements which are fundamentally about action. Noun phrases, prepositional phrases, etc. certainly play a role in specifying actions, but this is beyond the scope of this thesis.

have only $n + m$ lexical items rather than a potential $n \times m$ lexical items. Moreover, a consequence of a smaller lexicon is that the amount of data per lexical item will be much greater, allowing statistical techniques to home in better on the meaning of each. Perhaps most importantly, learning separate semantics for each components permits the model to both produce and obey novel combinations of words.

Children certainly are inclined toward this "take it apart and look inside" approach. In fact, one of the first morphological rules mastered by children—at around age two—is the *-ing* ending (Brown 1973), whose semantics are obviously related to action and process.

So the big question is how to dissect a compositional meaning into its parts during learning, and then how to compose meanings from their parts during language use. If compositionality were just a matter of forming a union of features, the problem would be trivial. But many other types of composition exist, in which one component overrides a feature from another, or modifies a feature in some non-trivial way, etc. Fauconnier (1985) provides numerous examples and proposes the term "blending" to better reflect the subtlety of the phenomenon. Weber (1994) developed a connectionist account of such blends as they occur in adjective-noun phrases. Wu (1992) worked on similar issues for noun compounds. It is far from obvious how to capture the full range of action description blends in the current model. Yet it turns out that some easy cases can be handled by fairly simple modifications to our architecture, to which we now turn.

7.2 Slots: A Provisional Solution

Our partial solution can be summed up quite simply: we divide both linguistic input/output, and our lexicon model, into "slots" corresponding to each of the grammatical positions (prefix, root, particle, etc.) which we want to deal with. The term "slots" is chosen to convey the idea that the model initially has no notion of the *particular* grammatical role played by any slot. The partitioning of an utterance into its various slots represents grammatical knowledge which we are not modelling; this process is done by hand.

Figure 7.1 shows a revised version of our model with two slots: Slot 1 is for verb roots, and Slot 2 is for particles and other directional specifiers. It's important to note that there is no structural distinction between the slots. And since our multiple-sense word

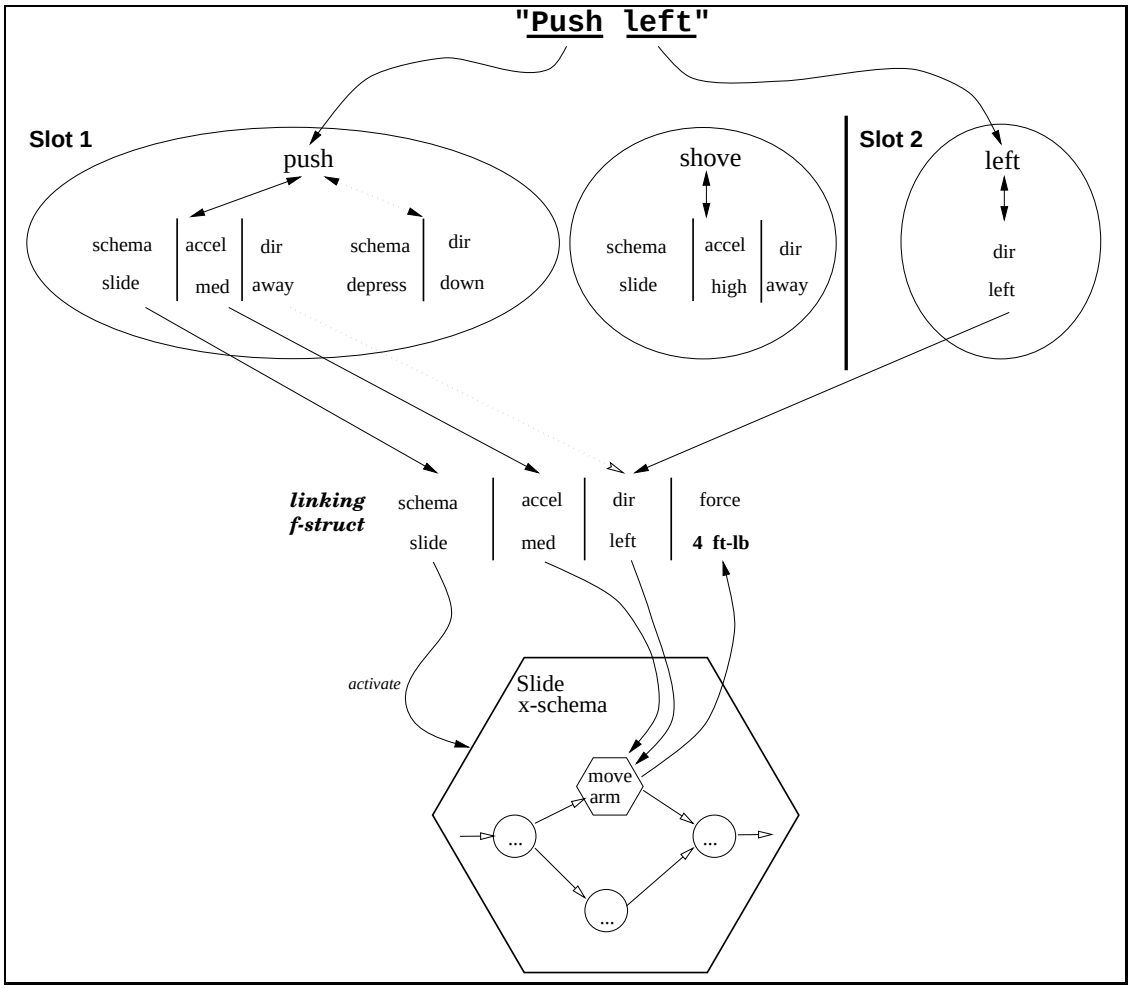


Figure 7.1: Word senses from each of two slots are chosen and then combined to process the command *push left*.

representation does not involve any structure "connecting" the word senses, there is no apparent boundary between the slots when looking at the lexicon. The boundaries become relevant in two places: first, an utterance is chopped up and each word (or morpheme) is delivered to its appropriate slot. Again, we don't model the details of this process and the grammatical knowledge it would require. But the boundary is also relevant at the interface between the lexicon and the linking f-struct, and this will be the focus of this chapter.

We will now look at how the labelling, obeying and learning algorithms have been adapted to deal properly with slots. Except as noted, these algorithms have been implemented.

7.2.1 Labelling and obeying

Labelling proceeds much as described earlier, except that it is repeated for each slot. Here is the algorithm:

```
LABEL_MULTISLOT(linking f-struct l, model m) returns a verb complex
  create an empty verb complex vc
  loop:
    for each unfilled slot i in vc:
      best_label_i ← LABEL(m_i, l)
    endfor
    if (at least one non-empty label was collected)
      then choose best_label_i with the highest probability and place it in slot i of vc,
                  as long as its probability ≥ MinLabel
           remove all features f from l with P(l_f | best label_i) ≥ MinExplain
      else terminate loop
    endif
  endloop
  return vc
end
```

In §5.3.1, our labelling algorithm was essentially "forced choice" in that the highest *a posteriori* label was always emitted (unless its posterior probability was less than $MinLabel$). We cannot, however, be quite so eager to fill slots in a multi-slot label, lest we produce potentially redundant verb complexes such as *shove hard* or *drop down*. We also want to prefer short phrases such as *pull* to longer phrases such as *slide toward*. These problems

have been addressed previously in the natural language generation literature; here, we capitalize on our use of probability to cleanly describe these preferences. Our algorithm works by filling slots one by one, and after each slot is filled those features "explained" by the filler are removed from the linking f-struct. The criterion for explaining away a feature is that its probability given the label exceeds a tunable parameter, $MinExplain$. With this procedure, subsequent slots will be filled only if there remain features which have not been expressed. With this scheme it is important that we fill slots not in an arbitrary order, but rather by choosing at each step a label with maximal *a posteriori* probability, so that the features of the linking f-struct will be expressed with the smallest number of words. Accordingly the $MinLabel$ parameter plays an even more important role than before, since it is now responsible not only for saying "I can't find a good label at all," but also is responsible for deciding when enough slots have been filled to adequately express the content of the action.

The algorithm for obeying a command by choosing suitable linking features must also change to handle verb complexes. Here is the changed algorithm:

```
OBEY_MULTISLOT(verb complex vc, initial world state w, model m)
returns motor-parameter linking features
  for each slot i of m:
    for each sense s of vc_i:
      prior ← relative frequency of s amongst senses of v
      likelihood ← product over each world-state feature f in s of:
                     s_f's likelihood of generating w_f
      posterior ← prior × likelihood
    endfor
    if (some sense's posterior ≥ MinObey)
      then mark the sense with highest posterior
    endif
  endfor
  create an empty f-struct p
  for each motor-parameter feature f:
    find the marked sense s with maximal peakedness for feature f
    if (s_f's peakedness ≥ MinSetFeature)
      then p_f ← s_f's mode value
    endif
  endfor
  return p
end
```

The essential problem is one of arbitrating conflicts amongst the multiple words (or morphemes) in the various slots. We have already seen how potential conflicts with the initial world state are considered in choosing a "best" sense for a verbal command. In the multi-slot case this procedure is performed in parallel in each slot, resulting in a *set* of word senses. Now, since there is potential for conflict amongst the senses, a mechanism is required to resolve these conflicts on a feature-by-feature basis in order to reduce them to a single (non-probabilistic) linking f-struct.

This is accomplished as follows. Each linking feature is set by looking to the sense with the most peaked distribution and choosing its peak value. This is only an approximation to the more probabilistically correct approach of summing each sense's distribution for the feature and then picking the mode of the summed distribution. The intuition here is that the word sense which most strongly codes for the feature should have the privilege of setting that feature, since that may well be the reason the word was included in the command. In the example in Figure 7.1, we assume that *push* codes for `direction = away`

but only weakly, whereas *left* codes for `direction = left` quite strongly. As a result, the `direction` linking feature is set to `left`.

7.2.2 Learning

No extensions were required to handle verb complexes (other than the simple step of noting the correct slot for each new lexical item). The lexicon is simply that much larger to account for the new words. And each training example involves multiple calls to INCORPORATE_EXAMPLE, one for each component of the verb complex.

While changes may not be required, one straightforward change has been made which leverages the knowledge of words' slot position to speed up learning. In §6.2.6 we described ADAPT_VIRTUALS, an optional step modifying the virtual sample table *virt*. In the multi-slot case, we modify the algorithm to maintain one such table for each slot, in an array. Each is updated by ADAPT_VIRTUALS to reflect only those lexical items in its own slot. In this way, the algorithm can learn the type of semantic information which tends to be encoded in each slot. For example, after exposure to an English training set we expect $virt[root]_{schema} \ll virt[particle]_{schema}$ to reflect the fact that the `schema` feature is usually coded by the verb root, not the particle. On the other hand, $virt[root]_{direction} \gg virt[particle]_{direction}$ since `direction` is more commonly encoded in the particle than the root.

Extension

In order for a word to be learned correctly in the multi-slot case, the learning algorithm must *marginalize* over many other words with which it may be used. Otherwise, it will be impossible to pick out the word's semantics from the semantics of those other words. For example, if *up* were observed only in conjunction with *push*, there would be no way to know that it encodes only a direction and not some of the other motor parameters which rightly belong to *push*. Only by observing *lift up*, *hold up*, *pull up*, etc. can the algorithm home in on direction as the relevant feature.

The training set, then, must reflect this diversity of usage. Siskind (1995) discusses the characteristics of so-called *cross-correlational learning* in detail. His conclusion is that such techniques are feasible even given the limited amount of input a child receives. His

algorithm differs from ours in that it maintains two lists for each word—one for required features and one for prohibited features[2]—and this facilitates elimination of possibilities even before the exact meanings of the words in an utterance are known.

A similar technique, unimplemented at the current time, could be applied to our algorithm. Consider the case of training examples where the verb complex contains an already well-understood word (as measured, say, by the peakedness of its probability distributions) along with a less known (or totally new) word. In this case those features explained by the known word can be removed from the linking f-struct, on the assumption that the multi-word label does not redundantly code for those features. Thus, the unknown word would gain an advantage in determining its relevant features.

7.3 Limitations of the Model

The above algorithms can handle some cases of compositional meaning, and they do so without our having to introduce any notion of grammar into the system. Naturally, though, there are many examples of composition which cannot be handled by such a simple model.

One limitation derives from our use of distinct slots as the sole way to represent what is going on grammatically in an utterance. Such a representation cannot easily handle inflections which *modify* the root (as opposed to appending to it) without manual "hacking" of the linguistic input so that the system will be able to recognize the root as familiar. In general, a fuller account of compositional meaning will require a serious model of syntax and morphology.

While OBEY_MULTISLOT does a good job of conflict resolution, it does only a fair job of conflict avoidance in choosing a sense for each word in the command. Polysemy involves not only the world state, but also the surrounding linguistic context. Some experiments with polysemy were performed in Regier's visual/spatial model by Zlatev (1992) (summarized in (Regier 1996:Section 7.1)) on recognizing spatial phrases consisting of a verb and a preposition (e.g. *be over vs. fly over*). Hierarchical cluster analysis of the resulting hidden layers revealed polysemous representations in which each sense of a word

[2]In this sense the algorithm is reminiscent of version spaces (Mitchell 1982) although it adds a mechanism to handle noise.

corresponded to a different linguistic context. That is, there would be a sense of *over* corresponding to its use with *be* (the static sense of *over*), and another sense corresponding to its use with *fly* (one of the dynamic senses of *over*). This suggests that linguistic context plays an important role in determining the intended sense of a polysemous word. Our model certainly allows correlations between senses of a word and the surrounding linguistic context, but it turns out that it also requires that there be some correlated features of the initial world state. This is the case because the OBEY_MULTISLOT algorithm chooses the best sense for each word based only on each sense's compatibility with the initial world state. To account for linguistic context effects, that algorithm would need to be extended to minimize conflicts *amongst* word senses, not just between each word and the world state. This is a general constraint satisfaction problem and one can imagine a number of solutions (including recasting the word sense representation into the belief network formalism (Pearl 1988)). Of course, any such algorithm will be iterative in nature and hence slower than the current algorithm.[3]

A common problem is found in verb complexes where a modifier alters, but does not itself specify, the appropriate value of a feature. Here are some thoughts on how to handle such cases. We would need to build at least a small amount of grammatical knowledge into the system: namely, we would need to distinguish the verb root slot, and classify the other slots (adverbs for instance) as "modifier slots". We would also need to allow a new kind of value for features: namely, modification rules. For example, the adverb *hard* could code for `force = "multiply root's value by 2"`.

It's worth pointing out, though, that even with these extensions we are still a long way off from handling all cases of action specification. An interesting case in point is the verb *stop* when used as an auxiliary verb (Crangle & Suppes 1994: Chapter 7). In the simplest case, the meaning is to abort an in-progress action (e.g. *stop pushing*), which can be straightforwardly modelled in our x-schema formalism. Even here, there is the difficulty of deciding whether to abort an in-progress primitive synergy (e.g. *stop pushing*), or whether to stop at the x-schema level (e.g. *stop picking up cubes*, which perhaps should be obeyed by finishing picking up the current cube but then terminating a loop before picking up another one). Harder still, though, is a command such as *stop holding* which in fact requires the

[3] Another approach is to encode linguistic context directly into the semantic representation for each word, such as in Charniak (1993). In other words, we might add a `used-with` feature whose values range over the lexical items in adjacent slots (a so-called n-gram approach).

initiation of a new action (i.e. to put the object down). Narayanan (1996) and Jonathan Segal have done work in this vein.

7.4 Thoughts on Construction Grammar and Learning

All these limitations conspire to suggest the need to account for how grammatical constructions might be learned. The Construction Grammar framework (Goldberg 1995) has been chosen because it is compatible with the assumptions made in this model and might even support learning using techniques similar to those used here for lexical learning.

Verb acquisition has long been associated with grammar acquisition. Indeed, some might find it strange for this thesis to have considered verb semantics independently of argument structure. Controversy has continued over the direction of causality in the learning of semantics and syntax (the so-called "syntactic *vs.* semantic bootstrapping" debate between Pinker (1989) and Gleitman (1990)). I believe that a Construction Grammar approach can solve this dilemma.

Construction Grammar replaces the traditional lexicon-grammar dichotomy with a continuum, ranging from the specific, such as lexemes and idioms, to the general, such as S $\rightarrow$ NP VP. In between there is scope for constructions which are somewhat rule-like but also have specific content—in both their surface forms and their semantics. For example, the "Way Construction" (*She made her way through the crowd*) includes syntactic elements like verbs and noun phrases but also a particular lexeme (*way*); similarly it involves semantic composition of its components but also some additional content particular to the construction (progress, effort, repetition). Importantly, Goldberg (1995) argues persuasively that much of the semantics which have typically been associated with verbs (namely for dealing with arguments) are more properly considered part of the semantics of the constructions in which the arguments are couched. Our study of the inherent semantics of verbs—apart from their arguments—meshes nicely with this perspective.

This perspective with its continuum of abstractness is also compatible with learning theory. Furthermore, we have reason to believe our simple hand action framework is an appropriate one for study of early grammar learning: Slobin (1985) argues that the earliest grammatical markings and rules acquired by children apply preferentially to "prototypical

events" such as a person manipulating a physical object.[4] Such an approach, in which syntactic generality is achieved only gradually, is driven by semantics, and is focused around verbs, is taken seriously in the the psychology literature and is sometimes referred to as the "Verb Island Hypothesis" (Tomasello 1991). Indeed, in recent work, Tomasello has begun to describe verb islands simply as early constructions (Tomasello & Brooks (to appear)).

How would such an approach work? If only we knew the set of constructions in advance (syntactically, that is), then we could apply the same learning algorithms used thus far to learn a sense or set of senses for each construction. But of course we don't have this advance information, and while we are busy searching for repeated syntactic patterns in the training sentences, we must be careful not to lose the associated semantic information. The model merging approach may offer a solution, and an initial attempt has been made by Stolcke (1994: Chapter 5). We begin by creating a construction for every training sentence. Then we try to find both semantic and syntactic regularities, eventually leading to a mixture of lexical entries, general syntactic constructions and intermediate constructions (such as the Ditransitive or the Way Construction) with their syntactic and semantic selectional constraints.

A difficulty of this approach is that, since the training data takes the form of pairings of entire sentences with their semantics, it takes a long time and a lot of training data before semantic information can be properly ascribed to the individual lexical items. Fortunately, the learning task faced by the child may not be quite so severe. Parents speak a simplified language to their children, including many examples of one-word sentences. The value of ordering a training set so it begins with simpler examples is well known (Elman 1991). If the system sees enough such examples early on it may develop accurate semantic representations of some lexemes and then be able to leverage this information in learning higher-level constructions.

The result of such learning would be a construction grammar augmented by probabilities reflecting the training data. Such a grammar would be well suited to probabilistic parsing, which, as demonstrated by Jurafsky (1996), offers promise as a model of human performance in language understanding.

[4] For more recent discussion of the semantics of grammatical forms see Slobin (to appear), which argues that rather than seeking an innate set of grammaticizable notions, we should think of grammaticizability as a continuum which, combined with linguistic experience, aids the child in determing the semantics of grammatical forms in his language.

Chapter 8

Learning Results

"An ounce of action is worth a ton of theory."
—Friedrich Engels

In this chapter, we turn to the validation of our model on selected verbs from English as well as other languages.

8.1 Training Procedure

The first step in our experimental procedure is the generation of training and test sets. Each element of these sets is called a *scenario* and describes a single action event. Ultimately, a scenario consists of two linking f-structs: an initial one describing the world

state before the action takes place, and a final one which includes the initial world state but also summarizes the action.

Scenarios may be generated in two ways. For reasons of practicality, the majority of scenarios have their initial and final linking f-structs generated directly by a program which randomly assigns values to each linking feature.[1] This Generator program is tuned to yield a set of scenarios which covers a range of actions exhibiting the linguistic distinctions we want to learn, and does so with appropriate relative frequencies. (In particular, the Generator produces more SLIDE executions than any other x-schema, since the thesis has focused on these actions.) The Generator also uses a filter to remove nonsensical scenarios.

Once a collection of scenarios has been generated, the next task is to label them for the various languages of interest. All languages share the same collection of scenarios; labels are stored separately for each language.[2] Labelling is based on the final linking f-struct. An intimate knowledge of the x-schema set and its interaction with the linking features is required in order to determine the action which would have generated the features; generally this has been performed by the author or other researchers familiar with the model. In some cases the labels, too, have been provided by the author, based on interviews with language informants. In other cases, the author has "acted out" the indicated action for labelling by the informant.

The scenario collection can be partitioned arbitrarily into training set, recognition (i.e. labelling) test set, and command-obeying test set. These partitionings are stored separately from the scenarios themselves, to facilitate switching amongst different partitionings of the same set of scenarios. By changing partitionings, and in particular the relative sizes of the sets, one can get a feel for the robustness of results.

The use of training scenarios is straightforward: for each scenario, the final linking f-struct and its label are given to the model merging algorithm as a training instance, for conversion to a word sense and potential merging with existing senses.

There are two ways to test the trained system: we can test recognition ability (labelling novel actions) and we can also test command-obeying ability (given a label, produce

[1] The other method of generating scenarios, still under development, is to use an animation package. This is discussed in the next sub-section.

[2] Most training is done with single word labels corresponding to verb roots. When multi-slot training is done, these multi-slot labels are stored separately from the single-slot labels—acting, in effect, as a different language.

an appropriate action). The use of recognition test scenarios is conventional: a scenario's final linking f-struct is given to the LABEL algorithm, and the result is compared to the stored label. The use of scenarios for testing the obeying of commands is more complex. The stored label is used as the command. It and the initial world-state f-struct are given to the OBEY algorithm, which produces a full linking f-struct. This f-struct is in turn given to LABEL, whose output can be compared to the original command.[3]

For testing of both recognition and obeying, there is a problem with this approach of comparison to the informant's label. Some actions may be good examples of multiple verbs. For example, a *push* is also a good example of a *move*. In this case, LABEL may emit a reasonable label which happens to differ from the stored label but is not truly an error. We do not have any quantitative way to discount this type of error. Instead we simply inspect the posterior probabilities of all word senses and come to a qualitative judgment of the "desired" label's probability in terms of (1) how close it is to that of the emitted label, and (2) how strongly it stands out from the probabilities of inappropriate labels.

8.1.1 Animating actions

As mentioned in §2.3, the preferred method for collecting labels and comand-obeying judgments from informants would be to animate x-schema executions graphically. The *Jack* animation package (Badler *et al.* 1993), developed at the University of Pennsylvania and now sold by Transom Technologies, provides the tantalizing possibility of realistic depiction of x-schema executions. Unfortunately, due to implementation difficulties and shortness of time, *Jack* has not yet been integrated into the verb learning system. However, work is still in progress on this front, and so it is worth reporting how *Jack* is envisioned to connect to the rest of the implemented model.

Jack Overview

By design, *Jack*'s main purpose is in evaluating the ergonomics of working environments such as cockpits or control panels. A human model can be placed into various configurations, and then various kinematic and dynamic properties of the posture can be

[3] A better technique would be to execute the indicated x-schema to produce additional feature bindings before passing the linking f-struct to LABEL.

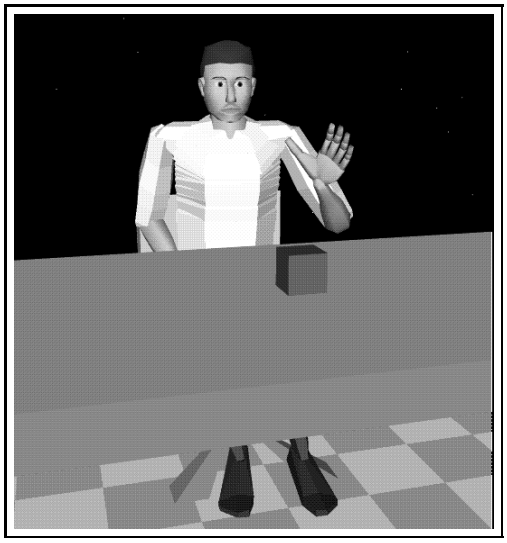


Figure 8.1: A typical Jack stillshot from a hand action animation.

---

evaluated, as well as other concerns such as visibility as the head moves or turns.

Jack is realistic in two ways. First, it models the biomechanics of the human body in intricate detail, including a fully jointed model, with appropriate joint limits and joint strengths. See Figure 8.1 for a sample screen shot depicting *Jack* about to push a cube. Second, the package contains a powerful inverse-kinematics engine which propagates constraints through all these joints so that, for example, a higher-force push will cause the shoulder to "lean in," and lifting a heavy object will cause the arms to pull in toward the body.

While aimed at this type of static analysis, *Jack* fortunately also provides the building blocks needed to animate motions. The primitive commands available match well to the primitive synergies we have posited in the model. This includes arm motions (such as moving the wrist to a coordinate position, or maintaining the hand orientation) and a host of hand postures roughly following the taxonomy in Cutkosky & Howe (1990) (such as power, precision, hook, etc.). It's noteworthy that this confluence arises from modelling biomechanics alone—*Jack* is not at all concerned with modelling neural motor control. Other conveniences include object-oriented access to the world state, so that, for example, one can easily specify that a particular object should be grasped by a particular grasp attachment point.

Jack's Role in Verb Learning

Earlier in this section we described how scenarios are generated randomly by a Generator program. With *Jack* connected to the system, there would be an alternative way to specify scenarios. These "grounded scenarios" would be initially specified by (1) a *Jack* environment file which sets up an initial world state; and (2) a minimal motor-parameter linking f-struct needed to initiate an action. The *Jack* environments would involve an actor seated at a table, and the objects would perhaps be limited to a fist-sized cube, a screwdriver-sized cylinder and a push-button. The action-initiating f-structs would be designed based on intuition of what goals and parameters are needed to generate the desired range of actions.

These grounded scenario specifications would then need to be fleshed out as follows. The *Jack* animator would load the initial world state. Then custom-written perceptual routines would extract the initial world-state linking features needed by our model. A *Jack* LISP implementation of the x-schema set would then issue simulator commands to carry out the action, with two important effects: the final linking features would be collected and stored, and the animation itself would be stored as a fairly realistic movie.

Then, during the labelling process, these grounded scenarios would cause the stored movie to be played, and the final linking features to be recalled. The informant would provide a label for the movie which would then be associated with the linking features.

During the command-obeying phase, grounded scenarios *can* be evaluated the same way as ungrounded scenarios (i.e. by applying the LABEL algorithm to the generated features). However, an additional option is available: the linking f-struct can be used to animate a novel action for visual inspection by the language informant.

8.2 Results for English

8.2.1 Training run

Approximately 200 scenarios were created by the Generator program and labelled with English verbs by the author. The range of actions led to the following vocabulary:

push	slide	slap	lift	turn
pull	press	hit	pick up	roll
shove	touch	tap	heave	
yank	feel	poke	hold	

These verbs occur in the data with different frequencies, ranging from *push* (47) and *pull* (18) to *hit* (5) and *poke* (4). This reflects our concentration on the SLIDE x-schema and results for these verbs should thus be taken more seriously than some of the less frequent verbs.

The most successful training runs use approximately 85% of the data as training examples, and the remaining 15% to test both recognition and command-obeying.

Some experimentation with parameters was required to get good results. The parameter settings for the training run we report on below are shown in the column labelled "Typical" back in Figure 6.3. Note, in particular, the low values for $MinLabel$ and $MinObey$. In fact, they both can safely be set to 0 for single-slot training; since all test scenarios include a label—and one which presumably is consistent with the scenario's initial world state—the system can never improve its performance by refusing to guess a label or carry out a command. The $MinSetFeature$ parameter, however, works best when set to approximately 1.5, which does prune a fair number of features from the linking f-structs used to obey commands. During learning, we chose not to use the $MinMerge$ threshold (by setting it to 0), instead trying all potential merges until one fails to increase the posterior. The strength of the prior is made relatively high by setting $ModelPriorWeight$ to 4.5. Offline learning is specified by setting $BatchSize$ to ∞. Adaptation of the virtual sample table is enabled, but the rate of adaptation is kept low by setting the $VirtualInertia$ to the relatively high value of 50. The initial virtual sample table includes a very low value for the `schema` feature (0.05) and a more moderate value (1.0) for all other features.

Recall the steps in the learning procedure. First the algorithm creates new senses for each training example. Each new sense includes virtual samples. Incorporating a new sense of *pull* is reported in the training log like this:

```
*  Incorporating scenario sc15:
   Creating new sense pull3 (1 ex) {
      size {small=0.333 LARGE=0.666}
      elongated {TRUE=0.666 false=0.333}
```

```
        depressible {true=0.333 FALSE=0.666}
        contact {true=0.333 FALSE=0.666}
        schema {SLIDE=0.840 lift=0.04 rotate=0.04 depress=0.04 touch=0.04}
        posture {grasp=0.142 wrap=0.142 pinch=0.142 PALM=0.285 platform=0.142 index=0.142}
        elbow {FLEX=0.5 extend=0.25 fixed=0.25}
        force {LOW=0.5 med=0.25 high=0.25}
        accel {zero=0.2 LOW=0.4 med=0.2 high=0.2}
        dir {away=0.142 TOWARD=0.285 up=0.142 down=0.142 left=0.142 right=0.142}
        aspect {ONCE=0.666 iterated=0.333}
        dur {short=0.25 MED=0.5 long=0.25}
}
```

This format for describing word senses will be used extensively. First, notice that the sense is given a name composed of the verb itself followed by an arbitrary number (`pull3`). In parentheses, the number of training examples from which the sense is derived is shown (here, naturally, it is 1). Then, for each linking feature, the full probability distribution is shown, and its mode value is displayed in capitals if the peakedness of the distribution meets the threshold $MinSetFeature$. In all the output fragments in this chapter, asterisks at the left margin highlight important details discussed in the main text.

Once all the new senses are created, the merge loop is run for each verb in succession. During each iteration of these merge loops, the model's pre-merge posterior probability is measured, the highest-similarity candidate merge is performed, and the posterior is measured again. An excerpt from the training log for *push* follows:

```
*  Starting merging for push (prior=6.63e-79 likelihood=4.19e-176 posterior=2.78e-254):
*  Merging push16 and push6 (similarity 1.0) to form push42 (2 ex) {
      size {SMALL=0.75 large=0.25}
      elongated {TRUE=0.75 false=0.25}
      depressible {true=0.25 FALSE=0.75}
      contact {true=0.25 FALSE=0.75}
      schema {SLIDE=0.911 lift=0.022 rotate=0.022 depress=0.022 touch=0.022}
      posture {GRASP=0.375 wrap=0.125 pinch=0.125 palm=0.125 platform=0.125 index=0.125}
      elbow {flex=0.2 EXTEND=0.6 fixed=0.2}
      force {low=0.2 MED=0.6 high=0.2}
      accel {zero=0.166 low=0.166 MED=0.5 high=0.166}
      dir {AWAY=0.375 toward=0.125 up=0.125 down=0.125 left=0.125 right=0.125}
      aspect {once=0.25 ITERATED=0.75}
      dur {SHORT=0.6 med=0.2 long=0.2}
   }
*  New push prior=5.97e-77 likelihood=1.80e-174 posterior=1.08e-250.
   ...
*  New push prior=1.22e-4 likelihood=3.53e-139 posterior=4.31e-143.
*  Merging push79 and push78 (similarity 0.070) to form push80 (29 ex) {
      size {small=0.516 large=0.483}
      elongated {true=0.516 false=0.483}
      depressible {true=0.064 FALSE=0.935}
```

```
      contact {true=0.258 FALSE=0.741}
      schema {SLIDE=0.822 lift=0.001 rotate=0.001 depress=0.035 touch=0.138}
      posture {grasp=0.4 wrap=0.028 pinch=0.028 palm=0.457 platform=0.028 index=0.057}
      elbow {flex=0.032 EXTEND=0.838 fixed=0.129}
      force {low=0.281 MED=0.468 high=0.25}
      accel {zero=0.060 low=0.333 med=0.454 high=0.151}
      dir {AWAY=0.529 toward=0.029 up=0.029 down=0.029 left=0.235 right=0.147}
      aspect {ONCE=0.806 iterated=0.193}
      dur {short=0.343 med=0.25 long=0.406}
   }
*  New push prior=0.010 likelihood=2.96e-141 posterior=3.25e-143.
*  Posterior decreased; removing push80 and restoring push79 and push78.
*  Stopped merging because the most plausible merge reduced the posterior.
```

Eventually merging stops, often because the posterior has decreased, but sometimes because only one sense remains. The reason for stopping is reported in the log, as can be seen by the final lines in the excerpt.

8.2.2 Tour of the learned lexicon

After training, we are left with a lexicon model which includes a small number of senses for each verb—often only one sense. This section discusses the lexicon arising from the training run described above. The next section reports test results.

The total number of senses is 21, down from 161 initial senses. This amounts to approximately a factor of 8 reduction in the number of parameters being estimated, which is crucial given the very limited amount of training data.

As a by-product of merging, the learning algorithm has adjusted the virtual sample counts for each feature. (Recall that these counts are used when a new verb arrives in the training data.) The new counts are an indication of what features have generally proven important for English verbs. Here they are:

```
Slot 0 virtuals:
   size=1.99 elongated=2.13 depressible=0.424 contact=0.516
   schema=0.036 posture=0.480 elbow=2.58 force=1.07 accel=0.769
   dir=1.27 aspect=1.63 dur=2.10
```

These numbers reveal that the `schema` feature has proven even more criterial than it was pre-wired to be, since its virtual sample count has dropped to 0.036 from 0.05. So have `contact` and `posture`, which have halved their virtual sample counts from their initial setting of 1. Meanwhile, other features like `size` and `duration` have proven to be less important than initially guessed, doubling their virtual sample counts.

When *push* comes to *shove*...

We'll begin with our favorite verb, *push*. Three senses have been learned for this verb, as shown below:

```
   Model for push (3 senses):

   push74 (12 ex) {
      size {SMALL=0.785 large=0.214}
      elongated {true=0.071 FALSE=0.928}
      depressible {TRUE=0.928 false=0.071}
      contact {true=0.071 FALSE=0.928}
*     schema {slide=0.004 lift=0.004 rotate=0.004 DEPRESS=0.983 touch=0.004}
      posture {grasp=0.055 wrap=0.055 pinch=0.055 palm=0.055 platform=0.055 INDEX=0.722}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
      force {low=0.466 med=0.2 high=0.333}
      accel {zero=0.062 low=0.312 MED=0.5 high=0.125}
      dir {away=0.166 toward=0.166 up=0.166 down=0.166 left=0.166 right=0.166}
      aspect {once=0.5 iterated=0.5}
      dur {SHORT=0.6 med=0.266 long=0.133}
   }

   push79 (8 ex) {
      size {small=0.4 large=0.6}
      elongated {true=0.3 FALSE=0.7}
      depressible {true=0.2 FALSE=0.8}
      contact {TRUE=0.7 false=0.3}
*     schema {slide=0.369 lift=0.006 rotate=0.006 depress=0.127 touch=0.490}
      posture {grasp=0.142 wrap=0.071 pinch=0.071 PALM=0.5 platform=0.071 index=0.142}
      elbow {flex=0.1 extend=0.5 fixed=0.4}
      force {low=0.181 MED=0.545 high=0.272}
      accel {zero=0.166 low=0.333 med=0.333 high=0.166}
      dir {AWAY=0.384 toward=0.076 up=0.076 down=0.076 left=0.230 right=0.153}
      aspect {ONCE=0.9 iterated=0.1}
      dur {short=0.363 med=0.090 long=0.545}
   }

   push78 (21 ex) {
      size {small=0.565 large=0.434}
      elongated {TRUE=0.608 false=0.391}
      depressible {true=0.043 FALSE=0.956}
      contact {true=0.086 FALSE=0.913}
*     schema {SLIDE=0.990 lift=0.002 rotate=0.002 depress=0.002 touch=0.002}
      posture {grasp=0.481 wrap=0.037 pinch=0.037 palm=0.370 platform=0.037 index=0.037}
      elbow {flex=0.041 EXTEND=0.916 fixed=0.041}
      force {low=0.333 med=0.416 high=0.25}
      accel {zero=0.04 low=0.32 MED=0.48 high=0.16}
      dir {AWAY=0.518 toward=0.037 up=0.037 down=0.037 left=0.222 right=0.148}
      aspect {ONCE=0.739 iterated=0.260}
      dur {short=0.333 med=0.333 long=0.333}
   }
```

For the most part, the lexicon is distinguishing the senses based on the `schema` feature: we have a sense for DEPRESSing a button (`push74`), a sense for applying pressure to an object for a long time using the TOUCH x-schema (`push79`), and a sense for SLIDE actions (`push78`). As expected, the DEPRESS sense codes for a `depressible` object while the other two senses do not. Other features also vary amongst the three senses. `Push74` codes strongly for the `index posture` since that is the only allowed `posture` in our simple DEPRESS x-schema. `Push79` codes fairly strongly for the `palm posture` and `long duration` and excludes `iterated aspect`, since other parameterizations of TOUCH actions usually lead to other verbs. Both `push79` and `push78` code for `extending` the `elbow` and the `away direction`, as expected. Yet not all the correlations seem plausible: for instance, `push78` codes more strongly for the `grasp posture` than the `palm posture`. And `push79`, despite coding most strongly for the TOUCH x-schema, has merged with some SLIDE examples due to coincident parameterizations.

Next consider *shove*, shown below:

```
   Model for shove (1 sense):

   shove27 (14 ex) {
*     size {small=0.187 LARGE=0.812}
      elongated {true=0.375 FALSE=0.625}
      depressible {true=0.062 FALSE=0.937}
      contact {true=0.312 FALSE=0.687}
      schema {SLIDE=0.985 lift=0.003 rotate=0.003 depress=0.003 touch=0.003}
*     posture {grasp=0.15 wrap=0.05 pinch=0.05 PALM=0.65 platform=0.05 index=0.05}
      elbow {flex=0.058 EXTEND=0.588 fixed=0.352}
*     force {low=0.058 med=0.294 HIGH=0.647}
      accel {zero=0.055 low=0.166 med=0.277 HIGH=0.5}
      dir {away=0.35 toward=0.05 up=0.05 down=0.05 left=0.25 right=0.25}
      aspect {ONCE=0.625 iterated=0.375}
*     dur {SHORT=0.588 med=0.235 long=0.176}
   }
```

It is similar to the third sense of *push*, most notably because it codes for the SLIDE x-schema. But it specifies different parameters. It codes for `higher force`, `shorter duration`, and is more likely to involve the `palm posture` and a `large` object. The senses `shove27` and `push78` overlap significantly, in that there are many actions which receive significant posterior probability from both senses. It is only by virtue of the competition between senses in the LABEL algorithm that a definite border between these two verbs is established.

Two more related verbs are *pull* and *yank*:

```
   Model for pull (1 sense):

   pull25 (13 ex) {
      size {small=0.4 large=0.6}
      elongated {TRUE=0.666 false=0.333}
      depressible {true=0.066 FALSE=0.933}
      contact {true=0.133 FALSE=0.866}
      schema {SLIDE=0.984 lift=0.003 rotate=0.003 depress=0.003 touch=0.003}
      posture {grasp=0.315 wrap=0.052 pinch=0.052 PALM=0.473 platform=0.052 index=0.052}
*     elbow {FLEX=0.875 extend=0.062 fixed=0.062}
      force {low=0.25 med=0.437 high=0.312}
      accel {zero=0.058 low=0.352 med=0.411 high=0.176}
*     dir {away=0.052 TOWARD=0.631 up=0.052 down=0.052 left=0.157 right=0.052}
      aspect {ONCE=0.666 iterated=0.333}
      dur {SHORT=0.5 med=0.312 long=0.187}
   }

   Model for yank (1 sense):

   yank3 (2 ex) {
      size {SMALL=0.75 large=0.25}
      elongated {TRUE=0.75 false=0.25}
      depressible {true=0.25 FALSE=0.75}
      contact {true=0.25 FALSE=0.75}
      schema {SLIDE=0.911 lift=0.022 rotate=0.022 depress=0.022 touch=0.022}
      posture {GRASP=0.375 wrap=0.125 pinch=0.125 palm=0.125 platform=0.125 index=0.125}
*     elbow {FLEX=0.6 extend=0.2 fixed=0.2}
*     force {low=0.2 MED=0.6 high=0.2}
      accel {zero=0.166 low=0.166 med=0.333 high=0.333}
*     dir {away=0.125 toward=0.25 up=0.125 down=0.125 left=0.125 right=0.25}
      aspect {once=0.5 iterated=0.5}
*     dur {SHORT=0.6 med=0.2 long=0.2}
   }
```

Both verbs code for SLIDE actions which involve a `flex`ing `elbow` and are directed `toward` the body, though *yank*'s correlation with this direction is weaker than one would expect. While `yank3` does code more strongly than `pull25` for `short duration`, this tendency is also weak. And there is no appreciable increase in `force` for *yank*. These shortcomings probably result from the very small number of training examples that were labelled with *yank*. What's more, this small number leads to an exaggerated frequency ratio (13:2) between the two senses `pull25` and `yank3`. As a result, *yank* is at a great competitive disadvantage and will rarely be chosen as a label, even for actions which are better *yank*s than *pull*s. Indeed, this happens in the test runs reported in the next section.

The verb *slide* ought to have a more general meaning than any of these verbs. However, since it occurs in the training data only for actions which do not have a more specific label, its learned meaning is skewed:

```
   Model for slide (1 sense):

   slide25 (13 ex) {
      size {small=0.6 large=0.4}
      elongated {true=0.6 false=0.4}
      depressible {true=0.066 FALSE=0.933}
      contact {true=0.133 FALSE=0.866}
      schema {SLIDE=0.984 lift=0.003 rotate=0.003 depress=0.003 touch=0.003}
      posture {GRASP=0.473 wrap=0.052 pinch=0.052 palm=0.315 platform=0.052 index=0.052}
      elbow {flex=0.187 extend=0.187 FIXED=0.625}
      force {low=0.25 MED=0.5 high=0.25}
      accel {zero=0.058 low=0.235 MED=0.529 high=0.176}
*     dir {away=0.052 toward=0.105 up=0.052 down=0.052 left=0.263 RIGHT=0.473}
      aspect {ONCE=0.933 iterated=0.066}
      dur {short=0.437 med=0.125 long=0.437}
   }
```

Slide prefers `left`- or `right`-ward motion over motion `toward` or `away` from the body, instead of learning a uniform `direction` probability distribution. This won't affect labelling results, since we prefer the more specific verbs when they are applicable. But it does affect obeying, because certain legitimate *slide*s (e.g. movement toward the body) will not be generated because they have low likelihood. In general, our architecture is not well suited to learning a hierarchical vocabulary, as opposed to learning a collection of mutually exclusive terms. A major design change would be needed to avoid this sort of problem.

Lastly, the verb *press* is essentially a synonym of one of the senses of *push* (`push74`):

```
   Model for press (1 sense):

   press7 (4 ex) {
      size {small=0.333 LARGE=0.666}
      elongated {true=0.166 FALSE=0.833}
*     depressible {TRUE=0.833 false=0.166}
      contact {true=0.333 FALSE=0.666}
*     schema {slide=0.011 lift=0.011 rotate=0.011 DEPRESS=0.952 touch=0.011}
*     posture {grasp=0.1 wrap=0.1 pinch=0.1 palm=0.1 platform=0.1 INDEX=0.5}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
      force {LOW=0.428 med=0.285 high=0.285}
      accel {zero=0.25 low=0.25 MED=0.375 high=0.125}
      dir {away=0.166 toward=0.166 up=0.166 down=0.166 left=0.166 right=0.166}
      aspect {ONCE=0.833 iterated=0.166}
      dur {short=0.142 med=0.428 long=0.428}
   }
```

It codes for depressing a button with the `index` finger, though its distributions on other features are somewhat different from those of `push74`.

Lifting

Let us now examine a different contrast set involving the LIFT x-schema, including the verbs *lift*, *heave*, *pick up* and *hold*.

```
   Model for lift (1 sense):

   lift21 (11 ex) {
      size {small=0.461 large=0.538}
      elongated {true=0.384 FALSE=0.615}
      depressible {true=0.076 FALSE=0.923}
      contact {true=0.230 FALSE=0.769}
*     schema {slide=0.004 LIFT=0.982 rotate=0.004 depress=0.004 touch=0.004}
      posture {grasp=0.058 wrap=0.176 pinch=0.235 palm=0.058 PLATFORM=0.411 index=0.058}
      elbow {flex=0.214 extend=0.357 fixed=0.428}
      force {low=0.142 med=0.357 high=0.5}
      accel {zero=0.066 low=0.466 med=0.333 high=0.133}
      dir {away=0.058 toward=0.058 UP=0.705 down=0.058 left=0.058 right=0.058}
      aspect {ONCE=0.692 iterated=0.307}
      dur {SHORT=0.5 med=0.214 long=0.285}
   }

   Model for heave (1 sense):

   heave7 (4 ex) {
*     size {small=0.166 LARGE=0.833}
      elongated {true=0.5 false=0.5}
      depressible {true=0.166 FALSE=0.833}
      contact {true=0.166 FALSE=0.833}
      schema {slide=0.011 LIFT=0.952 rotate=0.011 depress=0.011 touch=0.011}
*     posture {grasp=0.1 wrap=0.1 pinch=0.1 palm=0.1 PLATFORM=0.5 index=0.1}
      elbow {flex=0.142 extend=0.428 fixed=0.428}
*     force {low=0.142 med=0.285 HIGH=0.571}
      accel {zero=0.125 low=0.25 MED=0.5 high=0.125}
      dir {away=0.1 toward=0.1 UP=0.5 down=0.1 left=0.1 right=0.1}
      aspect {ONCE=0.666 iterated=0.333}
      dur {SHORT=0.428 med=0.285 long=0.285}
   }

   Model for pickup (1 sense):

   pickup17 (9 ex) {
*     size {SMALL=0.727 large=0.272}
      elongated {true=0.545 false=0.454}
      depressible {true=0.090 FALSE=0.909}
*     contact {true=0.090 FALSE=0.909}
      schema {slide=0.005 LIFT=0.978 rotate=0.005 depress=0.005 touch=0.005}
*     posture {grasp=0.066 wrap=0.333 pinch=0.266 palm=0.066 platform=0.2 index=0.066}
      elbow {flex=0.416 extend=0.166 fixed=0.416}
      force {low=0.25 MED=0.583 high=0.166}
      accel {zero=0.153 LOW=0.461 med=0.307 high=0.076}
      dir {away=0.066 toward=0.066 UP=0.666 down=0.066 left=0.066 right=0.066}
      aspect {ONCE=0.636 iterated=0.363}
```

```
      dur {SHORT=0.5 med=0.333 long=0.166}
   }

   Model for hold (1 sense):

   hold11 (6 ex) {
      size {small=0.5 large=0.5}
      elongated {true=0.125 FALSE=0.875}
      depressible {true=0.125 FALSE=0.875}
      contact {true=0.25 FALSE=0.75}
      schema {slide=0.008 LIFT=0.968 rotate=0.008 depress=0.008 touch=0.008}
      posture {grasp=0.083 wrap=0.083 pinch=0.333 palm=0.083 platform=0.333 index=0.083}
      elbow {flex=0.111 extend=0.111 FIXED=0.777}
*     force {LOW=0.666 med=0.222 high=0.111}
*     accel {ZERO=0.7 low=0.1 med=0.1 high=0.1}
      dir {away=0.083 toward=0.083 UP=0.583 down=0.083 left=0.083 right=0.083}
*     aspect {ONCE=0.875 iterated=0.125}
*     dur {SHORT=0.555 med=0.222 long=0.222}
   }
```

Since all these verbs map to the LIFT x-schema, they are differentiated only by the parameterization of this x-schema. *Lift* is fairly general, but *heave* is more applicable when the object is of `large size` and it prefers actions with `high force` and the `platform posture`. *Pick up* (treated as a single word) is more particular than *lift* or *heave* about the hand not having initial `contact` with the object (which is usually `small`), and also rules out the `platform posture` (instead preferring `pinch` or `wrap`). *Hold* differs from the others by specifying `zero acceleration` (and corresponding `low force`), which, with the LIFT x-schema, corresponds to keeping the object suspended in place—the appropriate meaning for *hold* in our limited action domain. Being steady-state in nature, this action codes for the `once aspect`. (It should also code for `long duration` but doesn't.)

Rotation

The verb *turn* leads to two senses:

```
   Model for turn (2 senses):

   turn26 (8 ex) {
*     size {small=0.1 LARGE=0.9}
      elongated {true=0.5 false=0.5}
      depressible {true=0.1 FALSE=0.9}
      contact {true=0.2 FALSE=0.8}
*     schema {slide=0.006 lift=0.006 ROTATE=0.975 depress=0.006 touch=0.006}
*     posture {GRASP=0.642 wrap=0.071 pinch=0.071 palm=0.071 platform=0.071 index=0.071}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
```

```
      force {low=0.272 MED=0.545 high=0.181}
      accel {zero=0.083 low=0.416 med=0.416 high=0.083}
*     dir {away=0.214 toward=0.142 up=0.071 down=0.071 LEFT=0.357 right=0.142}
      aspect {ONCE=0.8 iterated=0.2}
      dur {short=0.363 med=0.272 long=0.363}
   }

   turn24 (6 ex) {
*     size {SMALL=0.875 large=0.125}
      elongated {true=0.375 FALSE=0.625}
      depressible {true=0.125 FALSE=0.875}
      contact {true=0.125 FALSE=0.875}
*     schema {slide=0.008 lift=0.008 ROTATE=0.968 depress=0.008 touch=0.008}
*     posture {grasp=0.083 wrap=0.083 PINCH=0.583 palm=0.083 platform=0.083 index=0.083}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
      force {LOW=0.555 med=0.222 high=0.222}
      accel {zero=0.1 low=0.2 med=0.4 high=0.3}
*     dir {away=0.083 toward=0.25 up=0.083 down=0.083 left=0.166 right=0.333}
      aspect {ONCE=0.75 iterated=0.25}
      dur {short=0.333 med=0.444 long=0.222}
   }
```

Both senses code for the ROTATE x-schema. Both have fairly broad probability distributions for `direction`, as is appropriate for this rather general verb. So why two separate senses? The learning algorithm has noticed a strong correlation between object `size` and hand `posture`, and separate senses are the only way to preserve the correlation. What the learning algorithm doesn't know is that the ROTATE x-schema is capable of choosing the proper posture based on the object size, and therefore omitting both these features and using only a single sense would not degrade performance. In general, our architecture has no way to convey which relationships amongst linking features are enforced at the x-schema level and hence needn't be learned.

Hitting

Executions of the TOUCH x-schema with `high force` and `acceleration` and `short duration` lead to verbs like *hit* and *slap*:

```
   Model for hit (1 sense):

   hit9 (5 ex) {
      size {SMALL=0.857 large=0.142}
      elongated {true=0.285 FALSE=0.714}
      depressible {true=0.571 false=0.428}
      contact {true=0.142 FALSE=0.857}
*     schema {slide=0.009 lift=0.009 rotate=0.009 depress=0.580 touch=0.390}
```

```
*     posture {grasp=0.090 wrap=0.090 pinch=0.090 palm=0.272 platform=0.090 index=0.363}
      elbow {flex=0.2 extend=0.4 fixed=0.4}
      force {low=0.125 med=0.375 high=0.5}
*     accel {zero=0.111 low=0.111 med=0.111 HIGH=0.666}
      dir {away=0.125 toward=0.125 up=0.125 DOWN=0.375 left=0.125 right=0.125}
      aspect {ONCE=0.857 iterated=0.142}
*     dur {SHORT=0.625 med=0.25 long=0.125}
   }

   Model for slap (1 sense):

   slap7 (4 ex) {
      size {SMALL=0.666 large=0.333}
      elongated {true=0.5 false=0.5}
      depressible {true=0.166 FALSE=0.833}
      contact {true=0.166 FALSE=0.833}
*     schema {slide=0.011 lift=0.011 rotate=0.011 depress=0.011 TOUCH=0.952}
*     posture {grasp=0.1 wrap=0.1 pinch=0.1 PALM=0.5 platform=0.1 index=0.1}
      elbow {flex=0.142 extend=0.428 fixed=0.428}
      force {low=0.142 med=0.285 HIGH=0.571}
*     accel {zero=0.125 low=0.125 med=0.25 HIGH=0.5}
*     dir {away=0.1 toward=0.1 up=0.1 down=0.1 left=0.2 RIGHT=0.4}
      aspect {once=0.5 iterated=0.5}
*     dur {SHORT=0.571 med=0.285 long=0.142}
   }
```

Hit is interesting because it includes significant probability for the DEPRESS x-schema, which corresponds to usages such as *hit the button.* This turns out to be the more common usage in the training set, hence the higher probability on DEPRESS than TOUCH, and the mild tendency for the object to be `depressible`. The `posture` feature also becomes muddled, since the DEPRESS cases involve the `index` finger while the TOUCH cases involve the `palm`. This correlation is lost, unfortunately, because these two types of actions have merged into a single word sense. (Presumably a large portion of the other features happened to be set identically amongst the training examples for this verb, causing the merge to look artificially attractive. This is an example of how the algorithm can be overly sensitive to its tunable parameters, in this case $ModelPriorWeight$.) The sense for *slap*, on the other hand, suffers no such ambiguities, and codes strongly for TOUCH and the `palm posture`. Appropriately, it also keys in on the direction of approach of the arm, requiring a sideways approach (`direction` of `left` or `right`).

Other verbs involving the TOUCH x-schema include *touch*, *poke* and *tap*:

```
Model for touch (1 sense):

touch17 (9 ex) {
```

```
      size {SMALL=0.636 large=0.363}
      elongated {true=0.454 false=0.545}
      depressible {true=0.090 FALSE=0.909}
      contact {true=0.090 FALSE=0.909}
      schema {slide=0.005 lift=0.005 rotate=0.005 depress=0.005 TOUCH=0.978}
      posture {grasp=0.066 wrap=0.066 pinch=0.066 palm=0.266 platform=0.066 INDEX=0.466}
      elbow {flex=0.166 EXTEND=0.583 fixed=0.25}
      force {low=0.5 med=0.416 high=0.083}
      accel {zero=0.076 low=0.461 med=0.384 high=0.076}
      dir {AWAY=0.333 toward=0.066 up=0.066 down=0.133 left=0.2 right=0.2}
*     aspect {ONCE=0.818 iterated=0.181}
      dur {SHORT=0.583 med=0.25 long=0.166}
   }

   Model for poke (1 sense):

   poke5 (3 ex) {
      size {small=0.4 large=0.6}
      elongated {true=0.2 FALSE=0.8}
      depressible {true=0.2 FALSE=0.8}
      contact {true=0.2 FALSE=0.8}
      schema {slide=0.015 lift=0.015 rotate=0.015 depress=0.015 TOUCH=0.938}
      posture {grasp=0.111 wrap=0.111 pinch=0.111 palm=0.111 platform=0.111 INDEX=0.444}
      elbow {flex=0.166 EXTEND=0.666 fixed=0.166}
*     force {low=0.166 med=0.166 HIGH=0.666}
      accel {zero=0.142 low=0.142 med=0.142 HIGH=0.571}
      dir {AWAY=0.333 toward=0.111 up=0.111 down=0.111 left=0.222 right=0.111}
*     aspect {once=0.6 iterated=0.4}
      dur {SHORT=0.666 med=0.166 long=0.166}
   }

   Model for tap (1 sense):

      tap7 (4 ex) {
      size {small=0.5 large=0.5}
      elongated {true=0.333 FALSE=0.666}
      depressible {true=0.333 FALSE=0.666}
      contact {true=0.333 FALSE=0.666}
      schema {slide=0.011 lift=0.011 rotate=0.011 depress=0.247 TOUCH=0.717}
      posture {grasp=0.1 wrap=0.1 pinch=0.1 palm=0.2 platform=0.1 INDEX=0.4}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
*     force {LOW=0.714 med=0.142 high=0.142}
      accel {zero=0.25 LOW=0.5 med=0.125 high=0.125}
      dir {away=0.111 toward=0.111 up=0.111 DOWN=0.333 left=0.111 right=0.222}
*     aspect {once=0.333 ITERATED=0.666}
      dur {SHORT=0.714 med=0.142 long=0.142}
   }
```

Touch and *poke* are quite similar, though *poke* encodes higher `force` and is much less committed to the "`once`" `aspect` than *touch* is. *Tap* also distinguishes itself based on `aspect`, though in this case it is the `iterated` value which is called for. `Low force` is also important for *tap*, and it prefers the `down direction` compared to *poke*'s preference for

`away`. Lastly, an intuition which cannot be captured using our primitive TOUCH x-schema is that *poke* prefers motion of the arm while *tap* prefers flexion and extension of the index finger.

Patterns in the lexicon

To summarize this tour of the lexicon, we call attention to several patterns. Perhaps the most prevalent pattern is that very few senses code for more than one x-schema. This is a direct result of using fewer virtual samples for the `schema` feature than for the other features, discouraging merging of actions involving different x-schemas. The result is appropriate, since the other motor features (and the initial world state features too, for that matter) tend to correlate with the choice of x-schema.

Another pattern is that most verbs collapse down to a single sense. Partly, this is an artifact of our limited range of available x-schemas. I believe that with a richer range of actions, many more verbs would require multiple senses.

8.2.3 Test results

Recognition test results

To evaluate this learned lexicon, we test its ability to label the remaining 15% of the data. Here is the result:

```
Beginning recognition test...
Scenario sc6: desired=push, output=push
Scenario sc12: desired=push, output=push
Scenario sc18: desired=pickup, output=pickup
Scenario sc24: desired=feel, output=feel
Scenario sc30: desired=shove, output=shove
Scenario sc36: desired=heave, output=lift    *ERROR*
Scenario sc42: desired=hold, output=hold
Scenario sc48: desired=slide, output=push    *ERROR*
Scenario sc60: desired=slap, output=slap
Scenario sc66: desired=pull, output=pull
Scenario sc72: desired=turn, output=turn
Scenario sc78: desired=pull, output=pull
Scenario sc84: desired=push, output=push
Scenario sc90: desired=press, output=press
Scenario sc96: desired=turn, output=turn
Scenario sc102: desired=pickup, output=pickup
Scenario sc108: desired=pull, output=pull
Scenario sc114: desired=lift, output=lift
```

```
Scenario sc120: desired=hold, output=hold
Scenario sc126: desired=poke, output=poke
Scenario sc132: desired=pull, output=pull
Scenario sc138: desired=pull, output=pull
Scenario sc144: desired=push, output=push
Scenario sc150: desired=turn, output=turn
Scenario sc156: desired=tap, output=touch    *ERROR*
Scenario sc162: desired=slide, output=slide
Scenario sc168: desired=yank, output=pull    *ERROR*
Scenario sc174: desired=press, output=push    *ERROR*
Scenario sc180: desired=push, output=push
Scenario sc186: desired=heave, output=lift    *ERROR*
Scenario sc192: desired=press, output=push    *ERROR*
Scenario sc198: desired=push, output=push
Correctly labelled 25 of 32 test scenarios (78%).
Recognition test done.
```

A recognition rate of 78% is achieved. Realize that with an 18-word vocabulary, the "chance" recognition rate would be about 6%. Nonetheless, higher recognition rates would have been preferable. We can at least attempt to learn about the errors. In almost all cases, they involve subtle distinctions, not major goofs. For example, consider scenario `sc186`. Here are the posterior probabilities of all word senses in the lexicon for this action:

```
Scenario sc186: desired=heave, output=lift    *ERROR*
sense = tap7, prior = 0.024, likelihood = 1.09e-8, posterior = 2.72e-10
sense = roll3, prior = 0.012, likelihood = 1.62e-7, posterior = 2.02e-9
sense = slap7, prior = 0.024, likelihood = 1.48e-8, posterior = 3.69e-10
sense = push74, prior = 0.074, likelihood = 5.32e-11, posterior = 3.97e-12
sense = push79, prior = 0.049, likelihood = 6.42e-8, posterior = 3.19e-9
sense = push78, prior = 0.130, likelihood = 2.24e-8, posterior = 2.93e-9
sense = hold11, prior = 0.037, likelihood = 3.70e-6, posterior = 1.38e-7
*  sense = pickup17, prior = 0.055, likelihood = 7.63e-5, posterior = 4.26e-6
sense = feel5, prior = 0.018, likelihood = 6.25e-7, posterior = 1.16e-8
sense = press7, prior = 0.024, likelihood = 2.05e-8, posterior = 5.11e-10
sense = turn26, prior = 0.049, likelihood = 2.20e-7, posterior = 1.09e-8
sense = turn24, prior = 0.037, likelihood = 4.92e-9, posterior = 1.83e-10
sense = yank3, prior = 0.012, likelihood = 7.32e-8, posterior = 9.09e-10
sense = poke5, prior = 0.018, likelihood = 2.31e-8, posterior = 4.31e-10
*  sense = lift21, prior = 0.068, likelihood = 4.94e-4, posterior = 3.37e-5
sense = hit9, prior = 0.031, likelihood = 2.89e-9, posterior = 8.99e-11
sense = pull25, prior = 0.080, likelihood = 4.08e-9, posterior = 3.29e-10
*  sense = heave7, prior = 0.024, likelihood = 4.01e-4, posterior = 9.98e-6
sense = slide25, prior = 0.080, likelihood = 1.82e-8, posterior = 1.47e-9
sense = touch17, prior = 0.055, likelihood = 5.02e-8, posterior = 2.80e-9
sense = shove27, prior = 0.086, likelihood = 5.47e-9, posterior = 4.76e-10
```

Note that the correct label, *heave*, has barely lost out to the actual label, *lift* (by a factor of about three). *Pick up* is also competitive (which is reasonable, since it is a related verb), being within a factor of ten, but all other word senses are far behind. Most of the errors in

this training run are like this. Speakers are likely never to be perfectly consistent in their labelling of actions, rendering errors of this sort inevitable. However, it is also possible that the errors are exacerbated by the small amount of training data that was used. So, while it is probably appropriate to "discount" most of these errors, I would prefer to test on more data before drawing this conclusion.

Command-obeying test results

Now consider command-obeying. We use the same 15% of the data as used for recognition testing. The labels in the data are used as commands, which are interpreted in light of the initial world state features in each scenario. The resulting linking features are labelled, and the label is compared to the initial command. Results follow:

```
Beginning obey test.
Scenario sc6: command=push, output=push
Scenario sc12: command=push, output=push
Scenario sc18: command=pickup, output=pickup
Scenario sc24: command=feel, output=feel
Scenario sc30: command=shove, output=shove
Scenario sc36: command=heave, output=lift    *ERROR*
Scenario sc42: command=hold, output=hold
Scenario sc48: command=slide, output=slide
Scenario sc60: command=slap, output=slap
Scenario sc66: command=pull, output=pull
Scenario sc72: command=turn, output=turn
Scenario sc78: command=pull, output=pull
Scenario sc84: command=push, output=push
Scenario sc90: command=press, output=push    *ERROR*
Scenario sc96: command=turn, output=turn
Scenario sc102: command=pickup, output=pickup
Scenario sc108: command=pull, output=pull
Scenario sc114: command=lift, output=lift
Scenario sc120: command=hold, output=hold
Scenario sc126: command=poke, output=poke
Scenario sc132: command=pull, output=pull
Scenario sc138: command=pull, output=pull
Scenario sc144: command=push, output=push
Scenario sc150: command=turn, output=turn
Scenario sc156: command=tap, output=tap
Scenario sc162: command=slide, output=slide
Scenario sc168: command=yank, output=pull    *ERROR*
Scenario sc174: command=press, output=push    *ERROR*
Scenario sc180: command=push, output=push
Scenario sc186: command=heave, output=lift    *ERROR*
Scenario sc192: command=press, output=push    *ERROR*
Scenario sc198: command=push, output=push
Correctly recognized 26 of 32 obeyed commands (81%).
Obey test done.
```

Note that the success rate of 81% is higher than that for recognition. In other training runs, it is often significantly higher. The explanation is that the motor-parameter linking features of obeyed actions are *generated by the model itself*, via the interpretation of the command. The resulting actions thus basically correspond to the prototypes embodied in the lexicon. So it is not surprising that the model is in turn able to recognize these prototypical actions very successfully.

As was the case for recognition testing, the errors observed above are all cases in which both the actual and desired labels are plausible.

An example of command-obeying

To illustrate how the model chooses an appropriate interpretation of a command verb (i.e. a sense which best fits the initial world state), we will go through how the learned lexicon handles a *push* command in several initial world states.

We first give the *push* command in scenario `sc41`, which has the following initial world state:

```
{size=small elongated=false depressible=true contact=false}
```

The important cue is that the object is `depressible`, and the obeying algorithm correctly chooses the DEPRESS sense of push, as can be seen from the resulting linking feature instructions to the x-schema execution system:

```
Linking f-struct (pre-execution):
   {size=small elongated=false depressible=true contact=false
   schema=depress posture=index accel=med dur=short}
```

The next *push* command is given in scenario `sc6`, which has a different initial world state:

```
{size=large elongated=false depressible=false contact=false}
```

In this case, the SLIDE sense is chosen:

```
Linking f-struct (pre-execution):
   {size=large elongated=false depressible=false contact=false
   schema=slide elbow=extend accel=med dir=away aspect=once}
```

In either case, the x-schema which is chosen to execute has its proper parameters set, since these were learned separately for each of the two senses.

Trajectory of learning

How does the model's posterior probability, and also the recognition rate, change as merging proceeds? The plot in Figure 8.2 shows these statistics for the training run we have been considering. The plot reflects *only* the model for the verb *push*. The x-axis ranges from before merging begins until after it is done (40 merges total).

The top plot displays, on a logarithmic scale, the *push* model's posterior probability as well as its prior probability and likelihood (recall posterior $\propto$ prior $\times$ likelihood). As expected, the prior probability steadily increases during merging—indeed it is perfectly linear on this log plot, since the prior is just an exponential function of the (steadily-decreasing) number of word senses. The likelihood also increases initially, but eventually decreases. Why does the likelihood not drop until near the end of the training period? The explanation is that initially, there are a very large number of word senses (one for each example), and these give rise to a large number of "simple" merges of identical or nearly-identical senses. These merges tend to increase the likelihood since they decrease the relative importance of the virtual samples (with their "fuzzifying" effect) in those senses. Only after these easy merges are performed do dissimilar senses begin to merge. Since fewer senses exist at this point, fewer merges will occur during this generalization-inducing, likelihood-dropping phase. Turning to the third curve, the posterior can be seen to increase monotonically, as required by our learning algorithm. It can also be seen to level off toward the end due to the decreasing likelihood.

During this process, the recognition rate increases from 83% to 100%. Since there are only six test scenarios for *push* (five of which are recognized correctly from the start), this curve is rather abrupt and certainly does not tell us much about recognition performance in general. The important points are (1) that the recognition does not drop despite the shrinking of the model, and (2) that at some point the generalization ability of the model does in fact increase to cover the sixth test scenario.

To get a better handle on how generalization ability (as reflected in the recognition rate) relates to the number of word senses, a series of training runs were performed with different settings of the $ModelPriorWeight$ parameter. The result is a collection of lexicons with differing numbers of word senses. And since they are fully trained, we can then test each lexicon on the *full set* of recognition test scenarios, and thus obtain more reliable numbers.

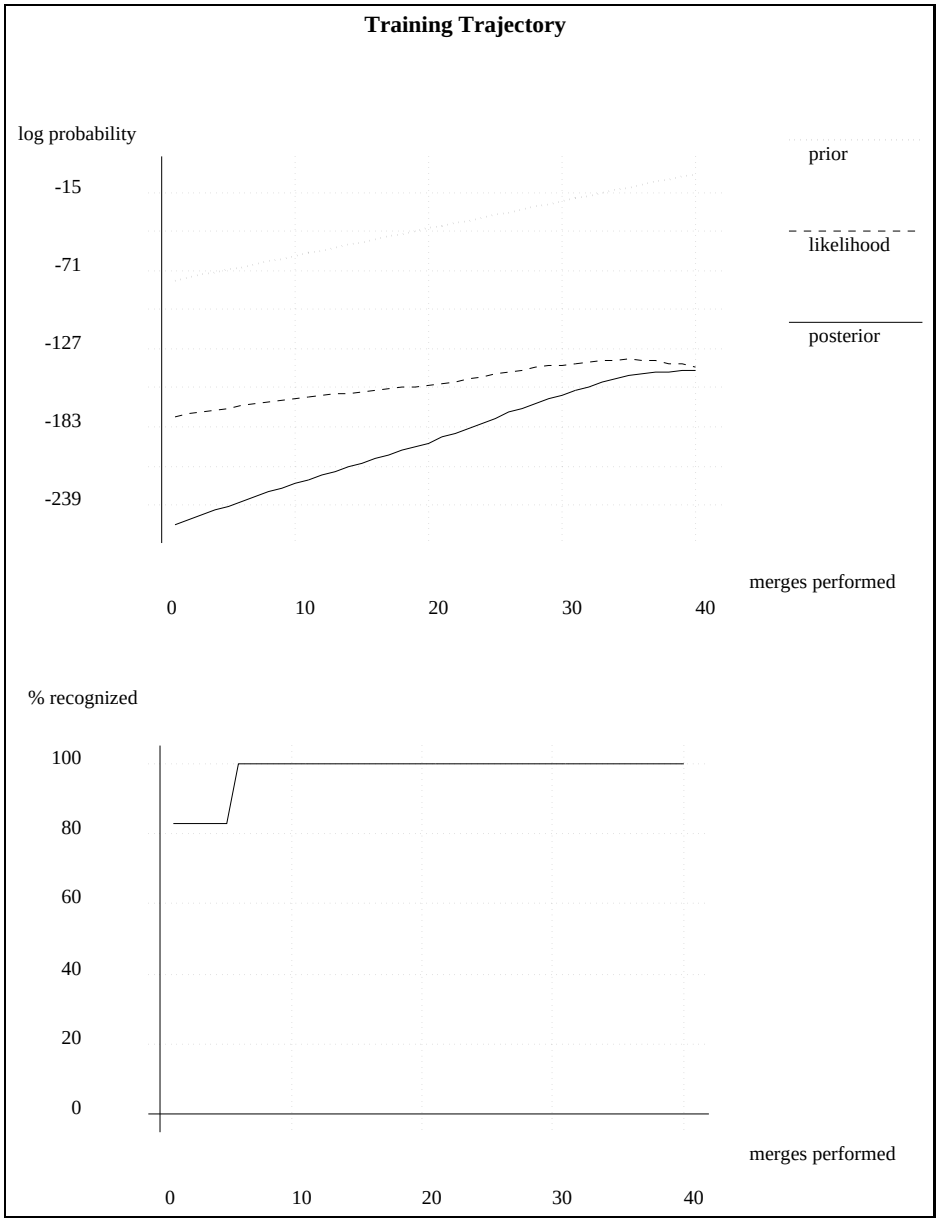


Figure 8.2: Plot of the *push* model's prior, likelihood and posterior probabilities, as well as the recognition rate, as merging proceeds.

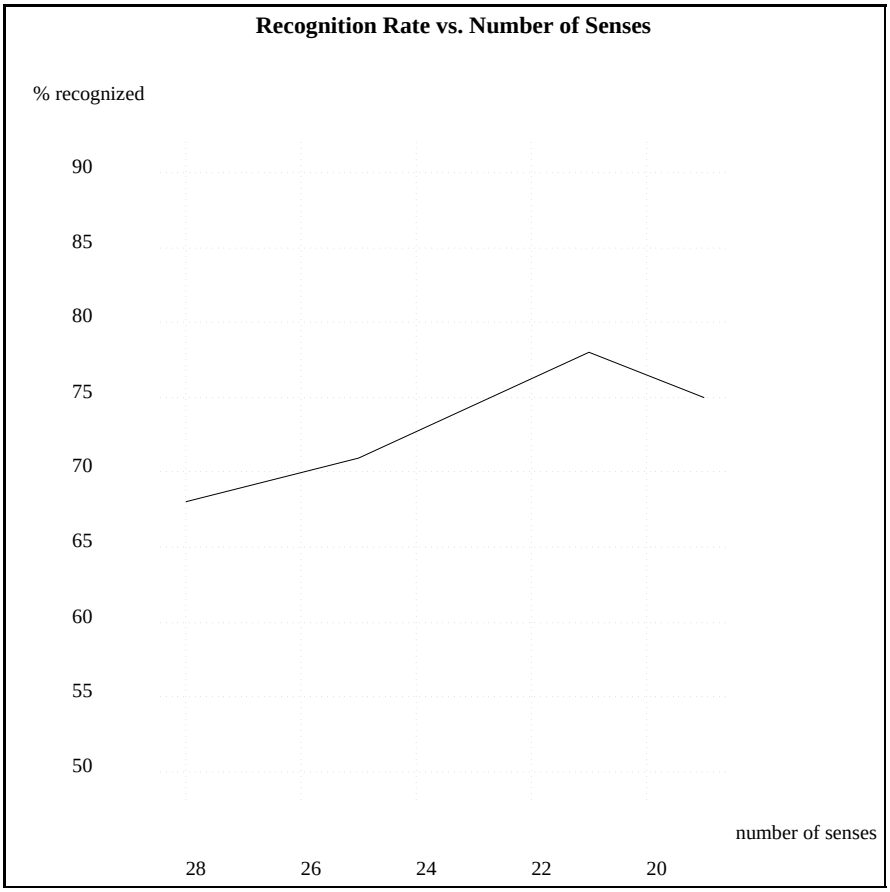


Figure 8.3: Plot of the recognition rate for various learned lexicons with different total numbers of word senses.

The results are shown in the plot in Figure 8.3. The lexicons are arranged on the x-axis in decreasing order of the number of word senses they contain. Recognition performance (i.e. generalization ability) is seen to *increase* as the number of senses *decreases*, up to a point—which is just what we want. However, if too much merging is performed, recognition performance eventually decreases. The optimum lexicon size is 21 senses.

If we were to repeat the experiment using the training set itself for recognition testing, we would get a different curve. In this case, memorization is the best policy and generalization can only hurt. Indeed, in one such series of training runs, a lexicon with 151 senses achieved 96% recognition, while lexicons with 30 and fewer senses achieved only about 80% recognition.

8.3 Crosslinguistic Validation

8.3.1 Farsi

The Farsi language exhibits some non-English-like distinctions within the sphere of actions represented by *push*, *press*, *pull* and their ilk. A subset of 61 of the scenarios used for the English training run were labelled with four Farsi verbs[4] and the system was trained using the same parameters used for English.

Farsi distinguishes two of English's familiar senses of *push*. One verb, *hol daadan*, encodes away-directed motion. Another verb, *feshaar daadan*, encodes applying force without motion. They are learned as follows:

```
   Model for hol_daadan (1 sense):

   hol_daadan17 (9 ex) {
      size {small=0.181 LARGE=0.818}
      elongated {true=0.272 FALSE=0.727}
      depressible {true=0.090 FALSE=0.909}
      contact {true=0.181 FALSE=0.818}
*     schema {SLIDE=0.978 lift=0.005 rotate=0.005 depress=0.005 touch=0.005}
*     posture {grasp=0.133 wrap=0.066 pinch=0.066 PALM=0.6 platform=0.066 index=0.066}
      elbow {flex=0.083 EXTEND=0.833 fixed=0.083}
*     force {low=0.083 med=0.166 HIGH=0.75}
      accel {zero=0.076 low=0.076 MED=0.538 high=0.307}
*     dir {AWAY=0.666 toward=0.066 up=0.066 down=0.066 left=0.066 right=0.066}
*     aspect {ONCE=0.909 iterated=0.090}
      dur {short=0.166 med=0.333 LONG=0.5}
   }

   Model for feshaar_daadan (2 senses):

   feshaar_daadan74 (26 ex) {
      size {SMALL=0.714 large=0.285}
      elongated {true=0.035 FALSE=0.964}
      depressible {TRUE=0.964 false=0.035}
      contact {true=0.142 FALSE=0.857}
*     schema {slide=0.001 lift=0.001 rotate=0.001 DEPRESS=0.992 touch=0.001}
      posture {grasp=0.031 wrap=0.031 pinch=0.031 palm=0.031 platform=0.031 INDEX=0.843}
      elbow {flex=0.333 extend=0.333 fixed=0.333}
      force {low=0.413 med=0.310 high=0.275}
      accel {zero=0.133 low=0.266 MED=0.433 high=0.166}
      dir {away=0.166 toward=0.166 up=0.166 down=0.166 left=0.166 right=0.166}
      aspect {ONCE=0.642 iterated=0.357}
      dur {SHORT=0.517 med=0.310 long=0.172}
   }
```

[4] Three of these four labels are in fact composed of multiple words. However, they were treated as single verbs in this training run, as indicated by underscores in the transcripts.

```
   feshaar_daadan73 (12 ex) {
      size {small=0.5 large=0.5}
      elongated {TRUE=0.642 false=0.357}
      depressible {true=0.071 FALSE=0.928}
      contact {true=0.357 FALSE=0.642}
*     schema {slide=0.004 lift=0.004 rotate=0.004 depress=0.004 TOUCH=0.983}
*     posture {grasp=0.055 wrap=0.055 pinch=0.055 PALM=0.722 platform=0.055 index=0.055}
      elbow {flex=0.2 extend=0.466 fixed=0.333}
      force {low=0.4 med=0.266 high=0.333}
      accel {zero=0.062 LOW=0.437 med=0.25 high=0.25}
      dir {away=0.235 toward=0.058 up=0.058 down=0.294 left=0.176 right=0.176}
      aspect {ONCE=0.928 iterated=0.071}
*     dur {short=0.266 med=0.2 LONG=0.533}
   }
```

These verbs are distinguished mainly by the fact that they map to different x-schemas. *Hol daadan* maps to the SLIDE x-schema, while *feshaar daadan* has two senses, one mapping to TOUCH and another mapping to DEPRESS. Both *hol daadan* and the TOUCH sense of *feshaar daadan* code strongly for the `palm posture`. The TOUCH sense of *feshaar daadan* requires `long duration` and cannot be iterated; it is similar to English *lean*. *Hol daadan* codes movements away from the body, requires `high force`, and prefers quick motions but does allow more continuous pushing. A closely related verb, *pas zadan*, parameterizes SLIDE differently:

```
   Model for pas_zadan (1 sense):

   pas_zadan9 (5 ex) {
      size {small=0.142 LARGE=0.857}
      elongated {TRUE=0.714 false=0.285}
      depressible {true=0.142 FALSE=0.857}
      contact {true=0.142 FALSE=0.857}
      schema {SLIDE=0.961 lift=0.009 rotate=0.009 depress=0.009 touch=0.009}
      posture {grasp=0.090 wrap=0.090 pinch=0.090 PALM=0.545 platform=0.090 index=0.090}
      elbow {flex=0.125 EXTEND=0.75 fixed=0.125}
*     force {low=0.375 med=0.375 high=0.25}
      accel {zero=0.111 LOW=0.555 med=0.111 high=0.222}
      dir {AWAY=0.545 toward=0.090 up=0.090 down=0.090 left=0.090 right=0.090}
*     aspect {once=0.285 ITERATED=0.714}
*     dur {SHORT=0.75 med=0.125 long=0.125}
   }
```

It shares the preference for the `away direction` and the `palm posture`, but prefers `lower force` and `short duration`. It allows an `iterative` interpretation whereas *hol daadan* does not.

The closest equivalent to *pull* in Farsi is *keshidan*:

```
   Model for keshidan (1 sense):

   keshidan17 (9 ex) {
      size {small=0.545 large=0.454}
      elongated {true=0.454 false=0.545}
      depressible {true=0.090 FALSE=0.909}
      contact {true=0.090 FALSE=0.909}
      schema {SLIDE=0.978 lift=0.005 rotate=0.005 depress=0.005 touch=0.005}
      posture {GRASP=0.466 wrap=0.066 pinch=0.066 palm=0.266 platform=0.066 index=0.066}
      elbow {FLEX=0.833 extend=0.083 fixed=0.083}
*     force {low=0.25 med=0.333 high=0.416}
      accel {zero=0.076 low=0.230 med=0.384 high=0.307}
      dir {away=0.066 TOWARD=0.666 up=0.066 down=0.066 left=0.066 right=0.066}
*     aspect {ONCE=0.909 iterated=0.090}
*     dur {short=0.25 MED=0.5 long=0.25}
   }
```

Compared to *pull*, *keshidan* codes for `higher` `force` and `longer` `duration`. It also prefers the `once` `aspect`. In some respects it resembles English *haul*.

8.3.2 Russian

The next language we consider is Russian. From the perspective of our model, Russian is interesting because its verbs almost always appear with prefixes and suffixes. Thus, we have used Russian as a testbed for the multi-slot version of our model as presented in Chapter 7. The results are mixed.

The same set of 200 scenarios described above was labelled by a Russian informant. The set of labels used was as follows:[5]

Prefixes	Roots	Suffixes
po- pere- pri- no- ot- pod-	trogat tolknut davit zhat dvinut stuknut derzhat tyanut vernut tknut	-ut -at

As might be expected, the different grammatical positions tend to encode different aspects of the semantics of actions. In particular, it is clear that the suffixes code for perfectiveness. The verb roots tend to code for goals, posture and effort. The prefixes tend to encode more image-schematic features such as paths. We will soon see how the model captures these tendencies.

[5]Note that, in the process of dividing the labels into three slots, some liberties have been taken with spelling. In particular, verb root labels retain a suffix when they would be difficult to recognize without it.

Training

Training the model on this data proceeded differently than it did for English and Farsi in several respects. First, there are effectively three times as many training examples for Russian than for the other languages, since each training example involves incorporating a new instance for each of the three components of its label. Training times grew accordingly.

Secondly, since there are only two possible suffixes each of them occurs quite often in the training set. The offline version of the model merging algorithm, since its runtime complexity is cubic in the number of training examples (see §6.2.5), proved intractable. It was essential to utilize the $BatchSize$ parameter to implement online learning; a batch size of ten turned out to produce reasonable results and greatly speeded up learning.

Satisfyingly, it did not prove necessary to modify the value of $ModelPriorWeight$, even in the face of these different training set statistics. This is an encouraging sign that the model merging criterion is robust to such variations. This kind of robustness would be essential in modelling a wide range of languages.

Tour of the learned lexicon

We will not present the entire learned lexicon here, but rather just mention some highlights. One of the more interesting results is the representation for the imperfective suffix *-at*. It is learned as follows. (To save space, this section will present word senses by simply listing the set of feature values which would be used for obeying a command.)

```
Model for at (4 senses):

at58 (11 ex)
   {elongated=false depressible=false contact=false schema=lift
   accel=med dir=up aspect=iterated dur=short}

at79 (5 ex)
   {size=small elongated=false depressible=true contact=false
   schema=depress posture=index force=high aspect=iterated
   dur=short}

at78 (22 ex)
   {size=large depressible=false contact=false schema=slide
   posture=palm aspect=iterated dur=short}

at51 (2 ex)
   {elongated=true depressible=false contact=false schema=touch
   posture=palm accel=low aspect=once dur=long}
```

In the judgment of the informant, imperfectiveness corresponded to two types of x-schema execution. One type corresponded to iterated but brief motions, and the other type corresponded to non-iterated but long-duration motions. In order to capture this dependence between the `aspect` and `duration` features, multiple senses are required. Observe that `at51` encodes non-iterated but long actions. The other three senses encode iterated short actions. Unfortunately the model has not seen enough data to generalize over some of the other features, resulting in the three separate senses for the case of iterated short actions. This does not negatively impact recognition performance, though, as will be shown shortly.

The learned representations for the verb roots are generally acceptable, in that the appropriate correlations are captured. However, a large number of spurious associations are also picked up. It is not clear if more data would help. Several examples are listed here. To begin, consider the roots *tolknut*, which corresponds to crude and forceful pushing involving object motion, and *dvinut*, which is used for "calmer" sliding motions. They are learned as follows:

```
Model for tolknut (1 sense):

tolknut31 (16 ex)
   {size=large depressible=false contact=false schema=slide
   posture=palm force=high accel=med aspect=once}

Model for dvinut (3 senses):

dvinut21 (1 ex)
   {size=small elongated=true depressible=false contact=true
   schema=touch posture=palm elbow=fixed force=med
   accel=med dir=away aspect=once dur=long}

dvinut95 (30 ex)
   {size=small depressible=false contact=false schema=slide
   posture=grasp elbow=extend aspect=once}

dvinut94 (16 ex)
   {size=large depressible=false contact=false schema=slide
   posture=palm force=med dur=short}
```

`Tolknut31` properly codes for the SLIDE x-schema, the `palm` posture, and `high force`. In contrast, *dvinut* codes for a less restrictive set of SLIDEs, since it is modelled by a sense (`dvinut95`) involving the `grasp posture` as well as a sense (`dvinut94`) involving the `palm posture`. `High force` is not coded for by either sense.

Consider another example, the verb root *stuknut*. It corresponds to English *hit* or *slap*, and is learned as follows:

```
Model for stuknut (2 senses):

stuknut27 (10 ex)
   {size=small depressible=false contact=false schema=touch
   posture=palm force=high accel=high dur=short}

stuknut24 (4 ex)
   {size=small elongated=false depressible=true contact=false
   schema=depress posture=index force=high aspect=iterated
   dur=short}
```

This verb is learned rather well. The first sense, `stuknut27`, encodes high-speed TOUCHes using the `palm`, which corresponds to either English *hit* or *slap* depending on the direction of approach of the arm—which the sense properly does not code for. The other sense, `stuknut24`, codes for hitting a button. It insists on `high force`, which is correct since a different Russian verb root (*zhat*) is used to refer to gentler button-pushing, as can be seen in the first sense for *zhat* shown below:

```
Model for zhat (2 senses):

zhat25 (12 ex)
   {size=small elongated=false depressible=true contact=false
   schema=depress posture=index force=low aspect=once
   dur=short}

zhat8 (1 ex)
   {size=large depressible=false contact=false schema=slide
   posture=palm force=high accel=high dir=toward
   dur=short}
```

Prefixes prove to be the most troublesome in Russian, due to the inadequacies of our linking features for representing image-schematic concepts. For example, the prefix *pod-* can mean "under" or "a little". It is learned as shown below—no such image-schematic pattern is obvious. (Although, again, the representation does a passable job in recognition testing as reported shortly.)

```
Model for pod (3 senses):

pod57 (3 ex)
   {size=large depressible=false contact=false schema=slide
   posture=palm elbow=flex accel=med dir=toward}
```

```
pod66 (8 ex)
   {size=small depressible=false contact=false schema=lift
   accel=low aspect=once}

pod58 (22 ex)
   {size=large depressible=false contact=false schema=lift
   posture=platform dir=up aspect=once dur=short}
```

Lastly, let us examine the virtual sample counts in each of the three slots of the model, to see how they have adapted in response to the training data:

```
Slot 0 (prefix) virtuals:
   size=3.46 elongated=3.48 depressible=0.024 contact=0.225
   schema=0.065 posture=0.188 elbow=2.71 force=1.54 accel=0.723
   dir=1.72 aspect=0.977 dur=2.19

Slot 1 (root) virtuals:
   size=1.13 elongated=5.07 depressible=0.057 contact=0.322
   schema=0.002 posture=0.119 elbow=4.33 force=0.903 accel=0.722
   dir=2.39 aspect=1.57 dur=1.47

Slot 2 (suffix) virtuals:
   size=6.13 elongated=2.62 depressible=0.023 contact=0.148
   schema=0.078 posture=0.272 elbow=3.75 force=2.26 accel=0.778
   dir=1.98 aspect=0.075 dur=3.75
```

This array of numbers is a far cry from representing a clear-cut decision about the semantic roles played by the three slots. But we can see that a few trends have been partially picked up. For instance, note that slot 1 (the verb root slot) codes much more strongly for the `schema` feature than the other two slots, as we would expect given the tendency for Russian verb roots to encode the goal and overall type of action. Slot 2 (the suffix slot) codes much more strongly for the `aspect` feature than the other two slots, which is consistent with its role as a perfectiveness marker. Lastly, slot 0 (the prefix slot) appropriately codes most strongly for `direction`—which is one of the more image-schematic of the model's linking features—although the actual virtual sample count is not as low as one might have expected.

In summary, the three-slot Russian lexicon appears to have been learned somewhat less cleanly than one-slot English or Farsi. The question then is, how does its recognition performance compare?

Recognition testing

The model was tested on the 33 test scenarios with the following results:

```
Beginning recognition test...
Scenario sc6: desired=ot-dvinut-ut, output=ot-tolknut-ut    *ERROR*
Scenario sc12: desired=ot-dvinut-at, output=ot-dvinut-at
Scenario sc18: desired=pod-nyat-ut, output=pod-nyat-ut
Scenario sc24: desired=-trogat-at, output=po-trogat-at    *ERROR*
Scenario sc30: desired=ot-dvinut-at, output=ot-dvinut-at
Scenario sc36: desired=pod-nyat-ut, output=pod-nyat-ut
Scenario sc42: desired=pod-derzhat-ut, output=pod-derzhat-ut
Scenario sc48: desired=pere-dvinut-ut, output=pere-dvinut-ut
Scenario sc54: desired=po-dvinut-ut, output=po-dvinut-at    *ERROR*
Scenario sc60: desired=-stuknut-ut, output=pri-stuknut-ut    *ERROR*
Scenario sc66: desired=pri-dvinut-at, output=pri-dvinut-at
Scenario sc72: desired=ot-katit-ut, output=pere-katit-ut    *ERROR*
Scenario sc78: desired=pri-dvinut-ut, output=pri-dvinut-ut
Scenario sc84: desired=pere-dvinut-ut, output=po-dvinut-ut    *ERROR*
Scenario sc90: desired=na-zhat-ut, output=na-davit-ut    *ERROR*
Scenario sc96: desired=po-vernut-ut, output=po-vernut-ut
Scenario sc102: desired=pod-nyat-at, output=pod-nyat-at
Scenario sc108: desired=pri-dvinut-ut, output=pri-dvinut-ut
Scenario sc114: desired=pod-nyat-ut, output=pod-nyat-ut
Scenario sc120: desired=pod-derzhat-ut, output=pod-derzhat-ut
Scenario sc126: desired=-tknut-at, output=na-stuknut-at    *ERROR*
Scenario sc132: desired=pri-dvinut-ut, output=na-dvinut-ut    *ERROR*
Scenario sc138: desired=pri-dvinut-ut, output=pri-dvinut-at    *ERROR*
Scenario sc144: desired=na-zhat-ut, output=na-zhat-ut
Scenario sc150: desired=pere-katit-ut, output=pere-vernut-ut    *ERROR*
Scenario sc156: desired=-trogat-at, output=na-stuknut-at    *ERROR*
Scenario sc162: desired=po-dvinut-at, output=po-dvinut-at
Scenario sc168: desired=pod-tyanut-ut, output=pri-vernut-ut    *ERROR*
Scenario sc174: desired=na-zhat-at, output=na-zhat-at
Scenario sc180: desired=-stuknut-at, output=na-zhat-at    *ERROR*
Scenario sc186: desired=pod-nyat-ut, output=pod-nyat-ut
Scenario sc192: desired=na-davit-ut, output=na-davit-ut
Scenario sc198: desired=po-dvinut-ut, output=po-dvinut-ut
Correctly labelled 19 of 33 test scenarios (57%).
Recognition test done.
```

The first thing to notice is that the overall recognition rate is somewhat lower lower than it is for English and Farsi. There are several reasons for this. The first reason is that the three-slot recognition task is simply much harder than the one-slot recognition task. While random guessing on the 18-verb English vocabulary would yield a recognition rate of 6%, the random guessing rate for the Russian data, with its 7 possible prefixes, 10 roots, and 3 suffixes (including the possibility for null prefixes and suffixes), is less than 0.5%. Viewed in this light, the results look rather good.

A more serious problem, though, is that it proved impossible to set the *MinLabel* parameter appropriately. Several of the test examples require the prefix slot to be left unfilled, which can only be accomplished by setting *MinLabel* above 0. However, any

setting of *MinLabel* which accomplished this goal exhibited the unfortunate side-effect of omitting too many labels, often including verb roots. In other words, this parameter is very brittle.

A related problem occurred regarding the *MinExplain* parameter. Recall that the multi-slot labelling algorithm always fills the slot for which it has found the highest-posterior-probability label, and then removes (or "explains away") the features strongly coded for by that label before going on to fill the next slot. *MinExplain* defines the criterion for which features will be removed. Since many word senses are coding for spurious features almost as strongly as for the proper features, this parameter, too, has proven brittle. When set too low (e.g. less than 0.8 or so), features which are spuriously correlated with the label in one slot get explained away and then are not available to help choose the label for the remaining slots. This problem manifests itself most clearly by producing drastically incorrect verb root labels—errors which do not occur if *MinLabel* is set to a very high value (e.g. 0.95, which virtually disables the explaining away strategy).

One hypothesis—although it is only a hunch—is that if explaining away were to be utilized during learning, then it would prove more effective during testing. The reason is that fewer spurious correlations (hopefully) would make their way into the learned lexicon, and therefore explaining away during recognition would remove only the proper features. This remains to be implemented and tested.

Variations

An interesting question is how the performance on verb roots alone compares to the one-slot training reported in earlier sections. If we examine the above trace and count as errors only those scenarios for which the verb root label is wrong, we obtain a recognition rate of 81%. This is in line with the earlier one-slot results.

To further investigate this issue, the Russian training data was stripped of all prefixes and suffixes and used to train a one-slot model. This strategy for obtaining one-slot data is somewhat suspect. For example, an action labelled *hold down* may well be labelled *press*, not *hold*, when the informant is requested to give one-slot labels. But the strategy is convenient and allows us to get a rough measure of what kind of performance might be obtained with "real" one-slot labels. As it turns out, the recognition errors for

this one-slot model are identical to the errors on the verb root slot in the three-slot lexicon (and thus are also comparable to the English results).

Finally, there is the question of what improvment is obtained by dividing the multi-word Russian labels into slots compared to treating them as wholes. To get a measure of this improvement, a one-slot model was trained with the three-slot labels, except each label was treated as a single "verb". The resulting lexicon consisted of 43 total word senses when trained using the same algorithm parameters as English. This is a few more than the total number of senses in the three-slot lexicon. One would expect much lower recognition performance, though, since the one-slot model is not generative, i.e. it cannot form novel combinations of roots, prefixes and suffixes. However, the recognition performance obtained was 54%—comparable to the three-slot case. This appears to be due to there being less productivity within the labelled data than was expected. Most of the target labels in the recognition test had in fact appeared in the training data. Whether this would remain true with more data or more actions is unknown.

8.3.3 Other crosslinguistic examples

A number of other interesting crosslinguistic examples have come to the author's attention,[6] but have not been trained on due to either lack of full data for the language, or lack of time. Nevertheless, many of these examples seem to be representable—or nearly representable—within the model developed here, and so are worth reporting. This section, then, presents some of these examples along with guesses as to how they might be represented in terms of our linking features. These guesses are based on the informal descriptions of the verb meanings provided by informants. They are written here in an abbreviated form for clarity and to emphasize that they are hypothetical.

Postural coding in Korean and Spanish

A variety of languages distinguish actions based upon hand posture. Consider two examples from Korean, *um kyo gi da* and *dul da*. Both are similar to English *hold*. But *um kyo gi da* codes specifically for holding a small object with the fingers wrapped around it, while *dul da* codes for heavier objects held with a flat hand. They might be represented as

[6] Many of them thanks to Carol Bleyle's interviews with her ESL students.

follows:

```
Model for um_kyo_gi_da:
   {size=small schema=lift posture=grasp accel=zero}

Model for dul_da:
   {size=large schema=lift posture=platform accel=zero}
```

Note that the `grasp` posture is not quite right for *um kyo gi da*; to perfectly capture this expression, we would need to further refine the `posture` feature to include more of the postures shown in Figure 3.1.

Another example from Korean is the case of *kon dur ida* compared with *tu chi da*. The first refers to tipping an object over using one finger, while the latter refers to tipping a large object. These might be represented as follows:

```
Model for kon_dur_ida:
   {size=small schema=rotate posture=index}

Model for tu_chi_da:
   {size=large schema=rotate posture='grasp or palm'}
```

A very interesting example from Korean is *kul ri da*, which refers to rolling an object off of the hand (and then across a surface, normally). Representing such an action is beyond the capabilities of the current set of x-schemas and linking features.

Turning to Spanish, we have two related verbs, *pulsar* and *presionar*. *Pulsar* refers to pressing with one finger (especially a button), while *presionar* refers to pressing with the palm. These verbs map to the DEPRESS and TOUCH x-schemas, respectively, and should correspond approximately to the two senses of Farsi *feshaar daadan* shown earlier.

Lastly, Spanish distinguishes general hitting (*pegar*) from hitting with a flat palm (*golpear*). This distinction can easily be captured by the `posture` feature; in one case the probability distribution is broad, while in the other case it is highly peaked around the `palm` value.

Aspectual coding in Tamil, Korean and Japanese

Another type of distinction found in verbs from some languages involves aspect. In particular, some verbs code for repetition of short-duration actions while other verbs code for a smooth and continuous version of the same basic action.

This shows up, for example, in the Tamil verbs for pushing (*thallu*) and pulling (*ilu*). Whereas most English speakers assume a continuous motion by default, the Tamil verbs, when used alone, strongly suggest a sudden motion, and may even suggest repetition of the sudden motion. (The repetition can be emphasized by reduplication—i.e., with *thallu-thallu*.) In order to suggest continuous motion, a directional suffix can be added, such as *-po* (away) or *-wa* (toward). *Thallu* might be represented as follows:

```
Model for thallu: {
   schema {SLIDE=0.9 ...}
   posture {PALM=0.9 ...}
   duration {SHORT=0.5 med=0.3 long=0.2}
   aspect {once=0.4 ITERATED=0.6}
}
```

Meanwhile, the suffix *-po* might be represented as:

```
Model for po: {
   direction {AWAY=0.9 ...}
   duration {short=0.1 med=0.3 LONG=0.6}
   aspect {ONCE=0.8 iterated=0.2}
}
```

Note that this suffix codes not only for direction but also for the `once aspect` and a longer `duration`. Moreover, its probability distributions for these two features are more peaked than those of *thallu*, allowing it to override the values suggested by *thallu*.

Korean makes a similar distinction in its expressions for poking. *Gi ru da* refers to a single poke, while *maani gi ru da* refers to poking repeatedly. Again, an affix plays the role of overriding the `aspect` setting of the root verb. Japanese, too, makes this distinction in its poking verbs. In this case, the `force` parameter is also implicated. *Tsuku* refers to a single poke, which may use any amount of force. *Tsutsuku*, on the other hand, codes for repeated poking but only with low force.

Directional coding in Spanish

Spanish has several verbs for tipping an object over. *Volcar* refers to tipping an object onto its side, while *volver* is used for tipping an object onto its back. Finally, *poner al reves* refers to tipping an object upside down. Our model's `direction` feature could be used to distinguish the first two verbs, if objects were always placed in a canonical position.

However, in order to capture the full generality of these verbs, the model would need to be augmented with a notion of deixis. In other words, it would need a mechanism for determining the appropriate point of view for determining the "back" *vs.* the "side" *vs.* the "top" of an object.

Some generalizations in Tamil and Arabic

Many of the examples presented thus far code for rather specific actions. But there are other verbs which refer to broad classes of actions. Interestingly, even these very general verbs can differ significantly from the general verbs of English (such as *move*).

One such example is the Tamil verb *pudi*. This verb covers catching, holding and restraining. It definitely codes for `high force`, and its connotation of restraint suggests a `zero acceleration`. Yet this verb can be used for carrying an object, so this is not quite right.

Another example is Arabic *erme al callem*, which can cover English *drop*, *throw*, *knock over* and *tip over*. All of these involve an object in free-fall, suggesting an important linking feature not presently in the model. More importantly, both dropping and knocking over an object are (or can be) unintentional actions. The current model, due to its focus upon actions which correspond to single, entire, intentionally-executed x-schemas, is not yet equipped to deal with such verbs.

8.4 Sensitivity to Parameters

The results reported above were obtained after a certain amount of tuning of the various parameters of our labelling, obeying and learning algorithms. Since robustness to such parameters is a desirable quality in any learning system, we review here the sensitivity of these various parameters on the above examples.

For review, all the parameters of our labelling, obeying and learning algorithms are summarized in Figure 6.3.

In practice, the $MinSetFeature$ has been the most troublesome. Recall that this parameter specifies the minimum peakedness a feature's probability distribution must have in order for the mode value of that distribution to be included in the prototype. If set

too high, statistically significant patterns may get omitted from the linking f-struct. Too low, and merely-loosely-correlated features may get set, which can potentially overspecify x-schema parameters to the point where successful execution is impossible. The problem is partly due to the peakedness metric's failure to take into account the number of possible values for a feature: if the mode and runner-up have nearly equal probability (say 0.4 and 0.35), this ought to generate a very low peakedness for a feature with three possible values, but a fairly high peakedness if the feature has 10 values. In our experiments we have erred on the side of a high setting for $MinSetFeature$ to be sure to exclude weakly correlated features. As a result, the latter case presents problems. For instance: if all examples of *lift* exhibit `force`s of 1 or 2 (roughly similar number of each), while examples of *heave* exhibit `force`s of 4 or 5 (also roughly similar numbers of both), then we would like to have this significant correlation reflected in the linking f-struct when obeying commands. However, the peakedness measure will return a low value in each case and so the `force` feature will not be set. We have considered other peakedness measures but they have had their own problems.

The $MinMerge$ parameter is used to cut off merging when the best candidate merge consists of two senses of insufficient similarity (according to the heuristic described in §6.2.4). Since the similarity metric is sensitive to the peakedness of the probability distributions, so is $MinMerge$. While this is largely appropriate, it does lead to a certain amount of fiddling which is dependent upon the number of identical training examples in the training set (varied with the $TrainingPasses$ parameter): the more there are, the more peaked the distributions will become before non-trivial merges are considered, and thus the less likely they are to be judged similar enough to merge. Moreover, the values in the *virt* table are also implicated in the peakedness of probability distributions. The interdependence of all these algorithm parameters makes fine-tuning difficult.

The $MinLabel$ parameter has caused few problems, but this is partly an artifact of the type of training that has been performed. Most training has been with a single slot, using recognition test examples which do in fact have a reasonable label. Under these conditions any adequately low setting of the $MinLabel$ parameter produces acceptable results.

Another class of parameters is the initial settings of the *virt* table containing the number of virtual samples to use in probability distributions for each linking feature in a new word sense. Since in most languages there is a tendency for verbs to code for the

`schema` feature (more so than for the various features encoding x-schema parameters) it has proven useful to begin training with a smaller virtual sample count for that feature than for the others.

Finally, there is the question of sensitivity to the contents of the training set. In other words, for the modest-sized training sets which we have been able to work with, to what extent do variations affect the resulting lexicon? One observed effect is that it is important to "shuffle" the training data to ensure that the training and test sets end up containing examples that were labelled at a variety of times during the several-hour session with the informant. The labelling process itself seems to induce refinements of informants' ideas about these action verbs and thus they may label differently toward the end of the session than toward the beginning. For the labelling done by the author, an improvement of approximately 10% in recognition testing was made by this kind of shuffling.

8.5 Unlearnable Categories

Now that we've seen a range of word meanings capturable by our model, it's worth pausing to review the kinds of concepts which are *not* learnable. Unlearnable concepts fall into three categories.

The first type of unlearnable concept is one that depends on mechanical details of actions which are not representable at the schema level. For example, the SLIDE x-schema involves parallel invocation of an arm movement with a hand preshaping. The schema then specifies that grasping should not occur until both these actions are completed. But there is no way to specify exactly how these two actions should interrelate. As a result, if some language were to make a linguistic distinction based on whether the preshaping occurs toward the beginning of the arm motion *vs.* toward the end of the arm motion, our model would fail to capture it. If such distinctions were to occur frequently, it would be a rather fundamental type of failure, since it would argue against the appropriateness of our x-schema representation as a basis for the semantics of action terms.

A second type of unlearnable concept is one whose basis *is* present in the x-schema level, but which cannot be represented in word senses because the linking feature structure does not contain the appropriate features. For example, our model includes an `aspect` feature with values `once` and `iterated` which are set according to whether the x-schema

traverses a loop. We can imagine that some language might make more refined aspectual distinctions, such as distinguishing a few repetitions from many repetitions. Our linking feature interface would not be able to reflect this distinction. Part of the scientific aim of this project has been to determine the set of relevant features, so such discoveries are significant. However, such modifications to the set of linking features do not invalidate our model, and indeed are inevitable when surveying more languages (or when expanding the x-schema set to cover a wider range of actions).

A third type of concept is not strictly unlearnable, but instead is difficult to learn, because the resulting representation is unnatural (i.e. has a low prior). These concepts are those in which there are strong dependencies amongst the features, so that when expressed in disjunctive normal form there are a large number of disjuncts (i.e. word senses). For example, a word which codes for `force = high` when `elbow = extend` but `force = low` when `elbow = flex` cannot be represented in a single word sense. It is always possible to remedy this problem by introducing a new feature which expresses the appropriate abstraction. But there is a cost, because adding features exacerbates the "relevance problem" as described in §4.2.4. Fortunately, these cases so far seem rare. One example is the Russian suffix *-at*, which is an imperfective marker. Imperfectiveness can map to `long duration` when the `aspect` is `once`, or it can map to `iterated aspect` when the `duration` is `short`. Thus, this suffix is modelled using multiple senses.

8.6 Shortcomings

In recognition testing, many of the apparent errors encountered are in fact cases in which a relatively specific label is emitted but the informant had supplied a relatively general label. In other words, the "error" is simply a matter of multiple applicable labels and human labellers might exhibit the same behavior. In these cases it is necessary to inspect the posterior probabilities of all word senses to determine whether the general label also receives high probability. In most cases it does, although a quantitative analysis has not been made.

Another type of recognition error often encountered is the emission of a common word when a less common (and usually more specific) word is desired. This happens when the desired word is so uncommon in the training set that even its high likelihood of gen-

erating the linking f-struct is not enough for it to win over the more common word. The problem can be exacerbated if the desired word is so uncommon that it doesn't even have enough training examples to generate a nice peaked distribution, in which case it may not even generate a particularly high likelihood when it sees its prototypical case. One could attempt to address this problem by partially smoothing out the strong relative frequencies observed in the training data by using some sort of damping function on the occurrence counts. Indeed, it is psychologically plausible that children place low significance on relative frequencies, especially once the words in question have been learned well (Dan Slobin, personal communication).

Sometimes the model will learn multiple senses where we might expect only one. Multiple senses are required when two features can each take on a range of values but they are strongly correlated. In this case, the correlation can be captured only by multiple senses, e.g. one sense with `posture = palm` and `size = large` and another sense with `posture = pinch` and `size = small`. Intuitively, we would prefer a single sense which simply encodes "use a posture appropriate for the object size". But such a concept is beyond the power of our representation.

Due to the vagaries of the non-backtracking search procedure—especially if the $MinMerge$ parameter is set too high—training can occasionally produce more word senses than desired. An excessive number of senses is unappealing not only because it is counter-intuitive. It can also cause failure to label legitimate occurrences of the verb. The reason is that, since the overall frequency of the verb is "spread out" over its many senses, each individual sense has a low frequency count and is therefore at a disadvantage when competing against senses of other verbs. "Divided we fall..."

Chapter 9

Final Thoughts

"People are more than curious about language; they are passionate. ... Chances are you would never have made it to the last chapter of a book about the human hand."
—Steven Pinker

This final chapter takes stock of the model we've developed and suggests some new avenues of inquiry.

9.1 Summary

This dissertation has explored the hypothesis that in order to explain acquisition and use of action verbs, motor control must be integrated with language. We have presented

a model of lexical semantics for hand actions, which is embodied in the sense that it is intimately connected to a model of how those actions are actually controlled. We have provided an account of how such a semantics might be learned within a connectionist framework. The model was developed by building a system which, from a set of (action,verb) pairings from any of a variety of natural languages, learned to both label novel actions and obey verbal commands. Use of an active, Petri net representation called *executing schemas* proved essential for controlling actions. A hardwired mechanism for extracting a set of special *linking features* was then employed to provide the semantic building blocks. These linking features involved hand posture, joint motions, force and aspect. This traditional featural representation facilitated use of a Bayesian probabilistic learning algorithm, *model merging*, which displayed a number of desirable properties, including rapid learning of plausible word meanings and learning of an appropriate number of separate word senses. A moderate-size English vocabulary was learned, as well as some interesting distinctions from a handful of other languages.

Despite its successes, it's important to emphasize that the model is still quite provisional and no *strong* claims are being made about its cognitive validity. The isolation of verb learning from other cognitive activities is artificial; the learning results are limited; and indeed it is quite possible that the known constraints from computer science, linguistics and psychology aren't sufficient to converge upon the "correct" model of verb learning at the current time. Nevertheless, I feel the model does have something to offer to the study of cognitive science. The following sections attempt to place the model into this broader scientific context.

9.2 Contributions

9.2.1 Computer science

In the artificial intelligence literature there has been a longstanding divide between declarative and procedural representations. Declarative representations do not specify the algorithms which operate over them, imbuing them with a certain flexibility and the ability to focus on what is true rather than on how to reason. Yet with this flexibility can come inefficiency. Procedural representations solve the efficiency problem by directly hard-coding

the solution to a given task, but when interfaced with a declarative system, they become unanalyzable "black boxes," and hence have tended to be used only for simple functions which can be treated as atomic within the overall declarative framework. This thesis has investigated the use of a restrictive formalism for procedures—Petri nets—and has shown how it can facilitate the definition of a featural interface between such nets and more traditional declarative representations.

The thesis has also demonstrated how learning can be performed in a connectionist framework yet, unlike most neural network algorithms in the style of backpropagation, still retain the very valuable property of bidirectionality. Learning of bidirectional maps is crucial for tasks in which the resulting concepts must support not only recognition but also reasoning, acting or imagery.

The thesis is also an example of learning in the face of no negative evidence. Previous work on this problem has focused on rule induction such as grammar learning. The use of a Bayesian approach to lexical acquisition has shown that these techniques can also be effective in the semantic domain where the structures to be learned are not rule-like.

9.2.2 Cognitive modelling

Despite the successes of our model at its assigned task, we must be careful not to jump to the conclusion that we've discovered how children learn these action verbs. For one thing, it's perfectly possible that a completely different approach might work equally well. Even more importantly, we can't be sure that the task we sectioned off for ourselves corresponds to a "module" in children's overall learning strategy. In other words, we don't even know that we have considered the right input/output for verb learning, never mind the right algorithms!

Our hope is simply that having taken into account many of the known constraints, our model provides insights into these psychological processes. In turn, we hope this computational insight will spur further empirical research, so that future models will converge on the truth. In that spirit, then, let's look at some predictions suggested by the model, and consider how it may open new avenues of inquiry into semantics.

Most fundamentally, we hope we have presented a plausible account of how semantics for action-oriented words may be bodily grounded via a connection to the mo-

tor synergies which drive behavior. An important implication of the model is that x-schema internals—such as stretch receptor activation levels or detailed patterns of muscle contractions—are not available at the linguistic level, leading to the prediction that no language will contain verbs referring to these details. In other words, the model predicts that coordination and parameterization are the appropriate level for linguistic access to motor control.

The model, by showing the importance of motor control to semantics, also serves to partially explain the results of Huttenlocher *et al.* (1983) showing children's early tendency not to generalize action verbs beyond their own activity. If, for example, the *visual* effects of actions were primary, one might expect the child not to differentiate so strongly between his own and his parent's actions, since both are observed via the same visual mechanism.

Our use of adjustable priors over which features are likely to be relevant in each part of a verb complex predicts that children should be slower to learn words which violate the patterns represented by those priors. Choi & Bowerman (1991) report such results for Korean *vs.* English. The model predicts that this effect should become more pronounced as the vocabulary expands and these priors become stronger.

Our category representation comes down strongly on the side of richly detailed, embodied, gestalt-like prototypes, with generalization achieved largely through the use of multiple such prototypes and graded boundaries. Having made this perspective on categorization computationally explicit ought to facilitate fair comparison of this approach with the more traditional necessary-and-sufficient-conditions approach.

Predictions, though, are not the only contribution to cognitive science. It is my hope that this work has suggested a novel approach to semantic analysis which may prove fruitful in some circumstances. For example, in the course of building the model the `elbow` feature (which encodes whether the elbow joint flexes or extends during motion of the arm) turned out to be useful for discriminating *push*es and *pulls*. It was arrived at by considering the x-schematic grounding of those verbs, and solves some difficulties with previous analyses involving solely external properties such as center of mass and direction of motion. More generally, it is hoped that the model offers a concrete scientific language in which one can express neurally plausible representations of events—including motor actions but also less bodily grounded kinds of events—in a way which permits analysis, comparison, and integration into models of learning and processing.

9.3 Some Objections Considered

Many potential objections have been considered in the preceding chapters. This section briefly considers a few remaining issues.

- What if some languages code for smooth (or finer-grained) coordination of synergies rather than just the discrete kinds of coordination expressible in Petri nets?

  The current x-schema formalism can partially accomplish smooth coordination by introducing rate parameters. Then, two synergies can be smoothly coordinated by passing this rate parameter to both of them. Fine-grained coordination can always be implemented by introducing new (and otherwise unmotivated) triggering conditions. For example, if it proves important to capture the idea of preshaping the hand exactly when it has travelled halfway to the target object, one could add a new place to the Petri net which receives a token when the arm has travelled halfway. This place would then serve as the precondition for the preshape synergy. Both of these techniques are ad hoc. If they should prove to be necessary very often, it would be a sign that the underlying formalism should be changed.

- Isn't model merging a greedy algorithm subject to local minima?

  Yes. The effect is mitigated, however, by the nature of the merging approach. Since it is possible to accumulate a number of training examples before merging occurs, the likelihood of making premature commitments is reduced. Of course, storing these examples requires memory. Fortunately, in practice, collecting even relatively small numbers of examples (e.g. a dozen) between merging episodes seems to overcome local minima, and so we needn't posit cognitively implausible memory capacity in order to avoid local minima.

  Nevertheless, it would be trivial to add backtracking to the merging process. But this would come at quite a price in terms of runtime complexity, since the algorithm would lose its monotonic character.

- Why should words be represented by distinct senses as opposed to some sort of continuum?

  For one thing, Lakoff (1987) convincingly shows how categories exhibit structure, including multiple senses and the relations between them. However, since I haven't

modelled such inter-sense relations in this thesis, the question remains. The answer, I believe, is that separate representations of each sense is one of only two ways to capture multiple sets of correlations amongst features in a way which allows retrieval of the features given the category. The other way is to use a basin-of-attraction approach as seen with Hopfield nets (Hopfield 1982). However, such a net would require iterative activity in order to "settle" upon a prototype, which is probably too slow to be cognitively plausible.

- Isn't there more to action verbs than motor control?

  To more fully capture the (concrete) uses of these words, I think the next important step is to look at planning, e.g. Levison (1995). It is possible that higher-level x-schemas can be used to seamlessly integrate motor control with some sort of reactive planner, but I have done no work in this direction.

- What about qualia, i.e. what it "feels like" to push something?

  The model offers no clues on this perplexing philosophical problem.

9.4 New Questions

As any cognitive model should, our verb acquisition model suggests a number of questions for cognitive science. Some important ones have been discussed already, including: how to explicitly represent the image-schematic transformations which link the multiple prototypes of a radial category (§5.5.2), how to improve learning by recognizing contrast sets within the lexicon (§6.5), and how to extend the model to learn constructions (§7.4). This section discusses some others. I don't have the answers to these questions—I want to convince others to go and find them!

9.4.1 Classifiers

Let's start with a fairly specific but intriguing issue. We have seen how hand posture has proven to be an important determinant in a number of verbs and how it relates to x-schema activity. But hand posture (and motor behavior more generally) might have implications beyond verbs. In particular, many languages employ classifiers which must be

attached to nouns. A small number of classifiers are used for all objects. Generally, linguists have sought to characterize the semantics of object classifiers in terms of geometry, such as "long thin object". The question then arises, why those geometric abstractions? Perhaps the choice of abstractions can be better understood by examining how classifiers correlate with the motor activities involved in manipulating the objects. Perhaps "long thin object" should be better thought of as "objects held with a prismatic grasp"; other classifiers may correspond to power grasps *vs.* precision grasps. Even if this perspective does not lead to any cleaner a description of the classifier categories, it could serve as an explanation for them.

9.4.2 Reversatives

Reversatives present an intriguing challenge to the model. Consider the English prefix *un-*. Why can we form verbs like *unzip* and *unbutton*, while others like *unpush* or *ungrasp* are impossible? Perhaps some kind of account could be developed in terms of x-schemas.

One such hypothesis would be that the semantics of *un-* fundamentally involves the reversal of an x-schema which involves attainment of a goal state. Put another way, only resultative root verbs can take the *un-* prefix. Exploring the meaning of this claim in terms of our model would involve formulating a precise definition of resultatives in terms of patterns of x-schema execution. The abstract x-schematic aspectual model of Narayanan (1996)—which includes a state for achievement of a result—may be a starting point for such an analysis.

To carry this idea forward (pun intended), one would need to build x-schemas which allow bidirectional execution, which might well require extensions to the current formalism. Yet it seems to be within the spirit of the representation. First, though, it would be prudent to investigate how reversatives work in other languages. In the West African language Wolof, for instance, there is a common reversative suffix which seems to be more productive than English *un-*. For example, the suffix is applied to the "put" verb (*teg*) to form the "get" verb (*tiggi*) (Kevin Moore, personal communication).

9.4.3 Speech acts

Another area of inquiry involves speech acts. To make our task feasible we have simplified reality in many ways. In focusing on what kinds of actions a verb can refer to, we've ignored *why* the child might choose to communicate something about his actions at all. It is unclear how the model would change if we were to take into account motivations to communicate. For example, perhaps words uttered before carrying out an action are intended to declare goals, while verbs uttered during an action tend to comment on manner. If so, this fact is a candidate for pre-wiring into the model to aid learning. Another example: the present model has no built-in knowledge of the Gricean principle that one generally aims to communicate information which is unlikely to be shared by the listener (Grice 1975). Incorporating expectations about the listener's knowledge would affect the LABEL algorithm in ways which would be interesting to investigate.

9.4.4 Probabilistic linking f-structs

As discussed in §8.6, a verb may correlate with a given feature without necessarily specifying only one possible value for that feature. For example, *lift* may encode `force = low or med` while *heave* may encode `force = med or high`—clearly the verbs differ on this feature. Yet, since neither verb codes strongly for a *single* `force` value, it's possible that `force` will be left unset when the system obeys a *lift* or *heave* command. The difficulty lies in the fact that in the current model, each feature in the linking f-struct can be set to only one value, and so obeying a command requires making an all-or-none decision whether to set each feature. This decision is based on whether the word sense codes for the feature with a peakedness exceeding the threshold $MinSetFeature$. As a result, the model is very sensitive to this threshold, often erring on one side or the other. Sometimes it leaves features unset, ignoring significant correlations. Other times, it sets too many non-obligatory features, overly constraining x-schema execution, possibly to the point of causing execution failure.

The model could therefore be improved by allowing the linking f-struct—the sole interface between language and action—to be probabilistic itself. By doing so, the all-or-none decision described above would be unnecessary. Furthermore, x-schemas would have an indication of the strength of commitment to the given feature settings required by the command. So, for the above example, when the LIFT x-schema discovers the object is too

heavy to lift with `force = low`, it would know whether and how much it could increase the force and still comply with the command.

9.4.5 X-schema learning

Let's turn now to a larger issue: x-schema learning. Our model has assumed that a set of x-schemas exists prior to verb acquisition and does not change during acquisition. Now one obvious avenue of inquiry is how x-schemas are acquired in the first place. While some simple ones are probably innate, others are obviously learned from experience. The appropriate learning paradigm is probably reinforcement learning, except for those cases where a parent shows the child the reasons for his failures. The acquisition of stateful action policies like x-schemas—rather than simple world-state-to-action maps—is as yet an unsolved problem.

But another aspect of the x-schema learning story is how the learning process might interact with language learning. First, one could try to model the effects of ongoing motor development during language learning (e.g. Gopnik (1981)). More speculatively, perhaps the very labelling of multiple types of actions (a push, a pull, a lift, etc.) with the verb *move* might trigger the creation of a new higher-level MOVE x-schema which selects one of the more specific x-schemas based on world state. Or, perhaps two x-schemas sharing the same label might be *merged* into one, thereby sharing substructures for handling contingencies, etc. In either case the result may be that the child may become more flexible in choosing actions even in purely non-linguistic settings. Such learning would be an incarnation of the (very controversial) Sapir-Whorf hypothesis (Whorf 1956).

9.4.6 Integrating x-schemas with image schemas

In Chapter 7 we presented a simple model capable of combining prepositions with verbs. In that chapter, the semantics of spatial terms were modelled using the same motoric features used for verbs. In some cases, such as *push left* or *pull up*, this proved adequate. However, spatial terms cannot always be reduced simply to motoric features. Work in cognitive linguistics (Lakoff 1987) points to the need for "image schemas" for representing spatial relationships. For example, to handle *push around* or *lift through*, we really need an image-schematic representation of the desired path of motion. Connectionist models of

image schemas (Regier 1996) differ from our x-schema representation substantially, and it is an open question how to combine them—that is, how to translate an image schema into the necessary motor parameters. More generally, it is unclear exactly what types of reasoning should be performed down at the schema level as opposed to up at the feature level where standard artificial intelligence techniques can be applied.

Chang (1997) has begun investigation into related issues in a study of the compositional semantics of aspect. Since its essence is event structure, aspect is quite amenable to an x-schematic analysis, and often the relevant x-schemas are contributed by verbs. But aspect depends on the arguments of verbs as well, and the semantic contribution of these arguments can be image-schematic. For example, compare *wash the cart*, which can be read as either perfective or imperfective, with *push the cart*, which is most naturally read as imperfective.[1] It is the details of how the PUSH and WASH x-schemas interact with the direct object *cart* that account for this difference. Namely, in the context of the PUSH x-schema, the cart does not furnish a goal location. But by appending an additional argument, as in *push the cart to the store*, a perfective reading can be obtained. Here the image-schematic content of the phrase *to the store* comes into play by suggesting a path from source to goal.

In general, image schemas must "unfold" in coordination with x-schema execution. Perhaps the solution lies in identifying *key events* in dynamic image schemas, such as "enter" or "cross". Then, dynamic image schemas could be modelled with Petri nets in which these key events correspond to transitions.

9.5 X-Schemas for Abstract Thought

Of all the domains of human experience, why have we chosen to investigate motor activity? The choice is not random. As was mentioned back in §2.2, Lakoff & Johnson (1980) have shown that embodied concepts very often provide the grounding for more abstract concepts, for example via conceptual metaphor. And so the hope is that our x-schema representation might ultimately explain much more than how people communicate about hand actions. Here are some examples of how.

[1] A standard test for the imperfective reading is the acceptability of appending *for an hour* to the phrase; similarly, appending *in an hour* tests for the perfective reading. Either is acceptable with *wash the cart*, but only *for an hour* works with *push the cart*.

Active representations like x-schemas, while possibly originating in very concrete motor activity, also have advantages for general knowledge representation, especially for reasoning about processes. Suppose that the child were to notice certain patterns in his motor schemas. He may then create a new, abstract x-schema which reflects this structure. Such an abstract x-schema would represent the notion that actions must be enabled, may be cancelled, usually proceed, are sometimes interrupted and sometimes complete. Narayanan (1996) has designed such an abstract x-schema and shown convincingly that the linguistic phenomenon of "aspect" (i.e. the distinction between *had fallen*, *has fallen*, *was falling*, etc.) can be explained in terms of the activity of such an x-schema.

But simple abstraction is only one way in which motor representations can be brought to bear on non-motor domains of reasoning. Metaphor is another. Consider the following English phrases, none of which refers to actual hand actions:

grasp an idea	get a grip on reality	grapple with a problem
hold that thought	slip my mind	just drop the idea
let the issue go	put someone down	hit on an idea
push an idea (drug pusher)	reach a conclusion	pull for a candidate
offhand remark	out of hand situation	pick up on a fact
gripping movie	it's a pushover	pressing business
pull off a stunt	carry on	heavy-handed tactics

Narayanan (1997) presents a model in which metaphorical expressions such as these are understood via conceptual *mappings* between a "target domain" (such as thinking or politicking) represented by features, and a "source domain" (motor actions) represented by x-schemas. In a nutshell, a metaphorical sentence is understood by invoking an appropriate metaphor, mentally executing the x-schema referred to by the metaphor, and then using the mapping again to propagate the results of the mental execution back into the target domain. Especially for complex process-oriented source domain mappings, the active nature of the x-schema representation can provide much more efficient reasoning than re-representing the x-schematic knowledge in a logical form in the target domain. The model is particularly convincing for the case of novel metaphors, which are often understood immediately, and for which one cannot argue for the existence of a separate target domain sense of the relevant words.

The Petri net formalism can prove a useful modelling tool even in domains where

one does not consider the eventual mapping down to motor activity. Dean Grannes has investigated using Petri nets to encode the semantics of the ditransitive construction, which connotes transfer. (For example, in *John baked Mary a cake*, it is the grammatical structure, not the verb *bake* or any of the other words, which indicates that the cake has been given to someone.) As argued in Goldberg (1995: Chapter 2), the ditransitive can be considered a radial category with separate senses for different (but related) types of transfer. Grannes used Petri net places to represent predicates involving possession, existence, intention, motion and obligation, and Petri net transitions to represent the modifications of these predicates coded for by various action verbs and by the different senses of the ditransitive construction itself. The result is a simple model which can determine the applicability and semantic entailments of a range of usages of the ditransitive with action verbs. In a similar vein Jonathan Segal has attempted to represent the semantics of modals (*help*, *hinder*, *let*, etc.) using a Petri net representation of force dynamics.

If x-schemas are ultimately to be linked up with logical reasoning, we'll need a neural account of how that can be done. At the core of logical reasoning lies the notion of variable binding, a traditionally difficult task for neural models. Shastri & Ajjanagadde (1993) has proposed a clever solution which relies upon temporal synchrony. In §3.3.2 we sketched out how this temporal binding mechanism can support an interface to x-schema execution, but more work needs to be done.

9.6 The Real World

This thesis has been an exercise in basic science. Yet when wrapping up a piece of work such as this, one can't help but think about how it might eventually benefit the "real world". I'll conclude, therefore, with a brief and very speculative peek into the crystal ball.

One area is robotics, where space and action take center stage. For instance, telerobotics, which traditionally involves controlling a remote robot via a visuo-motor loop, could benefit in some circumstances from a natural language interface because it allows a higher level of control. In a similar vein, programming industrial robots (e.g. for factory assembly) could become accessible to non-experts if it could be done in natural language, making flexibility more practical.

Looking much further out to the future, enormous medical benefits could result

from a better understanding of how cognition is realized in the brain. Models such as ours may serve to nudge the field of computational neuroscience in a direction leading toward an understanding of language learning in which, for example, an overgeneralization syndrome might suggest a drug treatment to adjust the level of a certain neurotransmitter. We'll just have to see.

Appendix A

Guide to the VerbLearn Software System

This appendix describes the software system (creatively named "VerbLearn") which instantiates the model developed in this thesis. It is intended primarily as an overview of the implementation work done in support of the dissertation. However, it also serves as a (rather brief) user's manual, and a guide to the source code which is an interesting case study in the use of object-oriented design. The learning code is written in the new object-oriented language Java, and is available to be inspected (or executed!) at the URL `http://www.icsi.berkeley.edu/~dbailey/verblearn`.

An important note: This appendix describes functionality involving the *Jack* simulator. While the VerbLearn system is in many ways "*Jack*-ready", the simulator has not yet been interfaced to the system. Therefore, the reader should be aware that, wherever *Jack* is involved, the described functionality is not yet available for use.

A.1 Data Files

The home directory of the VerbLearn system contains a number of subdirectories which hold the data files needed to run the system.

The `scenario/` subdirectory contains the specification of the collection of scenarios which drive training and testing. The top-level specification of scenarios is held in

`scenario/specs`. For each scenario which will be supported by the *Jack* simulator (called "grounded" scenarios), we specify the scenario name (arbitrary) followed by the name of a *Jack* environment file and an initial goal f-struct. The *Jack* environment files are held in the `scenario/env/` subdirectory. A collection of named initial goal f-structs is held in the `scenario/goals` file. In order to facilitate experimentation with the learning algorithm, the system allows specification of "ungrounded" scenarios in the `specs` file. These consist of the scenario name followed by two f-structs; the first specifies the initial world state, and the second specifies the final linking f-struct. Here's a sample scenario set specification file:

```
// Grounded scenarios:

gsc1   cube_hand_off        slideflex
gsc2   cube_hand_touching   slideflexslow
gsc3   button_hand_off      depress

// Ungrounded scenarios:

usc1   {size=large}   {size=large schema=slide posture=palm force=med}
usc2   {size=large}   {size=large schema=slide posture=palm force=high}
usc3   {size=large}   {size=large schema=slide posture=palm force=med}
usc4   {size=large}   {size=large schema=slide posture=palm force=high}
usc5   {size=small}   {size=small schema=slide posture=grasp force=med}
```

The structure of the *Jack* environment files is determined by the *Jack* system and is not discussed here. The `goals` file consists of a list of goal names (arbitrary), each followed by an f-struct which specifies a minimal set of features needed to generate an action when passed to *Jack*. It should therefore specify an x-schema name and any required parameters. Here is a sample goals file:

```
// Goals

slideflex       {schema=slide elbow=flex}
slideflexslow   {schema=slide elbow=flex force=low}
depress         {schema=depress}
```

When the VerbLearn system is run (see below), additional information about each grounded scenario will be generated, including the initial world-state f-struct and the final linking f-struct. The collection of these is stored in the files `scenario/initial` and `scenario/final`. Each file consists of a set of pairs, where each pair contains a scenario name and an f-struct. Here's a sample final linking f-struct file:

```
// Final links

sc1   {size=large schema=slide posture=palm force=med elbow=extend}
sc2   {size=large schema=slide posture=palm force=high elbow=extend}
sc3   {size=large schema=slide posture=palm force=med elbow=extend}
sc4   {size=large schema=slide posture=palm force=high elbow=extend}
sc5   {size=small schema=slide posture=grasp force=med elbow=extend}
sc6   {size=small schema=slide posture=grasp force=high elbow=extend}
```

A movie of the action is also created and stored in the file `scenario/movie/<`*scenario-name*`>`.

The `param/` directory holds a collection of parameter files, each of which specifies values for each of the algorithm parameters and lists the motor and world-state linking features to use for learning. Here is an example parameter file:

```
// Default parameters

MinLabel           0.01
MinExplain         0.5
MinInterpret       0.5
MinSetFeature      2.0
MinMerge           0
ModelPriorWeight   1.0
BatchSize          10
TrainingPasses     1
AdaptVirtuals      true
VirtualInertia     50
MaxVirtual         10
VerboseSenses      true
TestRecognition    false

MotorFeatures {
     schema        0.05  {slide lift depress}
     posture       1     {grasp palm wrap}
     elbow         1     {flex extend fixed}
     force         1     {low med high}
     aspect        1.5   {once twice many}
}
WorldFeatures {
     size          1     {small med large}
     contact       1     {true false}
}
```

A few further notes on this file. First, `VerboseSenses` and `TestRecognition` are not true algorithm parameters, but are simply software control switches. `VerboseSenses` controls how word senses are printed during training and inspection of the lexicon. If `true`, full probability distributions are shown; if `false`, the system shows just the (non-probabilistic) f-struct containing the mode value for each feature (so long as its probability

exceeds `MinSetFeature`). `TestRecognition`, if set to `true`, will perform a recognition test and report the percent correct after every merge during training. Next, in the list of motor and world-state linking features, note the numbers listed between each feature name and its list of possible values. These are the number of virtual samples to use for each possible value in the feature's probability distribution, when creating a new word sense. Lastly, note that you cannot usefully add novel features to the model by merely inserting them into this file. The appropriate changes would need to be made to the x-schema code to utilize (and set) the new features.

The full set of scenarios can be divided into training, recognition-test, and obey-test sets in different ways. This is done by creating multiple "dataset files" in the `dataset/` directory. It is also legal for a dataset to omit some scenarios entirely. Here is a sample dataset file:

```
// Dataset

train      {sc1 sc2 sc3 sc4 sc5 sc6 sc7 sc8 sc9 sc10}
recognize  {sc11 sc12 sc13 sc14}
obey       {sc15 sc16}
```

A.2 Running the System

A.2.1 The main program

The VerbLearn system may be run as a Java applet over the Internet via the URL mentioned earlier. Or, if you have your own local copy of the system, you may run it by setting your current directory to the `java/class/` subdirectory and then invoking the Java interpreter with the classname `VerbLearn` as its only argument.

Once running, VerbLearn presents the user with a HyperCard-like interface; i.e. it consists of a number of windows only one of which is shown at a time and which are selectable by a row of buttons across the top. We will now go through the windows and their functionality.

The Generate Window is used when changes have been made in the specification of the scenario set. It is used to request *Jack* execution of some of the defined scenarios (by default just the unexecuted ones, but you may choose any or all of the *Jack*-grounded scenarios if you like). When such an execution is performed, the initial *Jack* environment

and the goal f-struct are read and passed to *Jack*. *Jack* then computes the initial world-state linking f-struct, carries out the specified action, and finally returns the initial world-state f-struct, the final full linking f-struct, and a movie file and these are added to the scenario set. Once this procedure has been applied to a given scenario, it may then be used for training or testing the model.

The Label Window is used to collect labels for scenarios for a given language. In this window one first indicates which language to collect labels for, and then which scenarios to label (by default all the unlabelled ones, but you may choose any or all of the executed or ungrounded scenarios if you like). During labelling, each scenario is dealt with in turn, by showing either its movie (if grounded) or its final linking f-struct (if ungrounded). A pop-up window then prompts for a label. Multi-word labels are specified with a dash separating the words, and these dashes are required even if some of the slots are to be left unlabelled.

Finally, with a collection of executed and labelled scenarios, we may proceed to the Training Window. Here, we specify a language along with a parameter file and dataset file. Also we specify an initial lexicon to use, normally "Empty" unless you wish to modify a previously trained lexicon. The results of training are shown in the log subwindow, which may be saved to disk. The current parameters and dataset may be inspected or modified by clicking on the corresponding "View/Edit" buttons. If retraining, be sure to reset the lexicon by again choosing the "Empty" lexicon setting.

Once training is complete, the resulting lexicon may be inspected in the Lexicon Inspection Window. There, the upper subwindow shows the virtual sample values for each slot (only interesting if adaptation of the virtual samples was turned on during training). The lower subwindow shows the collection of senses for any word in the lexicon, by clicking on the button for that word. The buttons are arranged in separate columns for each slot in the lexicon. The "Change format" button can be used to switch between viewing full probability distributions or simple f-structs for each word sense. This window is shown in Figure A.1.

The Recognize Window is used to test the lexicon against the recognition test set defined by the dataset chosen on the Training Window. For each test case, the desired and actual labels are shown in the log window. Errors are flagged and the ratio of correct cases is shown. Optionally you may choose to test recognition on a selected set of scenarios and you will then receive a more detailed description of the labelling process for each one,

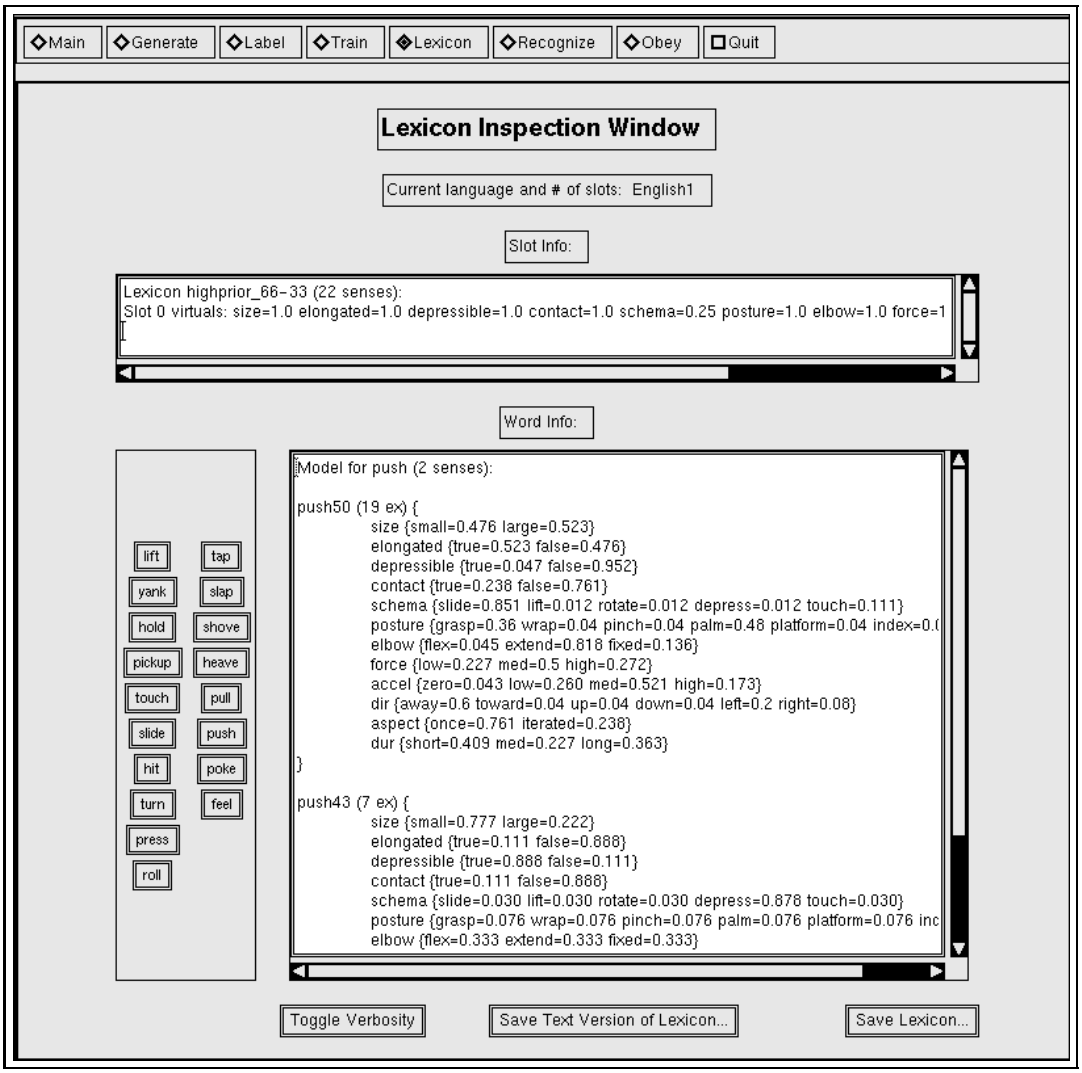


Figure A.1: The VerbLearn software system, showing the Lexicon Inspection Window.

including the posterior probabilities of each word sense in the lexicon.

Finally, the Obey Window is used to test the lexicon's ability to obey verbal commands. In one mode, it allows you to go through the current dataset's obey test set, evaluating each resulting linking f-struct by pushing it back through the recognizer and comparing the emitted label to the original command. However, you may also choose to have the *Jack* simulator animate the action resulting from the linking f-struct by selecting an individual scenario and entering a command using the buttons in the upper right subwindow.

A.2.2 The scenario generator program

As described in §8.1, the set of scenarios used by the VerbLearn system can be generated by a separate Generator program. This program can be invoked by setting your current directory to the `java/class/` subdirectory and invoking the Java interpreter with the class name `Generator`, an initial scenario number, the total number of random scenarios desired, and an optional constraint on the generated scenarios expressed as an f-struct of the form `'feature1=value1 feature2=value2'`. For example, the command

```
java Generator 100 10 'schema=slide'
```

would create 10 scenarios, named `sc100` through `sc109`, all of which involve the SLIDE x-schema.

The scenario specifications are printed to the standard output (i.e. the screen). In normal use, this output should be redirected to a file. This file can then replace, or be appended to, the `scenario/specs` file.

A.3 Code Overview

This section provides a brief overview of the implementation of the VerbLearn system. The linguistic component of the model—feature structures and model merging—is coded in Java, and is described first. The partially-completed *Jack* setup to implement x-schemas is then described.

A.3.1 Java Code

The Java source code for the VerbLearn system is contained in the `java/` subdirectory of the system, in three key files: `Lexicon.java`, `Context.java` and `VerbLearn.java`.

`Lexicon.java` contains the core algorithms of the model. They are organized in a hierarchy of classes which reflects the structure of the model's representations. At the top level we have the `Lexicon` class, which supports the incorporation of new training instances, the labelling of f-structs, and the interpretation of commands. It is essentially composed of an array of `Slot` objects, and its routines for the most part are concerned with integrating information across slots. For example, this class is responsible for assembling a multi-word label by filling the best slot first, and is responsible for resolving conflicts between words in a multi-word command. The `Slot` class implements the same three functions. A `Slot` contains a set of `Word` objects as well as a table with the number of virtual samples to use for each feature in a new word sense (and so the code for adapting the virtual samples to reflect already-learned words is found here). A `Word` object contains a collection of `Sense` objects and the major part of the model merging algorithm therefore occurs in this class. It is necessary for this class to keep track of all training examples ascribed to the word since it must compute the dataset likelihood during merging. While the `Word` class supports both labelling and obeying, it does so with a single routine which takes an `Fstruct` and returns a tuple containing the best-matching `Sense` along with the probability of the `Fstruct` given that `Sense`. During labelling, this routine is given the final linking `Fstruct` and we care only about the resulting probability. During obeying, it is given the initial world-state `Fstruct` and it then essentially returns the most compatible `Sense`. The `Sense` class represents a probabilistic feature structure. It has routines for creating initial senses, merging and measuring similarity to other `Sense`s, but these routines are essentially loops over the `Dist` objects which represent the probability distributions for each feature. This `Dist` class implements arbitrary discrete distributions by keeping frequency counts of possible values as well as "virtual" counts. It implements measures of peakedness as well as similarity to other `Dist`s, as well as merging by summing counts. Finally, the `Fstruct` class implements non-probabilistic feature structures and its only non-trivial routine is to create an `Fstruct` from an array of `Sense`s, which is used in multi-word obeying.

As you can see, each of the algorithms presented in Chapter 5 and Chapter 6 are

in fact distributed over several classes. While this has an obvious downside (especially when trying to present the algorithms in a non-object-oriented manner as done in those chapters), it has proven quite natural for iterative refinement of the code, since all the routines that operate at a given "level" are grouped together. For example, extending the model to handle multiple slots involved only writing the `Slot` class and changing the Lexicon class to handle an array of slots. Similarly, adding new probability distribution types (such as Gaussians) would involve little more than adding new classes with the same interface as `Dist`.

`Context.java` implements classes which support the core classes' need for data management. Most important is the `Context` object itself, which holds pointers to the scenario set, current language, number of slots, `Params`, `Dataset` and `Lexicon`, and ensures that they (and their GUI manifestations) are kept in synchronization. The `Scenario` class holds the various parts of a scenario as discussed in §A.1 and offers routines to determine the scenario type (grounded, unlabelled, etc.). The collection of them is managed by the `ScenarioSet`, which is largely concerned with disk file reading and writing. The `Param` class manages algorithm parameters, their modification and reading/writing them to disk. `Dataset` does similarly for the train and test sets. Finally, several smaller classes (`VerbComplex`, `CandidateSense` and `CandidateMerge`) are essentially just strongly typed tuples used in the core classes.

`VerbLearn.java` assembles all the above into a working application program or applet. The `VerbLearn` class provides the entry point for both the application and applet methods of invoking the system, sets up the GUI generally, and holds the `Context` as a static variable for easy access by all other classes in the system. Separate classes then are employed to set up each of the windows of the program, and these classes hold the top-level loops for each of the major functions of the system. For example, the `TrainPanel` class contains a `train()` routine which loops through the training set instructing the `Lexicon` to incorporate each example. Lastly, some of the GUI components repeated on several windows are implemented as separate classes, such as the `LangChoice` component.

In addition to these files, `Jack.java` contains several utility classes to allow a VerbLearn application running on one machine to invoke and interact with a *Jack* process on another machine. This involves a socket connection with a protocol for atomic transmittal of strings. Finally, `Utilities.java` defines a number of general utility classes—the Java library is still very young and sometimes needs a little help.

The Generator program is contained in the file `Generator.java`, which defines the single class `Generator`. The code in this class reads the parameter file `param/default` to determine the set of linking features and their possible values. Random f-structs are then generated. The most important function in this class is `fixOrReject()`, which is responsible for reducing the chances of emitting nonsensical f-structs. This is accomplished through a combination of: modifications of feature values; unsetting of features; and outright rejection of some f-structs. This routine is the only place in the Java code of the VerbLearn system which "knows" about the meanings of the linking features. For instance, `fixOrReject()` will reject an f-struct which specifies a `high force` and a `small size` object, yet only a `low acceleration`. Note that if new features are added to the system by inserting them into the `param/default` file, you may need to modify this routine to ensure that the new features interact sensibly with the previously existing features when generating your new scenarios.

A.3.2 *Jack* Lisp Code

Schemas are implemented in Lisp, because this allows them to utilize some code prepackaged with the *Jack* system which facilitates coordination of parallel and sequential actions. These so-called "parallel transition networks" are related to Petri nets although more powerful in general. The code for x-schemas is found in the `schema/` directory.

Some support code for executing x-schemas in *Jack* is contained in the `jack/` subdirectory of the VerbLearn home directory. Some of these are `.jcl` ("*Jack* Command Language") files for initializing the system, while others are `.fig` files which define the objects used in our environment (such as the cube).

While the actual implementation of x-schemas in *Jack* Lisp does not precisely follow the Petri net structure, an earlier implementation of x-schemas in the parallel language pSather (Stoutamire 1995) did follow the formalism more closely. Places and transitions were each implemented as a class. Each primitive synergy was encapsulated in its own class, as a subtype of Action, and each transition contained such an object. Parallelism was achieved by running each transition as a separate thread, whose basic behavior was to block until a token was present at each input place, then remove the tokens, tell its Action object to execute, and repeat. While the pSather language already had facilities for block-

ing on conditions such as presence of a token, it was necessary to extend the language to allow blocking atomically on an *array* of such conditions, since transitions have a variable number of inputs not known at compile time. While the parallel implementation allowed clean expression of concurrency, it did render "simple" chores like resetting an x-schema quite challenging.

Bibliography

Agre, P., & D. Chapman. 1987. Pengi: An implementation of a theory of activity. In *Proc. AAAI–87*, 268–272, Seattle, WA.

Arbib, Michael A., Thea Iberall, & Damian Lyons. 1987. Schemas that integrate vision and touch for hand control. In *Vision, Brain and Cooperative Computation*, ed. by Michael A. Arbib & Allen R. Hanson. Cambridge, MA: MIT Press.

Badler, Norman I., Cary B. Phillips, & Bonnie Lynn Webber. 1993. *Simulating Humans*. New York: Oxford University Press.

Bailey, David R., 1992. Inference through imagery: Understanding simple sentence sequences in L_0. Unpublished manuscript.

Ballard, Dana H., Mary M. Hayhoe, Polly K. Pook, & Rajesh P. N. Rao. 1996. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences* (to appear).

Berlin, Brent, & Paul Kay. 1969. *Basic Color Terms: Their Universality and Evolution*. Berkeley, CA: University of California Press.

Bernstein, N. A. 1967. *The Co-ordination and Regulation of Movement*. New York: Pergamon Press.

Brooks, Rodney A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* .

Brown, Roger. 1973. *A First Language: The Early Stages*. Cambridge, MA: Harvard University Press.

CAREY, SUSAN E. 1978. The child as word learner. In *Linguistic Theory and Psychological Reality*, ed. by Morris Halle, Joan Bresnan, & George A. Miller. Cambridge, MA: MIT Press.

CHANG, NANCY. 1997. A motor- and image-schematic analysis of aspectual composition. Presented at the 5th International Cognitive Linguistics Association Conference, Amsterdam. Available as International Computer Science Institute Technical Report TR-97-034.

CHARNIAK, EUGENE. 1993. *Statistical Language Learning*. Cambridge, MA: MIT Press.

CHOI, SOONJA, & MELISSA BOWERMAN. 1991. Learning to express motion events in English and Korean: The influence of language-specific lexicalization patterns. In *Lexical and Conceptual Semantics*, ed. by Beth Levin & Steven Pinker, 83–122. Amsterdam: Elsevier Science Publishers.

CLARK, EVE V. 1987. The principle of contrast: A constraint on language acquisition. In *Mechanisms of Language Acquisition*, ed. by Brian MacWhinney. Hillsdale, NJ: Erlbaum.

CRANGLE, COLLEEN E., & PATRICK SUPPES. 1994. *Language and Learning for Robots*. Stanford, CA: Center for the Study of Language and Information. Distributed by Cambridge University Press.

CUTKOSKY, MARK R., & ROBERT D. HOWE. 1990. Human grasp choice and robotic grasp analysis. In *Dextrous Robot Hands*, ed. by Subramanian T. Venkataraman & Thea Iberall, chapter 1. Springer-Verlag.

DAMASIO, ANTONIO R., & DANIEL TRANEL. 1993. Nouns and verbs are retrieved with differently distributed neural systems. *Proceedings of the National Academy of Sciences* 90, 4757–4760.

DEMPSTER, A. P., N. M. LAIRD, & D. B. RUBIN. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 34, 1–38.

DIEDERICH, JOACHIM. 1988. Knowledge-intensive recruitment learning. Technical Report TR-88-010, International Computer Science Institute, Berkeley, CA.

DRESCHER, GARY L. 1991. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press.

ELMAN, JEFFREY L. 1991. Incremental learning, or the importance of starting small. Technical Report 9101, Center for Research in Language, University of California, San Diego.

FAUCONNIER, GILLES. 1985. *Mental Spaces: Aspects of Meaning Construction in Natural Language*. Cambridge, MA: MIT Press.

FELDMAN, JEROME A. 1982. Dynamic connections in neural networks. *Biological Cybernetics* 46, 27–39.

——, & DANA BALLARD. 1982. Connectionist models and their properties. *Cognitive Science* 6, 205–254.

——, GEORGE LAKOFF, DAVID R. BAILEY, SRINI NARAYANAN, TERRY REGIER, & ANDREAS STOLCKE. 1996. L_0—the first five years of an automated language acquisition project. *AI Review* 10, 103–129. Special issue on Integration of Natural Language and Vision Processing.

FILLMORE, CHARLES. 1968. The case for case. In *Universals in Linguistic Theory*, ed. by Emmon Bach & Robert T. Harms, 1–88. New York: Holt, Rinehart and Winston.

FITTS, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology* 47, 381–391.

FLAVELL, JOHN H., PATRICIA H. MILLER, & SCOTT A. MILLER. 1993. *Cognitive Development*. Englewood Cliffs, NJ: Prentice–Hall, third edition.

FRANZ, ELIZABETH A., HOWARD N. ZELAZNIK, & GEORGE MCCABE. 1991. Spatial topological constraints in a bimanual task. *Acta Psychologica* 77, 137–151.

GALLESE, V., L. FADIGA, L. FOGASSI, & G. RIZZOLATTI. 1996. Action recognition in the premotor cortex. *Brain* 119(2), 593–609.

GEORGOPOULOS, APOSTOLOS P. 1993. Cortical representation of intended movements. In *Neuroscience: From Neural Networks to Artificial Intelligence*, ed. by Pablo Rudomin, Michael A. Arbib, Francisco Cervantes-Pérez, & Ranulfo Romo. Springer-Verlag.

GLEITMAN, LILA R. 1990. The structural sources of verb meanings. *Language Acquisition* 1(1), 3–55.

GODDARD, NIGEL. 1992. *The Perception of Articulated Motion: Recognizing Moving Light Displays*. University of Rochester dissertation.

GOLD, E. MARK. 1967. Language identification in the limit. *Information and Control* 10(5), 447–474.

GOLDBERG, ADELE E. 1995. *Constructions: A Construction Grammar Approach to Argument Structure*. University of Chicago Press.

GOPNIK, ALISON. 1981. Words and plans: Early language and the development of intelligent action. *Child Language* 9, 303–318.

——, & SOONJA CHOI. 1995. Names, relational words, and cognitive development in English and Korean speakers: Nouns are not always learned before verbs. In *Beyond Names for Things: Young Children's Acquisition of Verbs*, ed. by Michael Tomasello. Hillsdale, NJ: Lawrence Erlbaum Associates.

GRAFTON, SCOTT T., MICHAEL A. ARBIB, L. FADIGA, & G. RIZZOLATTI. 1996. Localization of grasp representations in humans by PET: 2. Observation compared with imagination. *Experimental Brain Research* (112), 103–111.

GRICE, H. PAUL. 1975. Logic and conversation. In *Syntax and Semantics*, ed. by Peter Cole & Jerry L. Morgan, volume 3. New York, NY: Academic Press.

HEBB, DONALD O. 1949. *The Organization of Behavior*. New York, NY: Wiley.

HEIBECK, TRACY. H., & ELLEN M. MARKMAN. 1987. Word learning in children: An examination of fast mapping. *Child Development* 58, 1021–1034.

HERTZ, JOHN A., ANDERS KROGH, & RICHARD G. PALMER. 1991. *Introduction to the Theory of Neural Computation*. Redwood City: Addison-Wesley.

HOPFIELD, JOHN J. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, 3088–3092.

HUTTENLOCHER, JANELLEN, PATRICIA SMILEY, & ROSALIND CHARNEY. 1983. Emergence of action categories in the child: Evidence from verb meanings. *Psychological Review* 90(1), 72–93.

JEANNEROD, MARC. 1988. *The Neural and Behavioural Organization of Goal-Directed Movements*. Oxford University Press.

——. 1994. The representing brain: Neural correlates of motor intention and imagery. *Behavioral and Brain Sciences* 17(2).

——. 1997. *The Cognitive Neuroscience of Action*. Oxford, UK: Blackwell.

JOHNSON, MARK. 1987. *The Body in the Mind*. The University of Chicago Press.

JURAFSKY, DANIEL. 1996. A probabilistic model of lexical and syntactic access and disambiguation. *Cognitive Science* 20, 137–194.

KANDEL, ERIC R., JAMES H. SCHWARTZ, & THOMAS M. JESSELL (eds.) 1991. *Principles of Neural Science*. New York: Elsevier, third edition.

KAY, D. A., & J. ANGLIN. 1982. Overextension and underextension in the child's expressive and receptive speech. *Journal of Child Language* 9, 83–98.

KAY, PAUL, & CHAD K. MCDANIEL. 1978. The linguistic significance of the meaning of basic color terms. *Language* 54(3), 610–646.

LAKOFF, GEORGE. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press.

——, & MARK JOHNSON. 1980. *Metaphors We Live By*. University of Chicago Press.

LAMMENS, JOHAN M. 1994. *A Computational Model of Color Perception and Color Naming*. State University of New York at Buffalo dissertation. Available as CS Dept. Technical Report TR-94-26.

LANDAU, BARBARA, & LILA R. GLEITMAN. 1985. *Language and Experience: Evidence from the Blind Child*. Harvard University Press.

LATASH, MARK L. 1993. *Control of Human Movement*. Human Kinetics Publishers.

LEDERMAN, SUSAN J., & ROBERTA L. KLATZKY. 1996. Action for perception: Manual exploratory movements for haptically processing objects and their features. In *Hand and Brain: The Neurophysiology and Psychology of Hand Movements*, ed. by Alan M. Wing, Patrick Haggard, & J. Randall Flanagan. San Diego: Academic Press.

LEVISON, LIBBY. 1993. The topic is *Open*. Presented at the U. Penn Linguistics Colloquium.

——. 1995. Connecting planning and acting: Towards an architecture for object-specific reasoning. Ph.D. thesis proposal, Computer and Information Science Department, University of Pennsylvania.

LYNCH, GARY, & RICHARD GRANGER. 1992. Variations in synaptic plasticity and types of memory in corticohippocampal networks. *Journal of Cognitive Neuroscience* 4(3), 189–199.

MACKENZIE, CHRISTINE L., & THEA IBERALL. 1994. *The Grasping Hand*. North-Holland. Advances in Psychology, vol. 104.

MANDLER, JEAN M. 1992. How to build a baby: II. Conceptual primitives. *Psychological Review* 99(4), 587–604.

MARCUS, GARY F. 1993. Negative evidence in language acquisition. *Cognition* 46, 53–85.

MARKMAN, ELLEN M. 1989. *Categorization and Naming in Children*. Cambridge, MA: MIT Press.

MCCARTHY, JOHN. 1977. Epistemological problems in artificial intelligence. In *Proc. IJCAI–77*.

MELTZOFF, A. N., & M. K. MOORE. 1977. Imitation of facial and manual gestures by human neonates. *Science* 198, 75–78.

MITCHELL, THOMAS M. 1980. The need for biases in learning generalizations. Technical Report CBM-TR-117, Dept. of Computer Science, Rutgers University.

—— 1982. Generalization as search. *Artificial Intelligence* 18(2), 203–226.

MOODY, JOHN, & CHRISTIAN DARKEN. 1988. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Models Summer School*, ed. by David Touretzky, Geoff Hinton, & Terrence Sejnowski, 133–143. San Mateo, CA: Morgan Kaufmann.

MURATA, TADAO. 1989. Petri nets: Properties, analysis and applications. *Proceedings of IEEE* 77(4), 541–580.

NAPIER, JOHN. 1993. *Hands*. Princeton, NJ: Princeton University Press. Revised by Russell H. Tuttle.

NARAYANAN, SRINI. 1996. The cognitive semantics of aspect. In *Proceedings of Conceptual Structure, Discourse and Language II*.

——. 1997. *Knowledge-based Action Representations for Metaphor and Aspect (KARMA)*. Computer Science Division, EECS Department, University of California at Berkeley dissertation.

NELSON, KATHERINE. 1973. *Structure and strategy in learning to talk*. University of Chicago Press. Monograph of the Society for Research in Child Devleopment 38(1-2).

NILSSON, NILS J. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1, 139–158.

OMOHUNDRO, STEPHEN. 1992. Best-first model merging for dynamic learning and recognition. Technical Report TR-92-004, International Computer Science Institute, Berkeley, CA.

PEARL, JUDEA. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

PEARSON, K. G. 1993. Common principles of motor control in vertebrates and invertebrates. *Annual Reviews in Neuroscience* .

PINKER, STEVEN. 1989. *Learnability and Cognition: The Acquisition of Argument Structure*. Cambridge, MA: MIT Press.

REGIER, TERRY. 1996. *The Human Semantic Potential*. MIT Press.

REISIG, WOLFGANG. 1985. *Petri Nets: An Introduction*. Berlin: Springer–Verlag.

RISSANEN, JORMA. 1984. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory* IT-30(4), 629–636.

ROSCH, ELEANOR. 1977. Human categorization. In *Advances in Cross-Cultural Psychology*, ed. by N. Warren, volume 1. New York: Academic Press.

——, CAROLYN MERVIS, WAYNE GRAY, DAVID JOHNSON, & PENNY BOYES-BRAEM. 1976. Basic objects in natural categories. *Cognitive Psychology* 8, 382–439.

RUMELHART, DAVID E., GEOFFREY E. HINTON, & RONALD J. WILLIAMS. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by David E. Rumelhart & James L. McClelland, 318–362. MIT Press.

——, & JAMES L. MCCLELLAND (eds.) 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press. Volumes 1 and 2.

SACERDOTI, EARL D. 1975. The nonlinear nature of plans. In *Proc. IJCAI–75*, 206–214.

SCHANK, ROGER C. 1975. *Conceptual Information Processing*. Amsterdam: North-Holland.

SHASTRI, LOKENDRA. 1988. *Semantic Networks: An evidential formalization and its connectionist realization*. Los Altos, CA: Morgan Kaufmann.

——. 1997. A model of rapid memory formation in the hippocampal system. In *Proceedings of the 19th Cognitive Science Society Conference*.

——, & VENKAT AJJANAGADDE. 1993. From simple association to systematic reasoning: a connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences* 16(3).

——, DEAN J. GRANNES, SRINI NARAYANAN, & JEROME A. FELDMAN. 1997. Connectionist parameterized routines. Submitted to Neural Information Processing Systems 10. Also presented as a poster at the 19th Cognitive Science Society Conference.

SHIK, MARK L., & GRIGORI N. ORLOVSKY. 1976. Neurophysiology of locomotor automatism. *Physiology Review* 56, 81–110.

SISKIND, JEFFREY MARK. 1992. *Naive Physics, Event Perception, Lexical Semantics and Language Acquisition*. Massachusetts Institute of Technology dissertation.

——. 1995. A computational study of lexical acquisition. *Cognition* 50, 1–33.

SLOBIN, DAN I. (ed.) 1985. *The Crosslinguistic Study of Language Acquisition*. Hillsdale, NJ: Lawrence Erlbaum Associates.

—— (to appear). Why are grammaticizable notions special?—A reanalysis and a challenge to learning theory. In *Language Acquisition and Conceptual Development*, ed. by Melissa Bowerman & Steven C. Levinson. Cambridge University Press.

SOECHTING, JOHN F., DAVID C. TONG, & MARTHA FLANDERS. 1996. Frames of reference in sensorimotor integration: Position sense of the arm and hand. In *Hand and Brain: The Neurophysiology and Psychology of Hand Movements*, ed. by Alan M. Wing, Patrick Haggard, & J. Randall Flanagan. San Diego: Academic Press.

STERNBERG, S., S. MONSELL, R. L. KNOLL, & C. E. WRIGHT. 1978. The latency and duration of rapid movement sequences: comparisons of speech and typewriting. In *Information Processing in Motor Control and Learning*, ed. by G. E. Stelmach, 117–152. New York: Academic.

STOLCKE, ANDREAS. 1994. *Bayesian Learning of Probabilistic Language Models*. University of California, Berkeley dissertation.

STOUTAMIRE, DAVID P. 1995. The pSather 1.0 manual. Technical Report TR-95-058, International Computer Science Institute.

TALMY, LEONARD. 1985. Lexicalization patterns: semantic structure in lexical forms. In *Language typology and syntactic description: grammatical categories and the lexicon*, ed. by Timothy Shopen, chapter 2. Cambridge University Press.

——. 1988. Force dynamics in language and thought. *Cognitive Science* 12, 49–100.

TANJI, JUN. 1994. The supplementary motor area in the cerebral cortex. *Neuroscience Research* 19, 251–268.

THRUN, SEBASTIAN B. 1992. The role of exploration in learning control. In *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, ed. by D. White & D. Sofge. Van Nostrand Reinhold.

TOMASELLO, MICHAEL. 1991. *First Verbs*. Cambridge University Press.

——. 1992. Joint attention on actions: Acquiring verbs in ostensive and non-ostensive contexts. *Journal of Child Language* 19, 311–333.

—— (ed.) 1995a. *Beyond Names for Things: Young Children's Acquisition of Verbs*. Hillsdale, NJ: Lawrence Erlbaum Associates.

——. 1995b. Pragmatic contexts for early verb learning. In *Beyond Names for Things: Young Children's Acquisition of Verbs*, ed. by Michael Tomasello. Hillsdale, NJ: Lawrence Erlbaum Associates.

——, & Patricia J. Brooks. (to appear). Early syntactic development: A construction grammar approach. In *The Development of Language*, ed. by M. Barrett. London: UCL Press.

Tversky, Amos, & Daniel Kahneman. 1974. Judgment under uncertainty: Heuristics and biases. *Science* 185, 1124–1131.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13, 260–269.

Weber, Susan Hollbach. 1994. A structured connectionist model of figurative adjective-noun combinations. In *Analogy, Metaphor, and Reminding*, ed. by John A. Barnden & Keith J. Holyoak. Norwood, NJ: Ablex. Vol. 3 in the series *Advances in Connectionist and Neural Computation Theory.*

Whorf, Benjamin Lee. 1956. *Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf*. Cambridge, MA: MIT Press.

Wilensky, Robert. 1991. Extending the lexicon by exploiting subregularities. Technical Report TR-91-618, University of California at Berkeley Computer Science Division.

Wing, Alan M., Patrick Haggard, & J. Randall Flanagan (eds.) 1996. *Hand and Brain: The Neurophysiology and Psychology of Hand Movements.* San Diego: Academic Press.

Winograd, Terry. 1973. A procedural model of language understanding. In *Computer Models of Thought and Language*. New York: W. H. Freeman.

WU, DEKAI. 1992. *Automatic Inference: A Probabilistic Basis for Natural Language Interpretation.* University of California, Berkeley dissertation. Available as UC Berkeley Computer Science Division Technical Report UCB/CSD 92/692.

ZLATEV, JORDAN. 1992. A study of perceptually grounded polysemy in a spatial microdomain. Technical Report TR-92-048, International Computer Science Institute, Berkeley, California.