Connecting Language and Vision Using a Conceptual Semantics

by

John R. Bender

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by .....
Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Connecting Language and Vision Using a Conceptual Semantics

by

John R. Bender

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In an effort to better understand how language and vision connect, I have implemented theories of the human capacity for description and visualization. A visually-inspired representation for language termed *Lexical-Conceptual Semantics* (LCS), invented by Ray Jackendoff, is key to these theories and the resulting programs. The first program, called VISUALIZE, maps a sentence to an image or movie. VISUALIZE demonstrates that

- LCS simplifies the visualization task.
- Visualization can be performed by *incremental composition*.
- Common-sense questions can be answered by *reading off* the visualized image, rendering symbolic rules unnecessary.

Program DESCRIBE inverts VISUALIZE, generating sentences from an image or movie. DESCRIBE v.1 shows that *matching by hallucination* can work when there is a strong "top-down" model and few hallucinations to consider. It uses bi-directional search to bring fragments of LCS structure into correspondence with parts of an image. DESCRIBE v.2 demonstrates that LCS frames can be filled using the results of surprisingly few *visual predicates* computed on the image.

Taken together, these programs indicate that LCS holds promise as a description language for vision.

Thesis Supervisor: Patrick H. Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgments

I am grateful to Nick Cassimatis for our countless conversations about AI and for introducing me to under-appreciated cognitive science work, particularly Ray Jackendoff's. I thank Tim Chklovski for showing me to AI.

I thank Ray Jackendoff, whom I have never met but upon whose work this thesis squarely rests.

I thank Marvin Minsky for his inspiring aphorisms.

Finally, I thank my advisor Patrick Winston for teaching me how to evaluate AI research, both others' and my own.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Two simple questions motivate this thesis:

1. How do we talk about what we see?

2. How do we visualize what others say?

Answering both questions will require a better understanding of how human language and vision connect. Through such an understanding, it is my hope that we make progress toward two goals: an explanation of physical thinking (scientific goal) and simulating such thought in a machine (engineering goal). It is my belief that these two goals are not separate but, in fact, joined at the hip. Creating a human-level AI will require understanding human thinking and, conversely, a full explanation of human cognition will take the form of a computational model.

As a step toward these goals, In have chosen to focus on visualization and description because I believe these reveal how we conceptualize and reason about the physical world. I have implemented and tested theories of how language and vision connect in computer programs. Central to these programs is a representation for language created by Ray Jackendoff called Lexical-Conceptual Semantics (LCS) [13]. Introduced in 1990, this representation has yet to be recognized by AI researchers. The LCS makes possible what previous representations could not.

Chapter 2 introduces Jackendoff's representation and compares it with previous representations for language. Logical Form (LF) and Conceptual Dependency (CD)

are shown to lack several important features which LCS possesses. In particular, LCS

- Represents what is in our minds, not what is in the world; it is a cognitive representation.
- Includes the contribution of syntax to semantics.
- Makes explicit notions like PATHs and PLACEs which are absent in LF and CD.
- Extends readily to domains other than the spatial-temporal which it specializes in.

In chapter 3 I describe program VISUALIZE which uses the LCS to map sentences to images. Humans perform visualization effortlessly, as part of a more general capacity for physical simulation. VISUALIZE partly models that performance. The program takes as input an LCS structure and outputs an image or movie depicting the state or event described. From these images, VISUALIZE can answer questions about the event simply by *reading off* the answer from the image. No symbolic rules are required, making VISUALIZE a more realistic model of human question-answering than rule-based systems.

Chapter 4 presents program DESCRIBE which performs the inverse operation of VISUALIZE: mapping an image to a sentence. This inverse operation is considerably more difficult because it is a many-to-many relation whereas VISUALIZE's task was many-to-one.

DESCRIBE uses a strategy that can be summarized as "Only see what you can say" in order to constrain visual interpretation. It searches for an image description in the small space of LCS representations, not the larger space of geometric configurations.

I've written two versions of DESCRIBE.

DESCRIBE v.1 matches images to sentences by aggressively *hallucinating* possible matches and then testing them against the image. It uses a bi-directional search

method to bring the elements of the image and the constituents of the LCS representation into correspondence.

DESCRIBE v.2 answers the question "What minimal information must vision supply to language in order to fill the LCS structures?" The short answer is "not much." By computing a few simple visual predicates on the image, DESCRIBE v.2 can fill LCS frames. These predicates can be computed retinotopically, using the same operations Ullman describes in his *Visual Routines* [32].

I leave for appendix A the psychological and neurological evidence for the Jackendoff representation. This evidence assures us that the representation is on the right track.

The final chapter reviews the contributions of this thesis. Briefly, they are:

VISUALIZE

- Jackendoff's representation simplifies the visualization task.
- The process of visualization parallels syntactic parsing.
- Question-answering using images and movies obviates the need for symbolic rules and is more cognitively plausible.

DESCRIBE v.1

- *Matching by hallucination* can work when there is a strong top-down model and few hallucinations to consider.
- The LCS is a useful description language for vision because it collapses the usual geometric description-space.
- Visual scene recognition can be accomplished by *incremental, bi-directional* composition rather than the more common one-shot, uni-directional filter methods.

DESCRIBE v.2

- LCS frames can be filled using the results of visual predicates computed on an image.

- Visual predicates can be computed using only retinotopic routines which *fixate*, *saccade* and *track*.

Chapter 2

Lexical-Conceptual Semantics

Chapter Summary

In this chapter I describe a novel representation for language invented by Ray Jackendoff which is core to the programs presented in succeeding chapters. The representation is termed *Lexical-Conceptual Semantics* (LCS). I compare LCS to previous representations for language, including Logical Form and Conceptual Dependency, showing how LCS improves upon them because it is cognitive and syntax-driven. Although the LCS centers on spatial language, it easily extends to other domains like time and possession.

2.1 Jackendoff's Novel Representation for Language

This work is inspired by and depends heavily upon a new representation for language invented by linguist Ray Jackendoff [13]. Jackendoff decomposes a wide swath of natural language into a set of primitive features and functions. What's most surprising about this decomposition is how *much* of language can be satisfyingly decomposed into so *few* primitives. In this regard, LCS is to natural language what machine instructions are to source code.

For a more complete introduction to Jackendoff's representation than I will present here, see Chapter 9 of his Semantics and Cognition or, for the most thorough treatment, see his magnum opus Semantic Structures [12, 13]. Below I describe Jackend-

off's work only in enough detail to make my own work comprehensible.

2.1.1 *Lexical*-Conceptual Semantics

Jackendoff terms his representation Lexical-Conceptual Semantics (LCS). It is *lexical* because it is intended to be included in every lexical entry, alongside phonology and syntax. The fact that this representation is lexical has some important implications.

First, it implies that each word has an isolated meaning apart from other words in the sentence. This contrasts with Logical Form (LF) – the classical semantics used by linguists – in which there is no clear correspondence between individual words and their meaning. In LCS, there is a direct mapping. Each part of speech corresponds to a conceptual entity: nouns correspond to THINGS, verbs to EVENTS and STATES, prepositional phrases to PATHS and PLACEs.

Second, lexicalness implies that semantic structures are constructed in parallel with syntax. So, for example, the prepositional phrase *in the house* has a direct correspondence in semantics, [$_{Place}$ IN [$_{Thing}$ HOUSE]], which is built from the meanings of the individual words IN and HOUSE. This lexicalness also stands in contrast to LF and other representations.

Because LCS is lexical, constructing an LCS structure from natural language is as straightforward as parsing syntax. For example,

"John walked into the room." $\rightarrow$

[s [np [n John]] [vp [v walk] [pp [p into] [n room]]]] $\rightarrow$

[$_{Event}$ GO [$_{Thing}$ JOHN] [$_{Path}$ TO [$_{Place}$ IN [$_{Thing}$ ROOM]]]]

Note how each syntactic constituent corresponds to a conceptual entity. I'll explain why this is important by way of comparing LCS to other representations.

2.2 Previous Representations for Language

To fully appreciate the advantages of the LCS representation, I will describe traditional representations for sentences, indicate their problems, and show how the LCS representation remedies them.

2.2.1 Logical Form (LF)

Traditionally, linguists represent sentences using a semantics called Logical Form (LF) [2]. Not surprisingly, LF has its roots in classical logic.

In a sense, LF is not strictly comparable to LCS. LF is meant to represent what is true of the world; it is, like any logic, a "science of statements" [34]. In contrast, LCS is a mental representation, meant to to represent what is true in our minds. As a mental representation, the LCS primitives map to concepts, not sets of possible worlds. As Jackendoff put it, "unlike propositions of the standard truth-conditional logic, LCS expressions refer not to the real world or to possible worlds, but rather to the world *as we conceptualize it*" (italics his) [12].

As an example of LF, consider *"Bob broke a glass"*, which might be represented:

$\exists$ `x (glass (x)` & `broke(Bob, x))`

Immediately, we see several problems with this formulation. First, *"a glass"* corresponds to no constituent in the formula. Instead, we roughly have *"there exists some thing which is made of glass"*. Second, the *"a"*, which is the most embedded and least meaningful element of *"Bob broke a glass,"* is at the highest, least embedded level in the formula, the existential quantifier.

If the above formulation is incorrect, then what is the correct one? We cannot say because LF has no theory of how to map a sentence's syntax to a logical formula. In the literature, LF semanticists debate which of several possible formulae is best for a given sentence. There is even debate over the meaning of primitive relations like *or* (is it the truth-conditional inclusive *or* or is it exclusive *or*, as in *"is it the truth-conditional inclusive* or *or is it exclusive* or*"*?) Lacking a consistent method for

mapping syntax to semantics, LF cannot account for the contribution syntax makes to semantics. Indeed, as *"Bob broke the glass"* demonstrates, the mapping is the reverse of what would be expected.

To see some the problems in *interpreting* LF formulas, consider how LF would represent the simple sentence *"*`A` *is in* `B`*"*. Intuitively, it as a relation, IN(`A`,`B`). Indeed, past AI researchers have treated prepositions this way [4]. But this intuitive treatment actually convolves two entities which ought to be kept separate: a place, IN(`B`), and a state, BE(`A`, IN(`B`)). Convolving IN and BE is a mistake because the place IN(`B`) can be more that just a location for `A` – it could also be:

- A point along `A`'s path, e.g. *"The water went in the bucket and out the hole in the bottom."*
- A place where `A` is kept, e.g. *"The dishes stayed in the cupboard."*

But in LF, IN(`B`) can only mean all possible worlds where "`B` is IN", which is not meaningful. This gets at the crux of the problem with LF: all statements evaluate to true/false for a set of possible worlds. To determine which worlds those are requires knowing the meaning of the sentence, which is what LF was supposed to represent to begin with!

Fundamentally, LF is circular because it does not decompose words to primitive features and functions as LCS does. In LF, every word is its own primitive. Even though *fly, walk* and *slither* are all verbs of motion, each has its own incomparable predicate: *fly(x,y,z)*, *walk(x,y,z)* and *slither(x,y,z)*. This of course leads to a profusion of incomparable predicates. In LCS, by contrast, similar verbs decompose similarly. For instance,

BUTTER: [$_{Event}$ GO [$_{Thing}$ BUTTER] [$_{Path}$ TO [$_{Place}$ ON [$_{Thing}$ X]]]]

HAMMER: [$_{Event}$ GO [$_{Thing}$ HAMMER] [$_{Path}$ TO [$_{Place}$ ON [$_{Thing}$ X]]]]

In both cases, we have one object taking a path onto another. This similarity is captured by the decomposition, but would be obscured in LF. We could conceivably

Table 2.1: Example role frame

SEND	
AGENT	Bob
PATIENT	Mary
OBJECT	bomb
INSTRUMENT	plane

make this same decomposition in LF, but then we'd actually be simulating LCS in LF.

To summarize, the problems with LF as a representation for language are:

- LF represents what is in the world, not what is in our heads, thereby failing to capture how we think about language.
- LF has no consistent method for mapping from syntax to semantics.
- LF does not decompose word meanings into primitive features and functions, but must be interpreted using a circular "possible worlds" model

2.2.2 Role Frames and Conceptual Dependency

AI researchers have made attempts to represent natural language for the purpose of manipulating language in computers. Role frames, which were invented by linguists, were quickly adopted by AI [6, 21]. Role frames reveal that syntax does indeed carry semantic information: the position of a noun with respect to the verb indicates its role in the action. Role frames use linguistic roles like *agent*, *patient* and *instrument* to fill verb-specific frame slots. For example, *"Bob sent the bomb to Mary by plane."* would fill the SEND frame shown in table 2.1.

Different verbs have frames with different slots, depending on what linguistic roles the verb takes. Some verbs have slots pre-filled with default values. For instance, the INSTRUMENT of the verb *shot* is usually *gun*.

Roger Schank augmented role frames with his Conceptual Dependency (CD) theory [26]. CD includes the usual role frame slots, but adds new ones like PTRANS

(physical transfer), ATRANS (change of attribute) and MTRANS (change of memory).

CD theory was an important first step in decomposing verbs into primitive features and functions. For instance, the verbs *'give', 'receive'* and *'exchange'* all decompose into variants of PTRANS. Schank's purpose in making this decomposition was to allow a series of sentences (a 'story') to be connected via the primitives, with the goal being to understand the story as a whole. That goal was met only for a few idealized stories, but nevertheless CD was a milestone because it represented sentence semantics *conceptually*, as its name implies.

However, *as* a conceptual semantics, CD is rife with problems. First, several of its 'primitives' are not really primitive. For instance,

- GRASP
- INGEST (put into mouth)
- EXPEL (send out of body)

all ought to be decomposed into more primitive elements [1]. Siskind gives a full treatment of GRASP in terms of contact, support and attachment, which I won't repeat here [28]. INGEST and EXPEL are both verbs of motion. In LCS, they are:

INGEST: [$_{Event}$ GO [$_{Thing}$ X] [$_{Path}$ TO [$_{Place}$ IN [$_{Thing}$ MOUTH]]]]

EXPEL: [$_{Event}$ GO [$_{Thing}$ X] [$_{Path}$ FROM [$_{Place}$ IN [$_{Thing}$ BODY]]]]

The 'primitives' ATRANS (example: give) and PTRANS (example: go) are also only variants of GO. And MOVE and PROPEL are considered primitive in CD, but doesn't PROPEL just mean cause to MOVE? In the LCS,

MOVE: [$_{Event}$ GO [$_{Thing}$ X] [$_{Path}$ ...]]

PROPEL: [$_{Cause}$ CAUSE Y [$_{Event}$ GO [$_{Thing}$ X] [$_{Path}$ ...]]]

[1] Schank may have invented INGEST and EXPEL because they were useful for the restaurant story which motivated his work.

The lack of orthagonality in Schank's primatives can be traced to his neglect of paths. Paths are at the core of at least seven of his primatives: PTRANS, ATRANS, MTRANS, MOVE, PROPEL, INGEST, EXPEL. The consquences of this neglect are clearer when we consider what PTRANS, for example, can and cannot represent. PTRANS will capture *"The boat sailed from Boston to New York"* but not any of these paths:

1. The boat sailed along the river.
2. The boat sailed toward shore.
3. The boat sailed away from port.
4. The boat sailed through the strait.

In all cases there is motion without point-to-point 'physical transfer', which PTRANS cannot account for. Nor can CD account for cases of PATHs *without* motion, such as:

1. The road extends along the river. *(extension)*
2. The sign pointed toward California. *(orientation)*

In LCS these are,

[$_{State}$ EXTEND [$_{Thing}$ ROAD] [$_{Path}$ ALONG [$_{Thing}$ RIVER]]]

[$_{State}$ ORIENT [$_{Thing}$ SIGN] [$_{Path}$ TOWARD [$_{Place}$ CALIFORNIA]]]

A practical theory of semantics must say how syntax is mapped onto the semantic representation, but CD does not do this. Instead, Schank dismisses syntax as a 'surface' representation and takes pride in CD's independence from syntax. The consquences of not deriving semantics from syntax are apparent in the examples above.

To summarize, CD theory suffers from problems similar to LF:

1. CD does not decompose language down to the most cognitively basic features and functions, although it takes a step in that direction

2. CD does not include syntax's contribution to semantics

3. CD primatives are not orthagonal because they don't factor out paths

2.3 The Complete LCS Formalism

For completeness, here is the full LCS formalism:

- [THING]

- [PLACE] ← PLACE-FUNC([THING])

  PLACE-FUNC ∈ (AT, ON, IN, ABOVE, BELOW, BEHIND, ...)

- [PATH] ← PATH-FUNC([PLACE] | [THING])

  PATH-FUNC ∈ (TO, FROM, TOWARD, AWAY-FROM, VIA, ALONG, ...)

- [EVENT] ←

  GO ([THING], [PATH]) |

  STAY ([THING], [PLACE])

- [STATE] ←

  BE ([THING], [PLACE]) |

  ORIENT ([THING], [PATH]) |

  EXTEND ([THING], [PATH])

- [CAUSE] ← CAUSE ([THING], [EVENT])

Bear in mind that these entities themselves decompose into even more primative features and functions. In his Semantic Structures, Jackendoff likens this representation to John Dalton's atomic theory of chemistry [13]. Dalton's theory bottomed-out at atoms, not the electrons or protons which, it would later be discovered, comprise them. Still, Dalton's rules for combining atoms remained true even after his 'atoms' were resolved into finer particles. Similarly, Jackendoff expects that his entities – PATH, THING, etc. – will someday be resolved into more primative elements, but this will not negate those larger entities. Indeed, since creating the LCS representation, he has already decomposed some of the entities himself [10].

What is the exact meaning of PLACE-FUNCs and PATH-FUNCs? PLACE-FUNCs map a THING to a PLACE. Similarly, PATH-FUNCs map PLACEs or THINGs to PATHs. Jackendoff writes that these are functions of 3D object models (in particular, he imagines 3-D Marr-like models) [25]. But I will show in the final chapter of this thesis that many of Jackendoff's PLACE- and PATH-FUNCs do not require a 3D model, but can instead be computed retinotopically.

2.4 Much of natural language is spatial

Jackendoff's representation centers on the physical world, as opposed to social or abstract domains. One might therefore challenge the representation's generality by asking, "Just how much of langauge *is* spatial?".

One crude method to answer this question is to examine the most frequently occuring words in English. Table 2.2 lists the 100 most frequently occuring words in English, ranked in desceding order of frequency. Keep in mind that these are computed from written and not spoken material, mainly newspaper articles [7].

After removing all articles, conjunctions, pronouns, possesives, and helping verbs we have a list that includes mostly words about space/time (*) and quantity (**), shown in table 2.3.

This crude assessment indicates that, indeed, spatial/temporal and quantitative words comprise the bulk of natural language.

Table 2.2: 100 most frequent English words

the	this	we	can	man
of	had	him	only	me
and	not	been	other	even
to	are	was	new	most
a	but	when	some	made
in	from	who	could	after
that	or	will	time	also
is	have	more	these	did
was	an	no	two	many
he	they	if	may	before
for	which	out	then	must
it	one	so	do	through
with	you	said	first	back
as	were	what	any	years
his	her	up	my	where
on	all	its	now	much
be	she	about	such	your
at	there	into	like	may
by	would	than	our	well
i	their	them	over	down

Table 2.3: Most frequent words after removing filler words.

of	what	after *
to *	up *	did
in *	about *	many **
for	into *	before *
with	than	through *
as	other	back *
on *	new *	years *
at *	time *	where *
by *	these	much **
not	two **	well
from *	do	down *
one **	first **	
all **	any **	
there *	now *	
when *	out *	
who	over *	
more **	man	
no	even	
out *	most **	
said	made	

* indicates space/time words
** indicates quantity words

2.5 LCS is extensible

Although all the LCS examples presented so far are in the spatial domain, the representation readily extends to other domains. *"Until 10 o'clock"* locates an interval of time just as *"until the road ends"* represents a path segment; *"at/by/around noon"* is like *"at/by/around the park"*. The path

[$_{Path}$ FROM [$_{Place}$ AT [2:00]], TO [$_{Place}$ AT [3:00]]]

specifies the interval of time between 2:00 and 3:00. All PLACE-FUNCs and PATH-FUNCs used in the spatial domain keep their same meaning in the time domain. Time is simply the 1-dimensional case. Some more complicated examples:

1. "The meeting will begin at midnight." →

   [$_{Event}$ GO [$_{Thing}$ MEETING] [$_{Path}$ FROM [$_{Place}$ AT [time MIDNIGHT]]]]

2. "The presentation was during the day." →

   [$_{State}$ EXTEND [$_{Thing}$ PRESENTATION] [$_{Path}$ ALONG [time DAY]]]

3. "I was at the office until 9:00." →

   [$_{State}$ BE [$_{State}$ STAY [$_{Thing}$ I] [$_{Place}$ AT [$_{Thing}$ OFFICE]]] [$_{Path}$ TO [time 9:00]]]

The LCS also readily extends to verbs of possesion. In *possession-space*, the meaning of GO becomes transfer of possession:

[$_{Event}$ GO [$_{Thing}$ MONEY] [$_{Path}$ FROM [$_{Thing}$ TOM], TO [$_{Thing}$ HARRY]]]

Physical proximity equates with ownership. Again, the meanings of all PATH-FUNCs and PLACE-FUNCs are preserved.

The LCS appears at first incapable of representing social and emotional statements. However, Leonard Talmy has shown that many social and emotional sentences can be analyzed in terms of "Force Dynamics", a notation like LCS [31]. For instance,

She pressured John to accpt the money. →

$[_{Cause}$ CAUSE $[_{Thing}$ SHE] $[_{Event}$ GO $[_{Thing}$ MONEY] $[_{Path}$ TO $[_{Thing}$ JOHN]]]]

Talmy shows that verbs like *pressured* and *forced* behave according to the same rules whether applied to physical or social situations.

These mappings from physical space onto other spaces are not analogies. Instead, the claim is that we necessarily make this mapping (even if unconsciously) every time we think in those "abstract" spaces [17, 8]. Many more examples of how abstract domains map onto physical ones are presented in Lakoff and Johnson's book Metaphors We Live By [17].

2.6 What LCS represents

It is important to be clear about what the LCS representation is meant to capture and what it leaves out. Obviously, the symbols $[_{Path}$ TO $[_{Place}$ ON $[_{Thing}$ TABLE]]] do not represent the particular contour of the path itself. The actual path contour is represented in vision. But to *describe* that contour with words, we need PATH-FUNCs which reference points in the visual domain. That is what language is for and what LCS represents: a method of referring to what we see.

Tokens in the LCS like BALL are meant to be pointers to information in visual agencies which store shape, color, size, etc. The LCS leaves these features to vision.

Although the LCS is representation is closely tied to language, it is invariant to some aspects of language:

- Some word choices: *"Bill walked into the room"* and *"Bill came into the room"* are both represented by $[_{Event}$ GO $[_{Thing}$ BILL] $[_{Path}$ TO $[_{Place}$ IN($[_{Thing}$ ROOM])]]]

- Word order: *"Bill walked into the room"* and *"Into the room came Bill"* are represented the same

- Language: Both Japanese and English sentences use the same LCS

2.7 Summary

To summarize, the benefits of the LCS representation are that it is:

Lexical : It decomposes the meaning of words into primitive features and functions.

Cognitive : It represents what is in our heads, not what is in the world.

Syntax-driven : It comes with a method of mapping syntax to semantics which takes syntax's contribution to semantics into account.

Extensible : It maps readily to more abstract domains like time and possession.

Specific : It captures exactly the information a language system needs to produce sentences; the rest is left to other faculties to represent.

Chapter 3

VISUALIZE: From Sentence to Image

Chapter Summary

In an effort to model human visualization ability, program VISUALIZE maps sentences to images. The LCS representation makes visualizing sentences straightforward. The process of visualization is *compositional*, executed in same order that the sentence is syntacticly parsed. Once the sentence is visualized, questions about it can be readily answered by *reading off* the generated image(s). Thus the need for symbolic rules is obviated.

So far I have described Jackendoff's representation and compared it to other representations for language, in order to highlight its benefits. To a linguist, the very fact that Jackendoff's representation satisfyingly captures many example sentences is evidence enough of its power. But to an AI researcher, the power of any representation ultimately hinges on the question, "What problems can it be used to solve?". To answer that question, I have written several programs that use the LCS representation. I will discuss them in the remainder of this thesis.

The first program, named VISUALIZE, takes as input an LCS representation and outputs a 3D image or short movie. The goal of the program is to make a correspondence between what we say and what we see. The program demonstrates

how this correspondence can be made by *incremental composition* which pays strict attention to the syntactic embedding of the sentence.

Once a sentence has been visualized, the program can answer simple questions about what the sentence means. This is accomplished simply by *reading off* the answer from the image. No symbolic rules are required.

3.1 From sentence to image

Program VISUALIZE models a task humans perform effortlessly: visualization of a sentence, or more poetically "seeing in the mind's eye". Brain scan experiments demonstrate that during a visualization task the same regions are active as are used in normal vision, indicating that visualization is to some extent a controlled hallucination. We do not yet know how these hallucinations are kept separate from reality; perhaps they are isolated by a mechanism akin to how the brain stem cuts off signals to the body during dreaming.

Psychological evidence indicates that visualization is true to reality. Mentally rotating an object takes as much time to perform as would be required to rotate the object in reality [27]. Thus visual hallucination simulates reality not in an abbreviated, symbolic fashion but with high physical fidelity.

The goal of VISUALIZE is to partially model human visualization. It takes as input a sentence, given in LCS, and produces as output an image or movie (series of images) corresponding to the sentence. Past visualization programs began with explicit machine instructions for constructing the visual representation, rather than natural language, and represented only static relations, not motion [3]. VISUALIZE begins with natural language, as given by the LCS representation, and can represent both static scenes (images) as well as motion events.

3.2 Example Run

VISUALIZE's operation is best illustrated by example. Consider

"Bill went into the room."

represented in LCS as

[$_{Event}$ GO [$_{Thing}$ BILL] [$_{Path}$ TO [$_{Place}$ IN [$_{Thing}$ ROOM]]]]

Given this input, VISUALIZE returns a scene giving the coordinates of all objects or, for moving objects, their position as a function of time. This function can be 'expanded' into a series of images locating Bill along his path into the room. The movie can then be used to answer common-sense questions about the event, such as *"Where is Bill after he came into the room?"*.

The procedure for constructing a visual representation for this example is:

1. Imagining [$_{Thing}$ ROOM] by recalling its object model from a database.
2. Computing the PLACE [$_{Place}$ IN [$_{Thing}$ ROOM]] by applying the IN operator to the room object.
3. Computing the PATH [$_{Path}$ TO [$_{Place}$ IN [$_{Thing}$ ROOM]]] by imagining a vector which terminates at the PLACE [$_{Place}$ IN [$_{Thing}$ ROOM]].
4. Imagining [$_{Thing}$ BILL] by recalling his object model from the database.
5. Creating a function of time which gives Bill's location along the PATH computed in step 3.

Figure 3.2 shows the flow of control in VISUALIZE. First the sentence is decomposed to its primitive constituents, then those constituents are visualized, and finally reassembled into the final image.

The result is a 3-dimensional, viewer-independent model of the scene(s). To verify that VISUALIZE's output is correct, I wrote a utility to draw the final image or "play" the movie on screen.

To accomplish its task, VISUALIZE requires a database of object models for objects like Bill, room, table, cup, etc. These models are extremely course but do

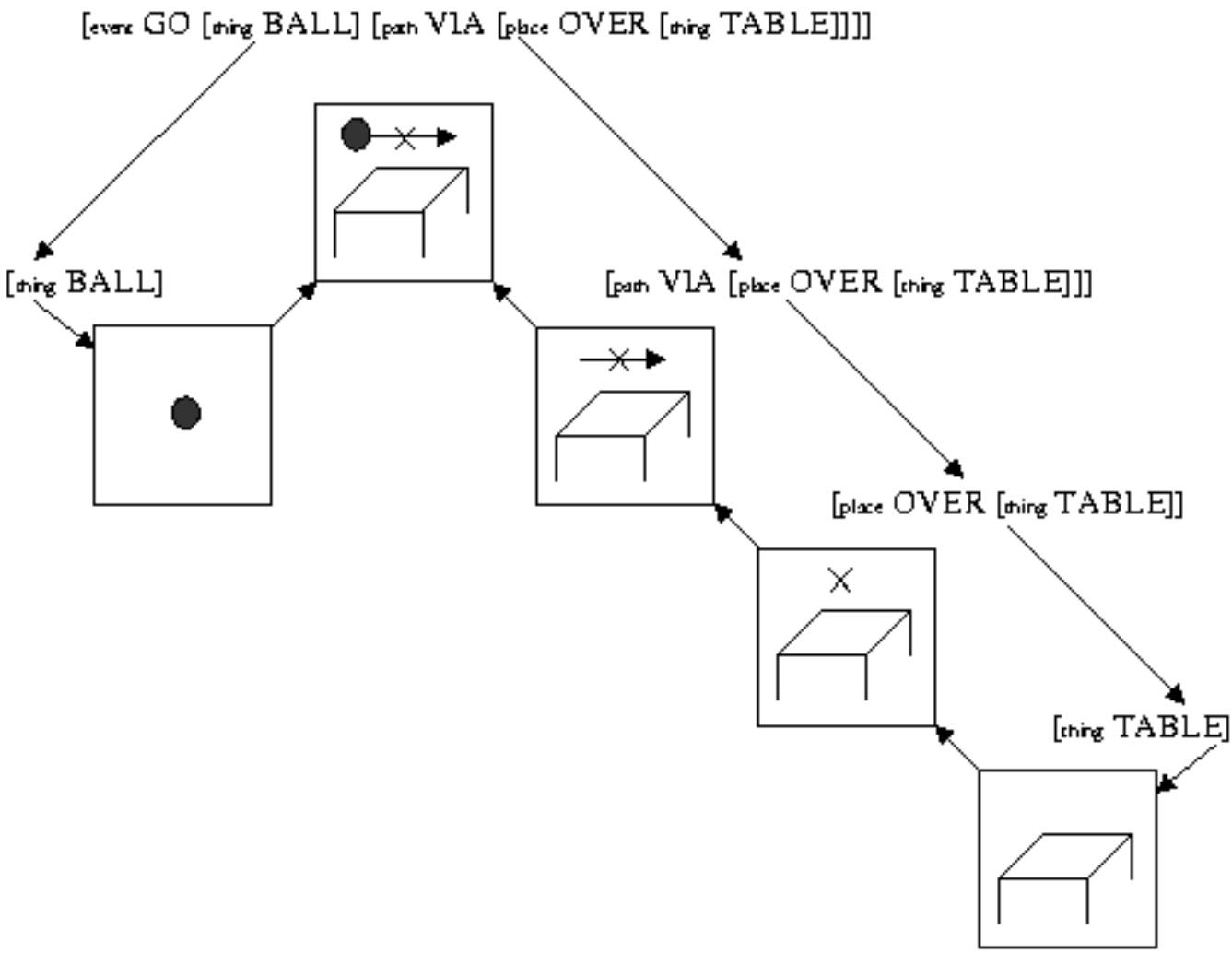


Figure 3-1: Flow of control in VISUALIZE.

capture the relative sizes and dimensions of the objects. Visualizing a THING just means recalling its object model from the database.

Visualizing PLACEs is more complicated than THINGs. PLACE-FUNCs for prepositions like IN, ON, ABOVE have been implemented. Each takes a THING as its argument and outputs a point (example: AT), surface (example: ON) or space (example: IN).

PATH-FUNCs for prepositions like TO, FROM, VIA, etc. take PLACEs as input and generate vectors representing a path.

Verbs of motion like GO take an object and path vector as input and return the object's location as a function of time.

VISUALIZE has a small lexicon of verbs and corresponding LCS representations. For example, the verbs *walk* and *butter* are represented in the lexicon as:

WALK: $[_{Event}$ GO $[_{Thing}$ X] $[_{Path}$ Y]]

BUTTER: $[_{Event}$ GO $[_{Thing}$ BUTTER] $[_{Path}$ TO $[_{Place}$ ON($[_{Thing}$ X])]]]

Obviously VISUALIZE can only accept sentences containing objects in its database, verbs in its lexicon, and prepositions given by implemented PLACE- and PATH-FUNCs. However, there is no reason why, in principle, VISUALIZE could not easily scale up to include a wider breadth of language, simply by filling the object database and lexicon. No changes need be made to VISUALIZE itself.

3.3 The Complete Mapping

For completeness, here is a detailed description of the full mapping made by VISUALIZE:

THING $\rightarrow$ Object: A 3D model, which may be a singe prism or, in the case of objects composed of parts, a hierarchy of prisms.

PLACE $\rightarrow$ Space: A point, line, area or volume, depending on what the PLACE-FUNC is (for example, 1D: a road, 2D: a baseball field or 3D: the interior of a room).

PATH $\rightarrow$ Function f(t): Gives (x,y,z) at t, where t ranges over the time interval the object moved or, in the case of ORIENT and EXTENT, t is a dummy variable. By using a generalized function rather than a linear vector, PATHs can represent any trajectory or contour, not only straight lines. In the case where f(t) represents motion, it also implicitly represents the velocity and acceleration of the object along its path, though VISUALIZE does not make use of this information.

EVENT $\rightarrow$ Movie: a series of sets of objects and their locations. The number of 'scenes' in the movie can vary depending on the granularity of the time step of f(t) [1].

STATE $\rightarrow$ Image: a set of objects and their locations.

CAUSE $\rightarrow$ Pair: (THING, EVENT) where THING is the *agent*.

3.4 The mapping is made in parallel with syntax

The mapping from sentence to image follows the same embedding pattern as the sentence's syntax. Most embedded are visualized first, least embedded last. Why is this important? Because, by visualizing sentences in the order they are parsed, we implicitly account for cognitive constraints on visualization. To see why this is the case, consider:

1. The bike is in front of the house.

2. The house is behind the bike.

Though both sentences convey the same information about the relative positioning of the bike and the house, people presented with both prefer (1) to (2). The reason for this preference should be obvious. In (1), the light, mobile bike is being located

[1] The ability to vary the granularity of the simulation is important. Sometimes a course simulation is sufficient to answer a question, such as "Where is Bill after entering the room?". Other times a finer-grained simulation is required, such as "Did Bill touch the wall as he entered the room?"

with reference to the heavy, immobile house. Syntax parallels physics: first locate the immobile objects, then locate the mobiles with respect to them. In (2), however, syntax and physics conflict, and this is why (2) is judged awkward: the syntax signals that the bike be placed first but physics says the heavy, immobile house ought to be.

This preference indicates that we place objects not with regard to which is *introduced* first in the sentence, but rather which is *parsed* first. Put differently, syntax directs the order in which a visualization is constructed.

3.5 Problems encountered

In writing VISUALIZE, I encountered several problems which are interesting in their own right and ought to be taken into consideration in any future version of VISUALIZE.

VISUALIZE's place-functions (i.e. prepositions) compute PLACEs in an object-independent manner but this is not always the case in natural language. For example, VISUALIZE computes

- [$_{Place}$ IN [$_{Thing}$ ROOM]] by averaging the dimensions of the room
- [$_{Place}$ ON [$_{Thing}$ TABLE]] by choosing an arbitrary point on the surface of the table

Thus place-functions compute places *independent* of the object they operate on. But in natural language, prepositions sometimes modify nouns in a noun-dependent manner. For instance, *"in the room"* takes the conventional meaning of the interior space bounded by the room's walls, but *"in bed"* does not pick out a space inside the mattress. *"In bed"* refers to the space between the mattress and cover. To account for noun-dependencies like this, VISUALIZE would either require that its place-functions be sensitive to the object operated on, or that its objects come attached with non-default interpretations.

A second problem is that prepositions have variable interpretations depending on the location of the speaker. For most spatial prepositions, there are at least four pos-

sible interpretations: viewer-centered ("deictic"), object-centered ("intrinsic"), which can be either along the canonical axis or motion axis, and absolute ("basic") [5]. For instance, *"in front of the car"* could mean:

1. The space in front of the headlights (object-centered along canonical axis)
2. The space between the car and the speaker, which depends on the location of the speaker (viewer-centered)
3. The space in front of whichever direction the car happens to be moving, which could even be canonical *"behind the car"* if the car were moving in reverse (object-centered along axis of motion)

Which of these applies in a given situation depends also on the particular object. *"In front of the car"* is more tightly bound to the canonical axis interpretation than *"in front of the rock"* because cars have a well-defined canonical axis whereas rocks do not.

To fully account for preposition ambiguity, VISUALIZE would need to know the location of the speaker, the canonical axis of the objects involved and the direction of motion of the objects at the time of the utterance. There are psychology experiments showing which of the several interpretations is preferred for different values of these variables. There is also considerable work attempting to pin down exactly what *"on"* or *"above"* mean without regard to context [5].

A third issue encountered involves the physical constraints on visualization. Visualizing *"the box is on the table"* seems like a simple matter of locating the box at the place given by [$_{Place}$ ON [$_{Thing}$ TABLE]]. But if the table is already covered with other items, we must be sure that the box isn't placed on those items and that there is enough table surface remaining to place the box without it falling off.

I added a check to VISUALIZE to prevent such object inter-penetrations and, if necessary, randomly choose another point *"on the table"* (there are many such points) where the box will not inter-penetrate other objects. I also added a simple mechanism for "bending" paths around obstacles to prevent inter-penetration when an object moves along a path.

But these workarounds only begin to address the full range of physical constraints on visualization, which include object stability, friction and gravity. I have little doubt that a complete physics simulator could be built to account for all these constraints, but the more difficult problem of how humans selectively apply them or find alternatives to satisfy them is still an open question.

3.6 VISUALIZE answers questions

VISUALIZE can answer common sense questions by *reading off* answers from the image. The advantage of a visual representation over a purely symbolic one like logic is that visual representations make a large number of inferences immediately available, without requiring multistep deductions and the combinatorial explosions which can come with them [1].

The following example situations and questions can be answered by VISUALIZE:

1. **Given:** "Jan buttered the toast." "Jan put the the toast on her plate."

   Q: Is the butter above the plate?

   A: Yes.

2. **Given:** "The cup and the plate are on the table."

   Q: How many THINGs are on the table?

   A: Two.

3. **Given:** "John walked into the room."

   Q: Is John's nose in the room?

   A: Yes.

In (1), a CYC-like system would require a "Transitivity of ON" rule to answer the question [18]. VISUALIZE instead generates an image for the situation and then tests weather or not the BUTTER's z coordinate is greater than the PLATE's.

Situation (2) requires the ability to count, which can be reduced to two operations: mark and increment. As we visit each object which is on the table, we mark it and increment the counter. Ullman points out in *"Visual Routines"* that *mark* is generally a useful operation for visual tasks like testing if a contour is bounded [32]. For that task, *mark* stores the start of tracking the contour and, if that *mark* is reached again during tracking, the contour is bounded.

For (3), a CYC-like system would need a rule:

IF (AND (IN A B) (PART-OF C A)) →

THEN (IN C B)

But a system which can perform physical simulation need only imagine A being in B and then test if C, which happens to be a part of A, is also in B. No special rules like the one above are required.

Another way to see the advantage of images over symbolic representations is to consider the amount of information required to symbolicly represent the information contained in an image. Consider an image of N objects. We can make at least one statement about the relative positions of every pair of objects, such as *"A is above B"* or *"B is left of C"*. There are N! such pairs, so N! symbolic sentences are needed to capture the same information implicitly contained in the image. A picture is indeed worth 1000 words or, in this particular case, a picture of only seven objects would be worth 5040 symbolic statements.

In addition to using images to answer questions about static scenes, VISUALIZE can use movies to answer questions about the STATE of affairs at various times during an EVENT. For instance, given that *"Bill walked into the room"*, the program can answer the common-sense question *"Is Bill now in the room?"* by finding that, in the final scene of the movie, Bill is indeed in the room.

This is accomplished by a procedure which takes an EVENT as input and "expands" it into a series of scenes which are then each visualized. The last several scenes in the sequence will show Bill in the room. More difficult questions questions like *"Did Bill go through the doorway?"* could conceivably be answered by testing Bill's path

for intersection through the plane bounded by the doorway, but VISUALIZE's object models are currently too primitive to accomplish this.

Jackendoff, in a 1976 paper [11], writes inference rules for his LCS representation to answer questions like *"Is Bill in the room?"*:

IF $[_{Event}$ GO $[_{Thing}$ X] $[_{Path}$ TO($[_{Place}$ Y])]] from time T_1 to T_2

THEN $[_{State}$ BE $[_{Thing}$ X] $[_{Place}$ AT($[_{Place}$ Y])] at time T_2

VISUALIZE demonstrates how the same inference can be made without such a rule. Jackendoff later downplays his early work as cognitively implausible, in favor of the method VISUALIZE employs.

Chapter 4

DESCRIBE: LCS for Vision

Chapter Summary

Program DESCRIBE tackles the inverse problem of VISUALIZE: it maps images to sentences. Two versions of describe have been implemented. The first demonstrates that *matching by hallucination* can succeed using the LCS to restrict the number of hallucinations (i.e. the size of the 'description-space') under consideration. The program uses bi-directional search to bring the hallucinations, coming from the top-down direction, into correspondence with LCS structures generated from the image, coming in the bottom-up. The second version of DESCRIBE takes a very different approach by filling LCS frames using primitive *visual predicates* computed on the image. The surprising finding is that the results of computing only a few simple predicates suffices to fill the LCS frames. Both versions of DESCRIBE demonstrate that LCS holds promise as a description language for vision.

The overarching goal of this thesis is to better understand how human language and vision connect. Having explored the language → vision direction of that connection in VISUALIZE, I next turn to the opposite direction, vision → language.

In "How do we talk about what we see?", Macnamara points out that in order to talk about what we see, information provided by the visual system must be translated into a form compatible with the information used by the language system [20]. In this

chapter, I present a program called DESCRIBE which uses the LCS representation to make that translation.

Traditionally, the problem of mapping an image to its description would be considered a Vision project, but DESCRIBE bears little resemblance to classical vision systems. DESCRIBE does not take raw image input and does not perform object recognition, line detection, or any of the usual image processing techniques. DESCRIBE assumes that object segmentation, recognition, and tracking are solved problems.

The program is given the identities and location(s) of objects. The program's goal is to use this information to construct a sentence describing the path or other spatial relations between the objects involved. A second version of DESCRIBE is given less information; indeed, its purpose is to fill LCS frames using a *minimum* of information from vision.

4.1 DESCRIBE: From image to sentence

DESCRIBE faces a considerably harder task than VISUALIZE. VISUALIZE began with a symbolic representation (the LCS) and mapped it to any one of many correct analogical representations (3D images or movies). Thus VISUALIZE made a one-to-many mapping. DESCRIBE, however, must make a many-to-many mapping. For example, there are infinitely many images satisfying "`A` is above `B`". `A` could be two feet above `B`, ten feet, 100 feet, etc. Furthermore, there are potentially N! statements that can be made about an image containing N objects, as explained in the last chapter. Finally, agent and patient roles can often be reversed, as in "`A` is above `B`" vs. "`B` is below `A`". Considering all these dimensions of variation, how will DESCRIBE decide what to say?

The first version of DESCRIBE does not decide, but simply generates all possible sentences involving `A` and `B`, then matches them piecewise and returns all matches. The second version does not generate all possible sentences, but instead fills in LCS frames which can later be used to generate any sentence. This second version is a

more cognitively plausible model of how human vision might inform language.

4.2 Classical Vision Problems

To appreciate the task facing DESCRIBE, we must review the "vision problem". Vision researchers frame their field's problem as one of "inverse optics" [9]. The goal is to invert a 2D pixel array to recover the 3D geometry which generated it. This problem, framed as it is, is known to be under-constrained. A single 2D image can be generated by a potentially infinite number of 3D scenes.

Knowing that vision is an under-constrained problem, the obvious remedy is to add a constraint. That is effectively what DESCRIBE does; it constrains a potentially infinite number of scene descriptions to only LCS representations. In effect, its strategy is to take *"Only see what we can say"*.

The goal of any vision system is to produce a *description* of an image [9]. In practice, *description* has almost always meant the identity of the objects in the scene and their positions and orientations in 3-space. DESCRIBE does not produce such a geometric description but rather outputs a sentence – in the LCS representation – describing the image. Because a sentence covers many particular geometries (recall that "`A` is above `B`" covers infinitely many positionings of `A` and `B`), the geometric "description-space" is effectively collapsed. More importantly, it is collapsed along exactly the categories which humans care most about: "`A` is above `B`", not "`A` is at `(1.3, 4, 7)` and `B` is at `(1.3, 4, 9)`".

4.3 DESCRIBE assumes object recognition

Object recognition has historically been the centerpiece of vision research, to the detriment of motion processing and higher-level vision. DESCRIBE, however, assumes object recognition is a solved problem and focuses instead on motion and the relations between objects. Is this a fair assumption?

Recent vision research indicates that using motion and color (another traditionally

ignored feature), object recognition is greatly simplified. Jeffrey Siskind, for example, has put inexpensive CCD cameras and fast processors to good use in an active vision system [29]. Siskind has demonstrated that object segmentation can be achieved with stunning accuracy using only motion and color to segment objects. Once segmented, object recognition is easier.

Some researchers take issue with Siskind's methods, believing they are somehow "unfair." But, if such criticisms hold, nature could also be accused of cheating. For, as Shimon Ullman points out, motion and color are extracted at the lowest levels of vision, in parallel units which cover the entire visual field, both in humans and animals [32].

Siskind's work suggests that earlier vision systems built in a static, colorless world may have tackled some artificially difficult problems. Given the progress he has made, along with the ever-increasing feasibility of brute-force object recognition [22], it seems reasonable for DESCRIBE to assume that object recognition is solved.

4.4 DESCRIBE v.1: Matching by Hallucination

The first version of DESCRIBE uses the generate-and-test paradigm to match an image to a sentence. The program generates LCS structures, VISUALIZEs them and then matches those visualizations against the input image. This version is obviously not meant to model human performance, as humans do not hallucinate ad hoc until coming upon a match. But DESCRIBE v.1 does demonstrate that "matching by hallucination" is possible and feasible when the LCS representation limiting the number of matches under consideration.

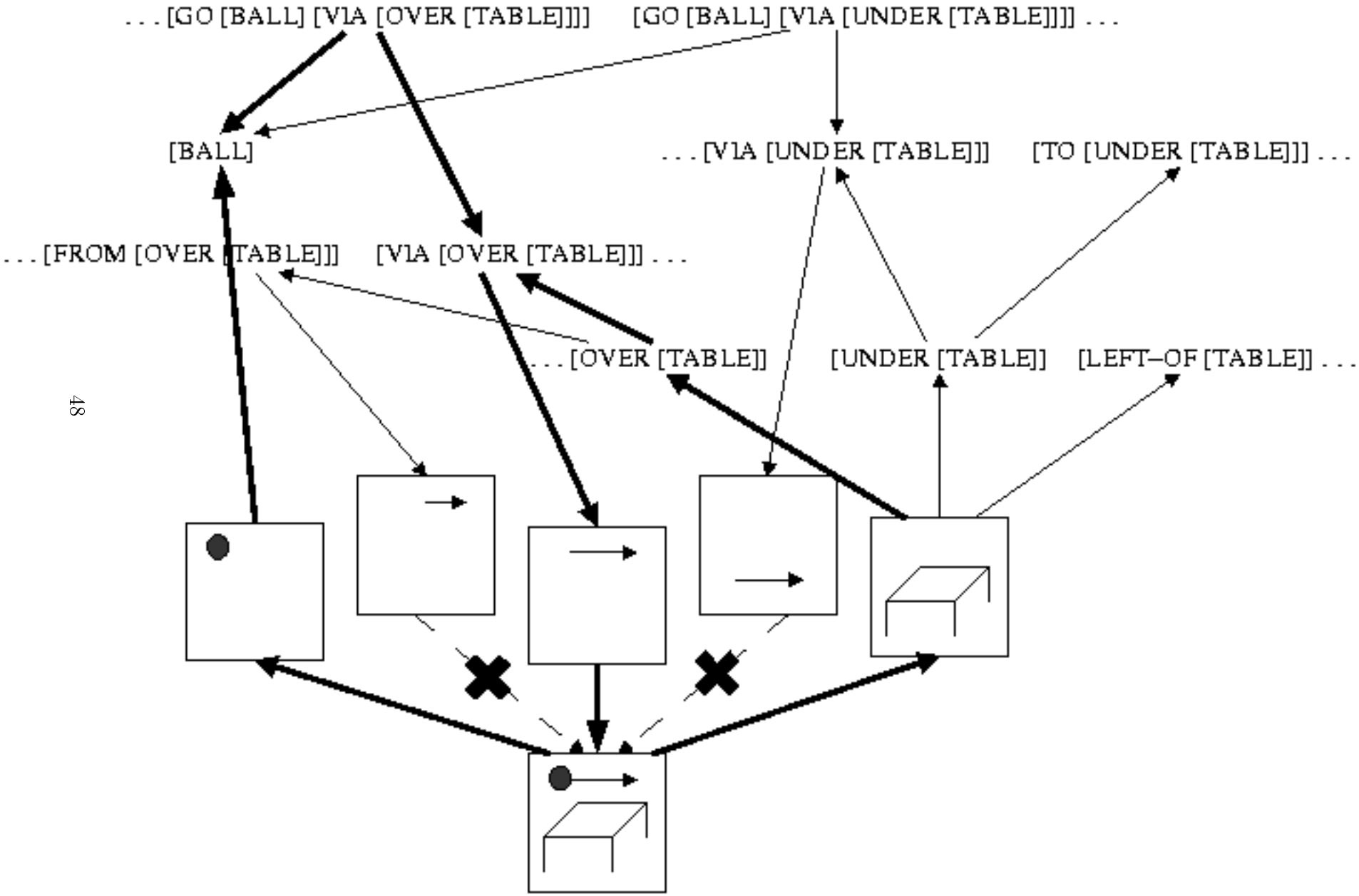
To guide the search for a match between the hallucinations and the actual image, I implemented the bidirectional search method described by Ullman [32]. This method is inspired by neurological findings suggesting that object recognition succeeds when top-down and bottom-up processing streams, both operating in parallel, meet at an intermediate point. Computationally, this method has the benefit of halving the depth of uni-directional search.

In DESCRIBE, the "top" of top-down corresponds to the hallucinated LCS representation, while the "bottom" is the input image. The program simulates parallelism by having the top and bottom streams take turns deepening their respective search trees. At each iteration, either the top or bottom will expand a node in its search tree. Coming from the top, expansion means "select one constituent from a larger LCS structure". Coming from the bottom, expansion means "select one constituent from the image".

All nodes at the frontiers of the search trees are marked *primed*. Primed nodes are preferentially expanded at the next iteration. This way, when either the top or bottom stream encounters a node already primed by the other, the search quickly terminates. One stream will immediately retrace a path already laid down by the other, and a match is then declared. See figure 4.4 for an example of how a match is made.

If the top is a complete LCS structure and the bottom an image, what then are the intermediate nodes? They are partial LCS structures – THINGs, PLACEs, PATHs – or parts of the image. Recall that the LCS representation is heavily embedded. DESCRIBE v.1 takes advantage of this by, for instance, applying all PLACE-FUNCs (e.g. "ON", "UNDER") to a THING in the bottom-up direction. Top-down, it begins with completed LCS structures like EVENTs and STATEs which are decomposed into smaller units to be matched against the image.

DESCRIBE v.1 demonstrates how scene recognition can be compositional. In contrast, neural nets and other uni-directional image processing techniques recognizes a whole scene in one-shot or not at all. DESCRIBE v.1 instead recognizes the scene in an incremental, piecewise fashion, with processing shifting direction between source and target. For example, an object in the image first spurs the generation of a larger structure, like a PLACE relative to that object. This is the bottom-up direction. Next, a full LCS structure at the top (a hallucination) is decomposed into smaller pieces which seek a match in the image. Eventually, all fragments generated bottom-up come into correspondence with the pieces of larger LCS structures coming from the top down, and a complete match is made.


. . . [GO [BALL] [VIA [OVER [TABLE]]]]
[GO [BALL] [VIA [UNDER [TABLE]]]] . . .
[BALL]
. . . [VIA [UNDER [TABLE]]]
[TO [UNDER [TABLE]]] . . .
. . . [FROM [OVER [TABLE]]]
[VIA [OVER [TABLE]]] . . .
. . . [OVER [TABLE]]
[UNDER [TABLE]]
[LEFT–OF [TABLE]] . . .

DESCRIBE v.1's main flaw, however, is that as a model of human performance it is an implausibly aggressive hallucinater. In terms of the generate-and-test methodology, we would say its 'generator' is too permissive. A second problem is that it terminates when one matching sentence is found when, in fact, many other sentences could also describe the image. Which is "right" or are all right to various degrees? These issues led to the development of DESCRIBE v.2.

4.5 DESCRIBE v.2: Filling LCS Frames Retinotopically

The primary question motivating the development of DESCRIBE v.2 is "What minimal information must vision supply to language?" More specificly, what low-level detectors must vision have in order to fill an LCS structure? Rather than generate-and-test in the top-heavy manner of DESCRIBE v.1, the methodology now is to fill bits bottom-up which can then generate *any* LCS structure which fits the image. This approach avoids generating unlikely candidates and it needn't decide on only one match but instead allows for any correct match.

Also, by treating the LCS as a static frame representation rather than an incrementally composed one, we can now pre-fill slots with defaults, give slots assignments with various degrees of certainty, and use information from other sources to fill slots. For example, knowing that a bird makes one shape of trajectory while a bug makes a more erratic one and a snake still another will alter our path perception of these animals.

The overall strategy employed by DESCRIBE v.2 is to compute what I call *visual predicates* from a raw image and use the results to fill LCS frame slots. First, the simple predicates `Exists?` and `Moving?` are applied to yield the objects (again, I assume object recognition) and whether they are moving (`1/0`). From this information, an LCS frame is initialized. In the case of moving object, a $[_{Event}$ GO ...] frame would be initialized.

Table 4.1: STATE example `Exists?` and `Moving?` bits

	`Exists?`	`Moving?`	
CUP	1	0	$\rightarrow$ [$_{State}$ BE [$_{Thing}$ CUP] [$_{Place}$?]]
TABLE	1	0	$\rightarrow$ [$_{State}$ BE [$_{Thing}$ TABLE] [$_{Place}$?]]

Next, the path the object took is related to other objects in the scene. At the most primitive level of human vision, a PATH is just a streak of activity across the retina (though DESCRIBE uses vectors). Visual predicates `DIR` and `TER` are applied to the image to select PATH-FUNCs. `DIR` computes whether the path is directed toward or away from a point (+/-/?) and `TER` determines if a point is a terminus of the path (1/0). From the results of these predicates, the PATH-FUNCs TO, FROM, TOWARD, AWAY-FROM, VIA are selected. Methods for computing `DIR`, `TER` and the PLACE-FUNCs are described below, though these have not been implemented. Instead DESCRIBE v.2 is handed the results of those visual predicates, though more realisticly it ought to compute them; perhaps DESCRIBE v.3 will.

DESCRIBE v.2's operation is best explained by example.

4.5.1 STATE example

As a first example, consider an image depicting "The cup is on the table."

To construct the LCS representation from this image, we begin by noting which objects exist and which are moving in table 4.1.

Existence and motion passively initiate an LCS frame. Once initiated, DESCRIBE actively seeks to fill in the other missing slots using visual predicates. The procedure is:

1. An object `exists` $\rightarrow$ [$_{Thing}$ CUP]
2. The CUP is not `moving`, so a BE frame is initialized $\rightarrow$ [$_{State}$ BE [$_{Thing}$ CUP] [$_{Place}$?]]
3. Having setup the BE frame, we need to fill the [$_{Place}$?] slot. Assuming TABLE

Table 4.2: STATE example `Intersects?` bits

PLACE	`Intersects?`
[$_{Place}$ ON [$_{Thing}$ TABLE]]	1.0
[$_{Place}$ OVER [$_{Thing}$ TABLE]]	0.0
[$_{Place}$ UNDER [$_{Thing}$ TABLE]]	0.0
.	.
.	.
.	.

Table 4.3: STATE example `Intersects?` for other objects

PLACE	`Intersects?`
[$_{Place}$ ON [$_{Thing}$ TABLE]]	1.0
[$_{Place}$ LEFT-OF [$_{Thing}$ WALL]]	0.9
[$_{Place}$ UNDER [$_{Thing}$ CEILING]]	0.7
[$_{Place}$ ABOVE [$_{Thing}$ FLOOR]]	0.6
.	.
.	.
.	.

is the only other object in the image, we begin to generate place-functions of TABLE (see table 4.2).

4. Each PLACE-FUNC of TABLE is tested for intersection with the CUP's position.

5. Having found that ON `intersects`, we complete the frame:

   [$_{State}$ BE [$_{Thing}$ CUP] [$_{Place}$ ON [$_{Thing}$ TABLE]]]

If there were other objects in the scene, there would be more than one intersection, with gradated judgments of how close to perfect (1.0) the intersection is (table 4.3).

With these additional PLACES, we can produce any sentence such as

[$_{State}$ BE [$_{Thing}$ CUP] [$_{Place}$ UNDER [$_{Thing}$ CEILING]]]

Note that a second sentence initiated at the start of this example

Table 4.4: EVENT example `Exists?` and `Moving?` bits

	`Exists?`	`Moving?`	
BIRD	1	1	$\rightarrow$ [$_{Event}$ GO [$_{Thing}$ BIRD] [$_{Path}$?]]
TREE	1	0	$\rightarrow$ [$_{State}$ BE [$_{Thing}$ TREE] [$_{Place}$?]]
PERCH	1	0	$\rightarrow$ [$_{State}$ BE [$_{Thing}$ PERCH] [$_{Place}$?]]

Table 4.5: Determining PATH-FUNC from `DIR` and `TER`

Point	`DIR?`	`TER?`	
`#213skd`	-	1	$\rightarrow$ [$_{Path}$ FROM `#213skd`]
`#762jnw`	+	0	$\rightarrow$ [$_{Path}$ TOWARD `#762jnw`]

[$_{State}$ BE [$_{Thing}$ TABLE] [$_{Place}$?]]

This sentence is also completed

[$_{State}$ BE [$_{Thing}$ TABLE] [$_{Place}$ UNDER [$_{Thing}$ CUP]]]

4.5.2 EVENT example

A more difficult example, this time involving motion, is

"The bird flew from its perch toward the tree."

1. An object `exists` $\rightarrow$ [$_{Thing}$ BIRD] (table 4.4)

2. The BIRD is `moving`, so initialize a GO frame $\rightarrow$ [$_{Event}$ GO [$_{Thing}$ BIRD] [$_{Path}$?]]

3. Having setup a GO frame, we need to fill in the [$_{Path}$?] slot. Knowing only the direction (`DIR`, +/-) and whether this is a terminal point (`TER`, 1/0) of the path, we can compute which of the PATH-FUNCs (TO, FROM, TOWARD, AWAY-FROM and VIA) holds in table 4.5.

   The gensyms `#213skd` and `#762jnw` represent absolute, retinotopic points along the path. They are 'markers' on the retina. I use gensyms because, at this stage of processing, we do not know the relation of these points to other objects.

Table 4.6: EVENT example `Intersects?` bits (1)

PLACE	`Intersects?`
[$_{Place}$ ON [$_{Thing}$ PERCH]]	0.9
[$_{Place}$ LEFT-OF [$_{Thing}$ HOUSE]]	0.4
[$_{Place}$ IN [$_{Thing}$ TREE]]	0.3
.	.
.	.
.	.

Table 4.7: EVENT example `Intersects?` bits (2)

PLACE	`Intersects?`
[$_{Place}$ IN [$_{Thing}$ TREE]]	0.8
[$_{Place}$ LEFT-OF [$_{Thing}$ HOUSE]]	0.4
[$_{Place}$ ON [$_{Thing}$ PERCH]]	0.2
.	.
.	.
.	.

4. We now relate the retinotopic points to objects in the scene using the same intersection method as before.

   (a) Find intersection points for `#213skd` (table 4.6)

   The LCS frame filled so far is:

   [$_{Event}$ GO [$_{Thing}$ BIRD] [$_{Path}$ FROM [$_{Place}$ ON [$_{Thing}$ PERCH]]]]

   (b) Find intersection points for `#762jnw` (table 4.7)

The final result, then, is:

[$_{Event}$ GO [$_{Thing}$ BIRD] [$_{Path}$ FROM [$_{Place}$ ON[$_{Thing}$ PERCH]] TOWARD [$_{Place}$ IN[$_{Thing}$ TREE]]]]

Another result with a poorer fit is:

[$_{Event}$ GO [$_{Thing}$ BIRD] [$_{Path}$ FROM [$_{Place}$ ON [$_{Thing}$ PERCH]] TOWARD [$_{Place}$ LEFT-OF [$_{Thing}$ HOUSE]]]]

Table 4.8: Computing PATH-FUNCs from DIR and TER

PATH-FUNC	DIR?	TER?
TO	+	1
FROM	-	1
TOWARD	+	0
AWAY-FROM	-	0
VIA	0	0

4.6 Computing PATH- and PLACE-FUNCs

In the examples above, I left out how the intermediate PATH-FUNCS (DIR and TER) and the PLACE-FUNCs (ON, IN, etc.) are computed from the image. DESCRIBE does not in fact compute these. I have assumed an as-yet-unimplemented visual system delivers the results of DIR and TER and the PLACE-FUNCs. Given those results, the language system has enough information to fill the PATH-FUNCs (see table 4.8).

How might DIR, TER and the PLACE-FUNCs be computed? What about PATH-FUNCs like ALONG which require tracing the path's contour, not just its endpoints and direction?

The answer to these questions comes in Shimon Ullman's *"Visual Routines"* [32]. There Ullman shows that visual processing can be directed to solve visual tasks. These tasks include testing a curve for closure or boundedness – routines more complicated than DIR, TER and others I have assumed. I will suggest a method for implementing my visual predicates in light of Ullman's work.

Ullman's routines use the same operations as natural vision systems: *'mark'*, *'fixate'*, *'saccade'* and *'track'*. Here is what each does:

- *mark* - store a point (equivalent to DESCRIBE's gensyms)
- *fixate* - fix on a point
- *saccade* - shift to a point
- *track* - follow a moving point

Monkey brain cell recordings respond selectively when the monkey fixates (fixation neurons), when it tracks (tracking neurons), and when it saccades to an object (saccading neurons) [23]. Psychological tests of human visual memory also demonstrate the reality of these operations [32].

Given we have these operations, the routines I've assumed can be computed like this:

- `(DIR A, P)` (i.e., is path `P` directed toward or away from point A?)
  1. fixate on `A`
  2. saccade to `P` head (let saccade time = t_h)
  3. saccade back to `A`
  4. saccade to `P` tail (let saccade time = t_t)
  5. if $t_h < t_t$, then `DIR+` else `DIR-`
- `(TER A, P)` (i.e., is point `A` one of the endpoints of path `P`?)
  1. fixate on any point on P
  2. track P until P ends (repeat for both directions)
  3. if `A` is where P tracking ends, then `A` is a `TER` point
- `Intersects`?
  1. fixate on `A`
  2. saccade to `B`
  3. let saccade time = t_s
  4. for $t_t < \epsilon$, `A` and `B` intersect

See figure 4.6 for examples of computing `DIR` and `TER`.

Note that, in these methods, saccade time is used as a measure of distance. Saccade direction can also be used to determine PLACE-FUNCs:

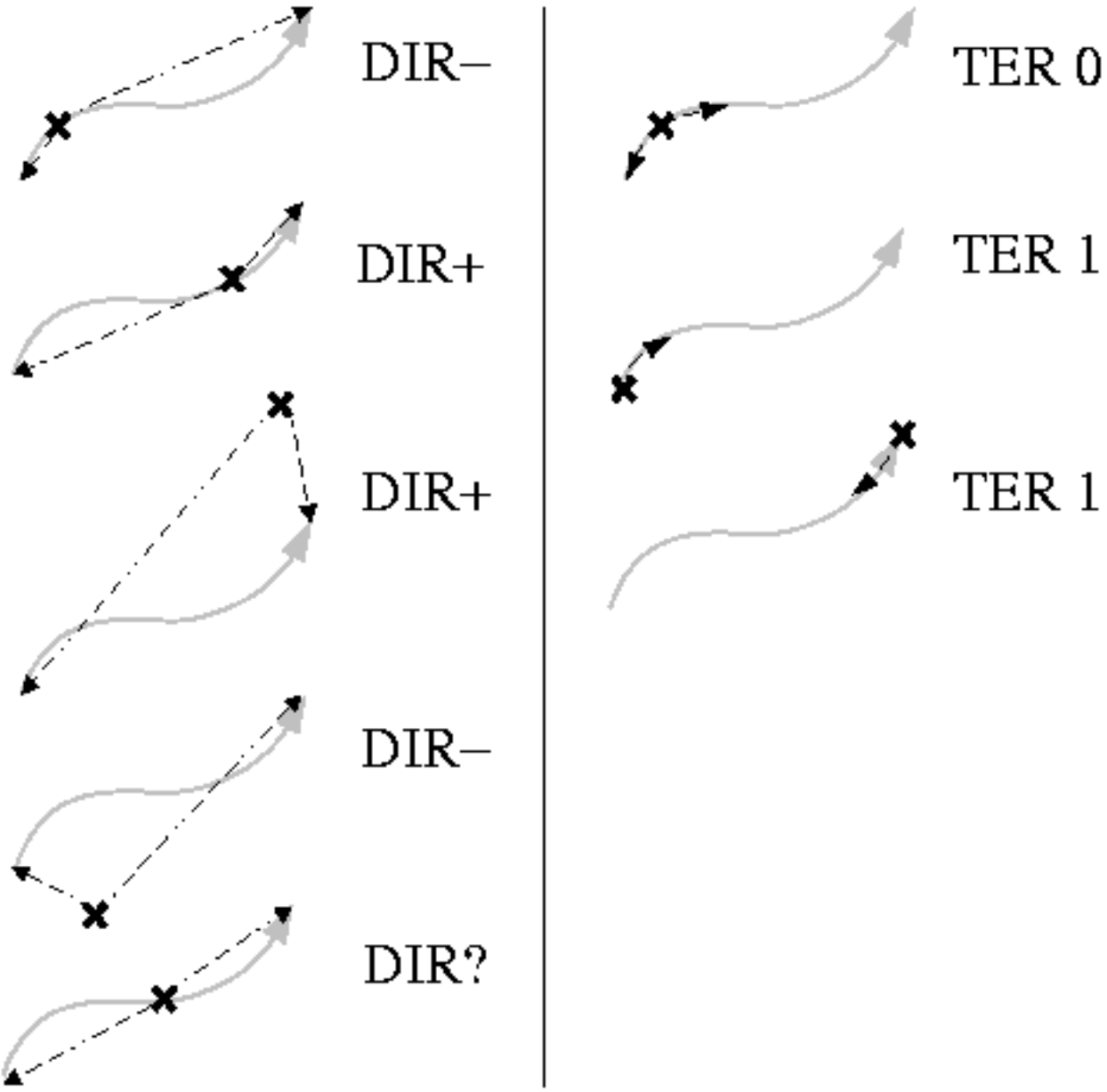


Figure 4-2: Example computations of `DIR` and `TER` visual predicates. A path is directed at a point (`DIR+`) when the time to saccade from that point to the head of the path is less than the time to saccade to the tail. Conversely for `DIR-`. When the times are equal, `DIR?` results, meaning the path is VIA the point (if the point is on path).

- `(ABOVE A B)`

    1. fixate on `A`
    2. saccade to `B`
    3. if saccade direction is down, then `A` is above `B`

- `(LEFT-OF A B)`

    1. fixate on `A`
    2. saccade to `B`
    3. if saccade direction is right, then `A` is left of `B`

Some more difficult PLACE-FUNCs:

- `(ALONG A B)`

    1. fixate on an endpoint of `A`
    2. track `A` until reaching opposite endpoint
    3. if `B` is fixated on all during tracking, then `A` is along `B`

- `(CONTACT A B)`

    1. fixate on a boundary point P of `A` (see Ullman for computing boundary points)
    2. track bounding contour of `A`
    3. if a boundary point of `B` is encountered, then `A` contacts `B`
    4. if point P is reached without encountering boundary point of `B`, then `A` does not contact `B`

- `(ON A B)`

    1. `(CONTACT A B)` and `(ABOVE A B)`

These routines give *one* method to compute the predicates, but as Ullman points out, there are often multiple visual routines which compute the same function.

Chapter 5

Contributions

The programs I've implemented demonstrate the power of Jackendoff's representation for visualization and description tasks.

In VISUALIZE, we saw how LCS made visualization by *composition* possible. In DESCRIBE v.1, LCS shrunk the description-space to make *matching by hallucination* work. And in DESCRIBE v.2, LCS focuses vision on only those *visual predicates* needed to fill the LCS frames.

5.1 VISUALIZE

VISUALIZE demonstrates that:

- Jackendoff's representation simplifies the visualization task.

- Visualization can be performed by an incremental composition which exactly parallels syntax. This method is true to human performance.

- Images and movies possess latent knowledge that obviate the need for symbolic rules. This knowledge can be recovered by *reading off* the image or expanding the movie into discrete images.

5.2 DESCRIBE

Both versions of DESCRIBE show that LCS holds promise as a description language for vision. In DESCRIBE v.1, the LCS shrunk the description-space being searched while in DESCRIBE v.2, the LCS served as a framework for organizing the results of computing *visual predicates* to form sentences.

5.2.1 DESCRIBE v.1

The first version of DESCRIBE was motivated by the question, "How can we invert the mapping made by VISUALIZE?" The main result is that inversion can be accomplished by piecewise matching of generated images against the actual image. A bi-directional search method brought together hallucinations (complete LCS structures) from the "top-down" and image elements from the "bottom-up" via partial LCS structures generated from the image elements. DESCRIBE v.1 demonstrates that visual recognition can be carried out in an incremental, compositional manner, rather than a one-shot, all-or-nothing method. The program also demonstrates the usefulness of a "top-down" force in directing recognition, as opposed to a strictly "bottom-up" methods.

5.2.2 DESCRIBE v.2

The second version of DESCRIBE was motivated by the question, "What minimal information must a vision system provide to language in order for language to generate all sentences true of that image?" The surprising answer is "very little". LCS structures can be filled using only retinotopic operations of the sort Ullman used in constructing visual routines – *fixate, track, saccade* – applied to salient points and streaks of activity across the retina. This second version of DESCRIBE did not actually implement those operations, but used simulated results to fill the LCS frames. DESCRIBE v.2 suggests a future project: implementing visual predicates for the LCS structures with only the primitive operations *fixate, saccade and track* and, eventually, executing these routines on real camera input. The final system, then, would in

fact be able to “say what it sees”.

Appendix A

The Evidence for Paths

Summary

Central to the LCS representation is the notion of a path. This chapter presents the psychological and neurological evidence for the cognitive reality of paths in humans and animals. This evidence assures us that LCS is on the right track.

The most innovative aspect of Jackendoff's representation is the notion of a path. A surprisingly large number of verbs involve motion, orientation, or extension along paths. For instance,

1. The boat traveled along the river. *(motion)*

2. The sign pointed along the river. *(orientation)*

3. The road extends alongside the river. *(extension)*

The LCS representation claims PATHs are a distinct ontological category, irreducible to other entities.

But what is the evidence that paths are cognitively distinct? How do we know that what we call a path is not a mere shorthand for a more basic representation, perhaps a list of `(time, location)` pairs? In this chapter I review infant and animal experiments demonstrating that paths are indeed a distinct, primitive category.

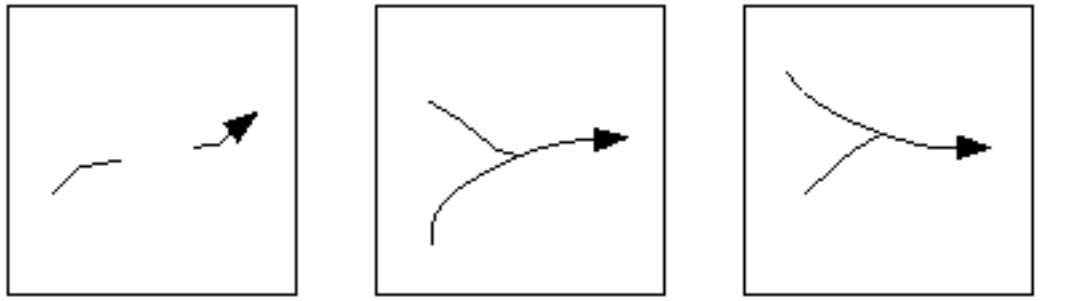


Figure A-1: Example violations of Spelke's principles of object motion.

A.1 Psychological Evidence for Paths

There is considerable psychological evidence for the cognitive reality of paths in the form of experiments on humans and animals.

A.1.1 Human Experiments

Elizabeth Spelke has shown that infants as young as 18 months stare longer at physically impossible paths [30]. Looking-time increases when any of the following three principles of object motion are violated:

CONTINUITY - "All parts of an object move on connected paths over space and time."

BOUNDEDNESS - "Two objects cannot occupy the same place at the same time."

COHESION - "All parts of an object move on connected paths over space and time."

Infants, despite having underdeveloped object recognition, have also been shown to smoothly pursue moving objects and, more impressively, they can do "predictive reaching" for a moving object [33]. The infant correctly estimates a moving object's velocity and the time needed to extend its arm such that the object will be where their

hand is by the time their hand arrives. This prediction ability requires interpolating the prior positions of the object into a curve which is then extrapolated, suggesting that object paths are represented continuously and not as a sequence of discontinuous states.

A famous experiment on adults indicates that paths are wired into the lowest levels of vision [16]. A person stands at a distance from two lights, one red and one green, positioned close together. First the red light is on and the green off, then red turns off and milliseconds later the green light turns on. The subject reports that *one* dot 'moved' rather than two alternating. Furthermore, they report the dot changed color gradually from red to green mid-path. These experiments indicate that the eye interpolates paths where none exist, even in the face of conflicting evidence from a color change. This suggests that paths are constructed at a lower level of vision than we can consciously control.

A.1.2 Animal Experiments

In *"What the frog's eye tells the frog's brain"*, Lettvin et. al. showed that frog vision is similarly wired at its lowest levels to see motion [19]. When a dot about the size of a bug streaks across the frog's retina, the ON/OFF cells fire. If the dot does not move, or moves at a slower or faster pace than a bug naturally would, the ON/OFF cells do not fire. These cells, along with others (ON only cells, OFF only cells and shape cells) form a 'bug detector' which solves the frog's vision problem.

Experiments with mice demonstrate that PATHs are whole conceptual entities [15]. When mice see food through a screen barrier, they do not run directly toward the food and collide with the barrier repeatedly until eventually bumping around it. Such 'hill climbing' behavior is how ants find food. But the mouse makes a smooth arc around the edge of the barrier and toward the food, as if it had planned a continuous path before taking its first step.

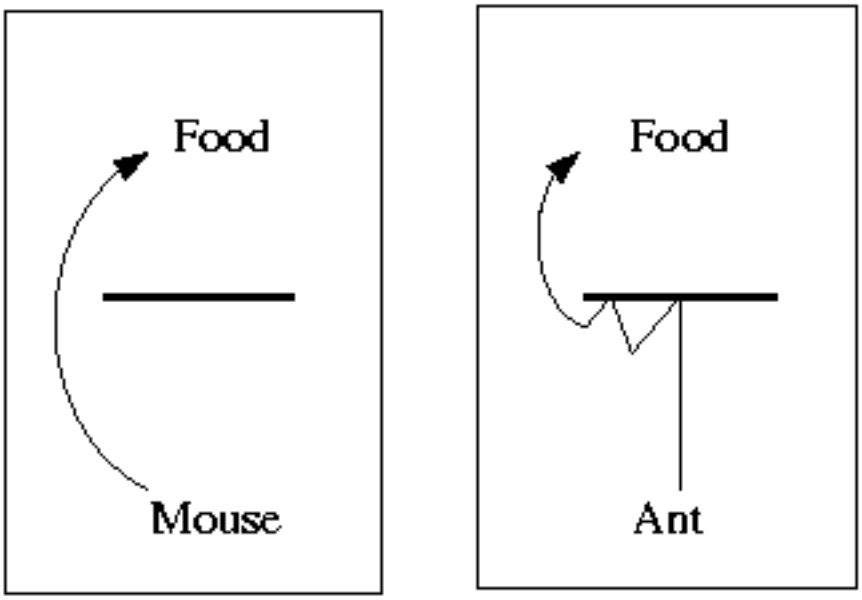


Figure A-2: The mouse's trajectory indicates a pre-planned path.

A.2 How might paths be implemented neurologically?

Neurological findings are beginning to reveal how paths are implemented in the human brain. We know that the hippocampus contains *place cells* which represent individual places [24]. We also know that the *direction code* is carried by a pattern of firing of head direction cells in the postsubiculum, which neighbors the hippocampus [14]. How place and direction are coordinated is not yet known. Based on the findings so far, John O'Keefe, author of The Hippocampus as a Cognitive Map and discoverer of place cells, theorizes that language may have evolved as a means to communicate spatial maps [24]. If this is indeed the case, it would lend strong support to spatial representations for language such as Jackendoff's.

Bibliography

[1] P. Agre and S. Rosenschein. *Computational theories of interaction and agency.* MIT Press, Cambridge, MA, 1996.

[2] J. Allen. Natural language understanding. In Paul R. Cohen Avron Barr and Edward A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume IV*, volume 4. Addison-Wesley, 1989.

[3] Richard Wayne Boberg. Generating line drawings from abstract scene descriptions. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering Department, 1973.

[4] Eugene Charniak. Toward A model of children's story comprehension. Technical Report AITR-266, MIT, 1972.

[5] Paul Bloom et al. *Language and Space.* MIT Press, Boston, 1996.

[6] C. Fillmore. A case for case. In *Universals in Linguistic Theory*, pages 1–88. Holt, Rinehart and Winston, New York, 1968.

[7] V. Gregg. *Word frequency, recognition and recall.* Wiley, London, 1976.

[8] Clark H. Herbert. *Space, Time, Semantics, and the Child.* Academic Press, New York/Chicago/San Francisco, 1973.

[9] Berthold Klaus Paul Horn. *Robot Vision.* McGraw–Hill Book Company, Cambridge, Massachusetts, 1986.

[10] R. Jackendoff. Parts and boundaries. *Cognition*, 41:9–45, 1991.

[11] Ray Jackendoff. Toward an explanatory semantic representation. *Linguistic Inquiry*, 7:89–150, 1976.

[12] Ray Jackendoff. *Semantics and Cognition*. MIT Press, Cambridge, MA, 1983.

[13] Ray Jackendoff. *Semantic Structures*. MIT Press, Cambridge, MA, 1990.

[14] JB Ranck Jr JS Taube, RU Muller. Head-direction cells recorded from the postsubiculum in freely moving rats. *J Neurosci*, 10:420 – 435, 1990.

[15] Wolfang Kohler. *The Mentality of Apes*. London : Routledge and Kegan Paul, London, 1927.

[16] P. Kolers and M. von Grunau. Shape and color in apparent motion. *Vision Research*, 16:329–335, 1976.

[17] George Lakoff and Mark Johnson. *Metaphors We Live By*. Chicago : University of Chicago Press, Chicago, 1980.

[18] M. Lenat, D., Prakash and M. Shepherd. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, 6(4):65–85, 1986.

[19] J. Lettvin, H. Maturana, W. McCulloch, and W. Pitts. What the frog's eye tells the frog's brain. *Proc. IRE*, 47:1940–1959., 1959.

[20] J. Macnamara. How do we talk about what we see? 1978.

[21] Marvin Minsky. A framework for representing knowledge. Technical Report AIM-306, MIT, 1974.

[22] Hans Moravec. *Robot: Mere Machine to Transcendent Mind*. Oxford University Press, Oxford, 1999.

[23] V.B. Mountcastle. The world around us: neural command functions for selective attention: The f.o. schmidt lecture in neuroscience 1975. *Neuroscience Research Program Bulletin*, 14:1–47, 1976.

[24] J. O'Keefe and L. Nadel. *The hippocampus as a cognitive map*. Oxford University Press, 1978.

[25] J. Ray. *Consciousness and the Computational Mind*. MIT Press, Cambridge, MA., 1987.

[26] Roger C. Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3(4):pages 532–631, 1972.

[27] Roger N. Shepard and Jacqueline Metzler. Mental rotation of three-dimensional objects. *Science*, 171, 1971.

[28] Jeffrey M. Siskind. Naive physics, event perception, lexical semantics, and language acquisition. Technical Report AITR-1456, MIT, 1993.

[29] Jeffrey Mark Siskind. Visual event classification via force dynamics. In *Proceedings AAAI-2000*, pages 149–155, 2000.

[30] Elisabeth S. Spelke. Principles of object perception. *Cognitive Science*, 14(1):29–56, 1990.

[31] L. Talmy. Force dynamics in language and cognition. *Cognitive Science*, 12(1):49–100., 1988.

[32] S. Ullman. *High-level vision: object recognition and visual cognition*. MIT Press, Cambridge, MA, 1996.

[33] C. von Hofsten. Predictive reaching for moving objects by human infants. *Journal of Experimental Child Psychology*, 30:369–382, 1980.

[34] Deniz Yuret. The binding roots of symbolic ai: a brief review of the cyc project, 1996.