Learning, Using Examples, to Translate Phrases and Sentences to Meanings

by

Adam Davis Kraft

Submitted to the Department of Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

October 2007

Author: . . . .
Department of Electrical Engineering and Computer Science
October 19, 2007

Certified by: . . . .
Patrick Henry Winston
Ford Professor of Artificial Intelligence and Computer Science

Accepted by: . . . .
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee for Graduate Theses

Learning, Using Examples, to Translate Phrases and Sentences to Meanings

by

Adam Davis Kraft

Submitted to the Department of Electrical Engineering and Computer Science

October 19, 2007

In partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Before we can create intelligent systems that exhibit the versatility of the human intellect, we must understand how our command of language enables the uniquely broad scope of our reasoning ability. To pursue such an understanding, we must discover the ways by which language gives rise to representations which, in turn, serve as the building blocks of models that capture constraints and regularities of our environment. The work described in this thesis constitutes a step toward this goal. I have combined aspects of Winston's Arch-Learning methodology with implementations of three powerful representations: Lexical Conceptual Semantics[Jackendoff 1983], Transition Spaces[Borchardt 1993], and Thread Memory[Vaina, Greenblatt 1979], in a system that learns to instantiate semantic descriptions from language based on a sequence of examples. My program, Lance, builds models of the correspondences between parse trees and semantic descriptions by generalizing from a sequence of pairs of sentence fragments and descriptions. Additionally, counterexamples of one type of correspondence model may be generated from examples of similar models in order to facilitate learning by near miss. The result is that my system can learn such constraints as *in order for a sentence to convey a transition, it must contain a verb that means either "change," "appear," or "disappear."* In this work I developed an approach based on presentation of parse trees paired with instantiated representations and the Arch-Learning paradigm, and implemented Lance, a 12,000 line Java program. I demonstrated that from a training sequence of 95 examples, Lance learned 27 models of THINGS PARTS, PLACES, PATHELEMENTS, TRAJECTORY-SPACES, TRANSITION-SPACES, CAUSES, and IS-A relations.

Acknowledgments

I am indebted to Patrick Winston for providing the inspiration and support for this work, and for giving me the freedom to pursue it in my own direction. Patrick Winston inspired my passion to understand human intelligence, and taught me everything that I know about effective communication.

I owe special thanks to my academic advisor, Paul Gray, whose wisdom and guidance have been invaluable to me during my years at MIT.

I thank Mark Finlayson, Sam Glidden, Mark Seifter and the many participants in the Genesis Project research group for helping me with this thesis and for sharing their knowledgeable enthusiasm for Artificial Intelligence.

Finally, I thank my parents for investing so much of their time and energy in my education, and for instilling in me their belief that education is the most important investment that one can make.

This research was supported in part by the National Science Foundation, Award Number IIS-0646933.

Contents

1. Introduction

Before we can create intelligent systems that exhibit the versatility of the human intellect, we must understand how our command of language enables the uniquely broad scope of our reasoning ability. Pursuit of such an understanding is guided by two fundamental questions:

- How does language empower us to construct models that exhibit the constraints and regularity inherent in our environment?
- How does language orchestrate symbolic and sub-symbolic reasoning abilities to enable multi-modal reasoning?

My particular goal is to shed light on the question of how language gives rise to models. The first step toward this goal is to relate the structure of language to its meaning. Specifically, I seek ways to connect sentences and phrases of natural language to world-modeling representations. To this end I have implemented Lance, a program that learns how to construct descriptions, cast in Borchardt's Transition Spaces [Borchardt 1993], Jackendoff's Lexical Conceptual Semantics (LCS) [Jackendoff 1983], and other powerful representations, from English text. Lance is a key component of the Gauntlet System, an ongoing enterprise that addresses both the modeling question and the sub-symbolic question. The Gauntlet System aims to discover the roles of language and vision in concept formation. Such an endeavor requires that we understand language not only in its ability to evoke symbolic representations, but in its capacity to marshal sub-symbolic agents in order to coordinate language and vision.

One way to relate the structure and meaning of language is to use the phrase structure and lexical information of sentences to extract unambiguous semantic descriptions from those sentences. LCS and Transition Space exemplify the type of representational framework that can convey such semantic descriptions concisely and with wide coverage. Figure 1-1 depicts two prepositional phrases and their semantic descriptions in LCS.

on the table	$\rightarrow$	$\left[_{\text{Place}}\ \text{ON}\left(\left[_{\text{Thing}}\ \text{TABLE}\right]\right)\right]$
to the table	$\rightarrow$	$\left[_{\text{Path}}\ \text{TO}\left(\left[_{\text{Place}}\ \text{AT}\left(\left[_{\text{Thing}}\ \text{TABLE}\right]\right)\right]\right)\right]$

Figure 1-1: Semantic Descriptions of Prepositional Phrases

In order to extract the semantic descriptions shown in Figure 1-1 from their corresponding sentence fragments, we can devise rules that map from the language domain onto the semantic domain. Such rules can stipulate, for example, that if a phrase is a prepositional phrase, and the preposition is the word *on*, and the object described by the noun phrase is a physical thing, then the semantic description of the prepositional phrase is a place. If, however, the object described by the noun phrase is a physical thing but the preposition is *to*, then the description is a path. That such rules seem obvious is a testament to the agility with which the human mind can acquire them from context. Codifying such rules manually, however, can be a tedious and error-prone process because of subtle interactions between language structures and semantic forms. Lance acquires the rules that generate semantic descriptions corresponding to language structures automatically, by learning from a sequence of language-description pairs such as those of Figure 1-1. Once learned, the rules can generate semantic descriptions of novel sentences and fragments in the context of the Gauntlet System.

My work on Lance differs from previous work in natural-language parsing because it is focused on extracting the meaning—not just the structure—of language. Lance differs from previous work in its own domain of semantic interpretation because it relies on a training process based on Winston's Arch-Learning methodology [Winston 1970]. It is this learning methodology that enables Lance to learn the rules for constructing semantic descriptions corresponding to language structures. With Arch Learning it becomes possible for Lance to learn models that specify how parse trees map onto descriptions. An example of such a model is shown in Figure 1-2. A detailed explanation of the type of model shown in Figure 1-2 is given in Section 3. Lance's ability to learn the rules that map parse trees to descriptions gives it an advantage over efforts to manually catalog the rules, whose number and formidable complexity imposes severe limitations on the utility of a manual approach. In this thesis I present the design and implementation of Lance, motivated by examples and by the requirements imposed upon it by its role in the Gauntlet System.

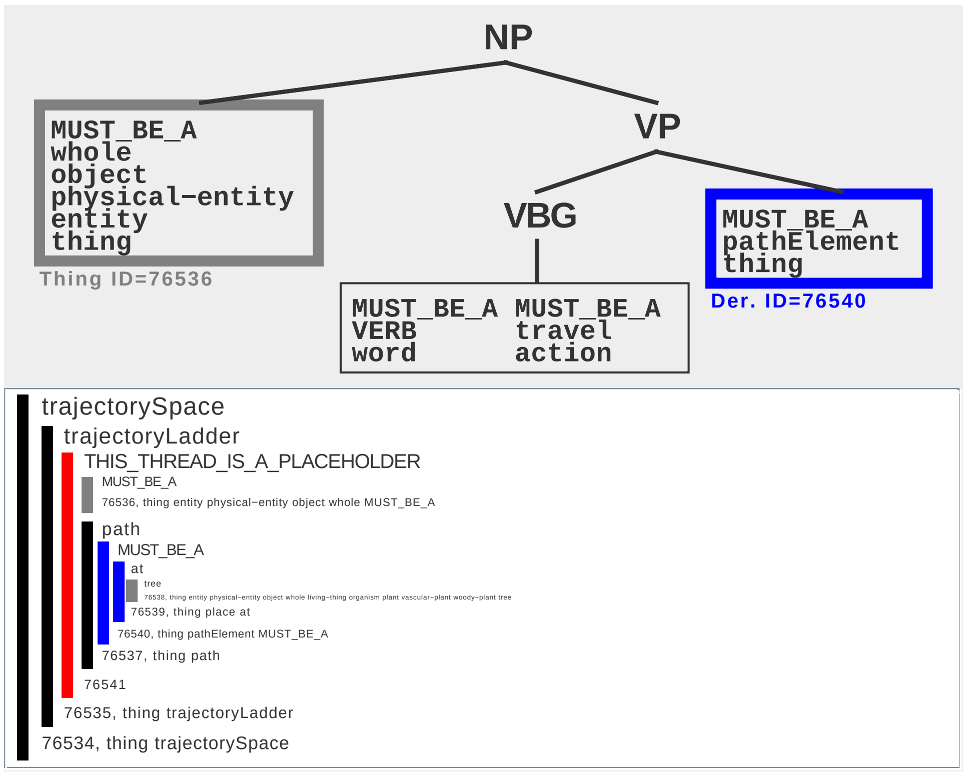

NP
MUST_BE_A
whole
object
physical-entity
entity
thing
Thing ID=76536
VP
VBG
MUST_BE_A
VERB
word
MUST_BE_A
travel
action
MUST_BE_A
pathElement
thing
Der. ID=76540
trajectorySpace
trajectoryLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
76536, thing entity physical−entity object whole MUST_BE_A
path
MUST_BE_A
at
tree
76538, thing entity physical−entity object whole living−thing organism plant vascular−plant woody−plant tree
76539, thing place at
76540, thing pathElement MUST_BE_A
76537, thing path
76541
76535, thing trajectoryLadder
76534, thing trajectorySpace

Figure 1-2: Model of a parse tree mapping to a TRAJECTORY-SPACE

1.1 Modeling Transformations between Language and Representation

Prior art in the domain of semantic extraction relied on carefully designed patterns to match against highly constrained language in order to determine what type of semantic frame best suited a phrase, and which variables within that frame correspond to which lexical items in the phrase. Such hand-made patterns characterize the BridgeSpeak parser [Miller, C. 2006, Larson 2003] which was equipped to extract trajectory, transition, and classification frames from sentences.

In practice, even simple semantic structures have some unusual manifestations and special-case exceptions in language, as exemplified by the discussion in Section 2 of how direction words influence whether PATHELEMENTS may be used to indicate PLACES. Designing elaborate rules to account for such special cases is a tedious enterprise, which becomes increasingly prone to error as the number of different representations supported by a system grows. Because the Gauntlet System may eventually support a dozen or so representations [Winston 2007], it is critical to alleviate the burden of designing rules by hand.

An important observation about designing rules by hand is that the process depends on thinking about specific example cases that the system designers know to embody certain semantics. The thought process is roughly one of identifying an example that implements a semantic category, and then generalizing it to form a template that matches similar examples. The first critical step—identifying an example and its semantics—seems to come quite naturally. It is the generalizing step that is time-consuming and prone to human error. In this work, therefore, I automate the process of generalizing from examples to templates.

Intuitively, we know that the prepositions in Figure 1-1 convey the critical information that distinguishes the semantic description of a place from that of a path, and that the type described by the noun phrase is not important, provided that it is a physical object. Because the only difference between "to the table" and "on the table" is the preposition, however, it is a simple matter to convey the intuition about prepositions systematically. In terms of the Arch-Learning paradigm[Winston 1970], "on the table" constitutes a near miss with respect to an evolving model of path prepositional phrases and their associated semantic descriptions. Likewise, "to the table" constitutes a near miss with respect to the place-phrase model. In order to convey the other part of our intuition, that the noun phrase may represent any physical object, we may present further examples of places and paths such as those of Figure 1-3. The objects represented by the noun phrases in Figure 1-3 are sufficiently different from a table that the system may generalize its models of place and path phrases to reflect the intuition that any physical object is acceptable.

on the tree $\rightarrow$ $\left[_{\text{Place}}\ \text{ON}\left(\left[_{\text{Thing}}\ \text{TREE}\right]\right)\right]$

to the ant $\rightarrow$ $\left[_{\text{Path}}\ \text{TO}\left(\left[_{\text{Place}}\ \text{AT}\left(\left[_{\text{Thing}}\ \text{ANT}\right]\right)\right]\right)\right]$

Figure 1-3: Semantic Descriptions of Prepositional Phrases, Permitting Generalization

It is in precisely this way that Lance learns how to build semantic descriptions from language. A single phrase-description pair serves as an initial model. Examples of different semantic categories that meet criteria of structural similarity and that qualify as near misses permit specialization of the model. Examples that relax constraints on a part of the semantics of the model permit generalization.

1.2 Representations, Models, and Concepts

The words *representation, model,* and *concept* have specific meanings in the context of the Gauntlet System. A standard definition of these three words is essential to further discussion of the Gauntlet System and my component, Lance, because everyday usage permits us to bend these terms in convenient but potentially misleading ways.

As noted by Patrick Winston[Winston 2007,1], the principle of concept formation is familiar to us in many contexts: from infancy we learn what it means to be happy, tired, or hungry. As we mature we gain knowledge of activities like children's games, and of abstract categories such as vegetable and animal. We then begin to recognize impalpable things such as love, honor, and mystery. As we pass into adulthood we master abstraction, enabling us to understand formalisms such as calculus. With such a repertoire of meanings, it is not surprising that *concept* has acquired status among the most notorious of the so-called suitcase words, whose tidy outward appearance belies the jumble we are able to cram within.

The definition of *concept* in the context of the Gauntlet System is aimed at bringing important technical questions to light: **a concept is a complex, cross-modal model that crystallizes out of experience** [Winston 2007, 1]. Here, the terms *complex* and *cross-modal* apply specifically to the way in which concepts unite disparate ways of perceiving or understanding a phenomenon, for example, through a combination of a verbal description and a visual scene. A concept is complex in that to have a concept means to retain the ability to interpret the phenomenon to which it pertains in more than one way. A concept also balances this variety in interpretation with uniformity, by allowing connections between different interpretations to expose new constraints and regularities. Such cross-modal constraints and regularities would not have emerged from any lone interpretation. That concepts crystallize out of experience means that concept formation is a process of iterative refinement based on observations, rather than a one-shot learning exercise or hard-wired knowledge.

The notion of a concept is rooted in the notion of a model. Science and engineering models express important regularities and constraints of a domain. A model of objects that participate in trajectories, for example, might capture the regularity that birds can fly, whereas elephants cannot. Models, in turn, rely on aptly-chosen representations. In order to model a particular regularity or constraint observed in the environment, it is necessary to employ a representational framework capable of capturing the observation. Choosing or devising such a framework is a creative endeavor that is the essence of science and engineering.

Although they are a necessary component of models, expressive representations are insufficient as models in and of themselves. Experience is the key to forming models from representations: although the representational framework chosen for trajectories may be capable of expressing the event in which an elephant flew up a flagpole, the absence of elephants from observed aerial trajectories may prove itself an important constraint to include in the trajectory model. In this way, representations are like elastic wrappers that conform to real-world phenomena, evolving models that accentuate the important aspects of those phenomena.

1.3 The Gauntlet System

If we are to understand human intelligence, we must understand our linguistic ability, which clearly differentiates human reasoning from the competencies of our closest primate cousins. It is this facility to use words, phrases, and sentences that allows us all to benefit from a wealth of recorded human experience, to form insightful analogies, and to bring many observations from disparate sensory modes to bear on concept formation.

Because it is clearly our rich use of symbols that sets us apart from other primates, our language abilities might appear to operate in a strictly symbolic domain. Any attempt to account for language abilities based on this appearance, however, would find itself mired in a host of difficulties, arising from the inherent ambiguity of many words and phrases. Behind language's symbolic facade, it synchronizes closely with sub-symbolic faculties in imagination-stimulating loops [Winston 2007, 2]. A successful approach to understanding language's contribution to intelligence must therefore focus on the exchanges that occur between the symbolic reasoning modes of language, and the sub-symbolic perceptual and motor apparatus upon which it relies.

The MIT CSAIL Genesis Group has undertaken the challenge to discover how language and vision cooperate to form concepts through the Gauntlet System[Winston 2007, 1]. The work described in this thesis is an integral part of this research. The Gauntlet System is organized conceptually as a bank of experts, each listening either to a stream of language or to a stream of visual input. Each expert selectively analyzes items appearing in its stream that pertain to its domain of expertise, and ignores the items that it does not recognize. Examples of domains in which particular experts operate are Jackendoff's trajectories, Borchardt's Transition Spaces, and Vaina's and Greenblatt's Thread Memories, which are discussed in Section 4. The experts' analyses may result in state changes in memories belonging to each expert. Each memory, in turn, is accessible to a global cross-representation memory, in which the linkages of nascent concepts will form.

My contribution to the Gauntlet System, Lance, is the first stage of recognition used by each language expert. Lance occupies the boundary between the language stream and each language expert's higher-level processing center, as depicted in Figure 1-4. As sentences and fragments enter the system in the form of text, Lance attempts to instantiate semantic structures corresponding to those sentences and fragments. If the attempt is successful, Lance presents the resulting semantic structures to the expert-specific processing machinery.

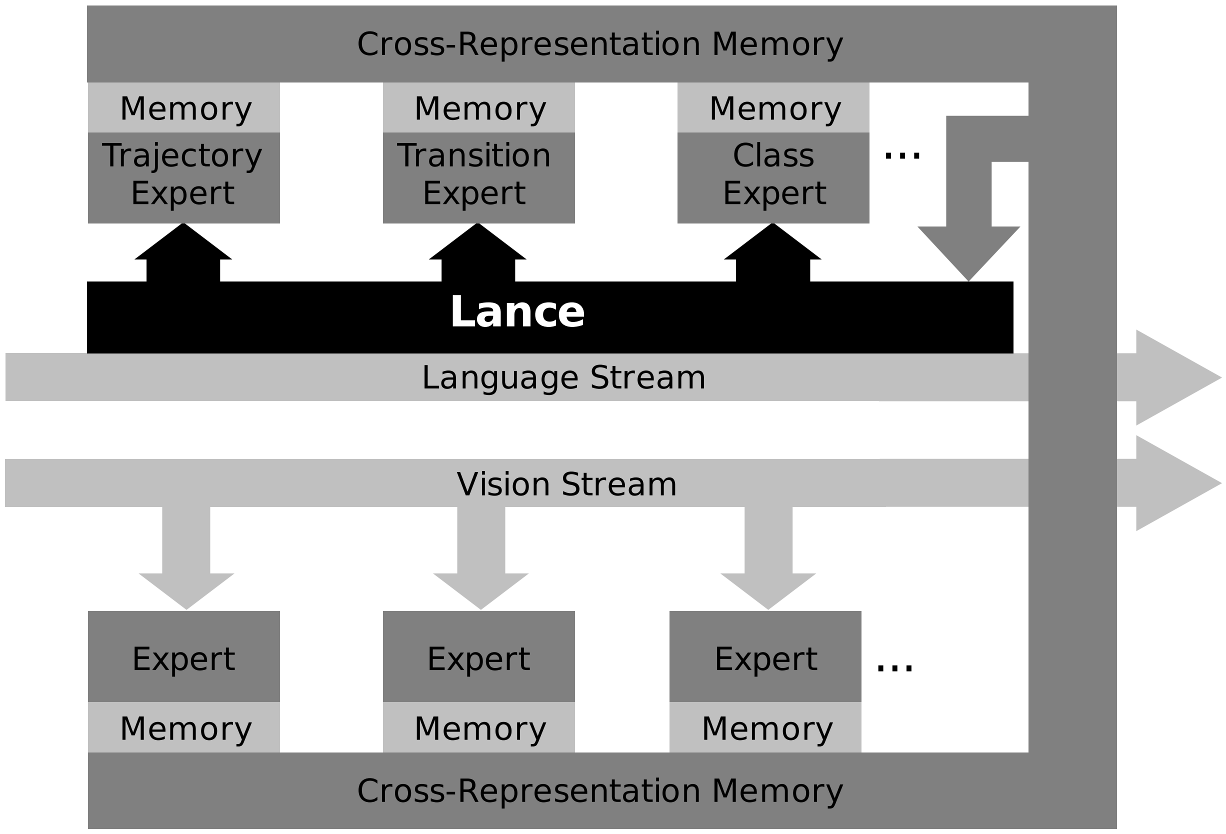


Figure 1-4: Gauntlet System Organization

1.4 Overview

This thesis is organized in six sections.

Section 2 is devoted mainly to examples that played a crucial role in motivating the design of Lance, and provided the most direct way to delineate the problem domain in this thesis. Specifically, these examples provide a means to address the following questions:

- How do representational frameworks manifest as sentences and sentence fragments?
- What methods have been used to extract representations from sentences, and what are their limitations?
- Given a proposed method for extracting representations, what types of misbehavior might we expect, and what modifications would correct the problems?

In Section 3 I explicitly catalog the design parameters and implementation details illustrated by example in Section 2, and present the design of Lance.

Section 4 is an overview of the state of the language art as far as it influences this work. I present a brief selective survey of the science of natural-language parsing, along with a description of the parsing technology that Lance uses. I also describe several of the representational frameworks developed by pioneers in the realm of semantics.

In Section 5 I motivate future inquiry by discussing aspects of the implementation that point toward new challenges.

Section 6 underscores my contributions in this thesis.

2. Motivation By Example

This section presents examples of the types of problems that Lance solves in order to motivate a discussion of the design parameters that address those problems. I have taken this approach for two reasons:

- I used example cases routinely to focus my design effort. This strategy helped me parameterize a formidable challenge—that of extracting descriptions cast in a yet-unspecified ontology, comprised of many disparate representational frameworks, from inherently ambiguous language—into manageable subgoals. Taken out of context, these subgoals might seem arbitrary, but within the context of the example cases the motivation behind the design choices is clear.
- Examples often constitute the most efficient way to evaluate a design choice. Given a nebulous problem domain, it is often either prohibitively difficult or deceivingly easy to justify or to criticize a design decision. Attempts to analyze a problem in depth before it has been observed in the form of a concrete example can leave a design effort stymied. To ward off such "analysis paralysis," I have chosen to motivate my design decisions with concrete examples.

Problems may befall a design methodology tailored to specific example cases. The obvious pitfall is the scenario in which the examples fail to capture nuances of the problem domain, so that the design is doomed to produce little more than a toy system in implementation. Another problem with example-driven design, articulated by David Marr in "Artificial Intelligence—A Personal View"[Marr 1976], is that this methodology may preclude discovery of any concise or elegant model to account for the observed phenomena. Marr called those contributions to AI that were rooted in implementation, rather than elegant theories, *Type-Two* contributions, and warned of the ideological sacrifice inherent in resorting to Type-Two without first searching for elegant *Type-One* alternatives.

Although Lance falls squarely in the Type-Two class, I believe that this compromise is warranted. Because Lance serves to bootstrap development of the Gauntlet System, an expedient solution to the representation-extraction problem will enable progress in concept modeling, perhaps leading to a Type-One discovery. Furthermore, as I illustrate in this section, a Type-Two approach can expose new ways to combine existing work, as exemplified by my use of Winston's Arch-Learning methodology to develop models of the correspondences between language and representation. To address the concern that examples will certainly fall short of capturing some nuances of the language-representation problem, I demonstrate an iterative refinement process in this section, to suggest that designing to fit example scenarios is a practical and powerful technique, whose use is justified until we are overwhelmed by elegant theories of language and representation.

2.1 Three Representational Frameworks

The ontology that the Gauntlet System will use to describe events, states and relations that language can express is not yet specified. One of the goals of the Gauntlet research endeavor is to discover a small group of representational frameworks that offer the best coverage of the domain spanned by both language and vision. Leaving this problem completely unspecified would be unacceptable, however, from the perspective of developing a system to extract descriptions from language. I have therefore chosen to focus on three frameworks that exemplify the Gauntlet ontology. The three frameworks are Jackendoff's Lexical Conceptual Semantics (LCS), Borchardt's Transition Spaces, and Viana's and Greenblatt's Thread Memories.

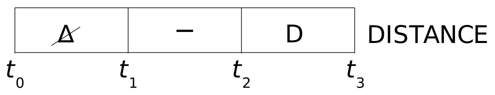
LCS provides a convenient way to represent spacial and causal relations. The most basic element of LCS is a THING, which represents a physical or abstract object. Building on this most basic element are PLACES, PATHELEMENTS[1], PATHS, TRAJECTORIES, and EVENTS. The following exemplifies an LCS description:

$$\left[_{\text{Event}}\ \text{GO}\left(\left[_{\text{Thing}}\ \text{SQUIRREL}\right], \left[_{\text{Path}}\ \text{UP}\left[_{\text{Thing}}\ \text{FLAGPOLE}\right]\right]\right)\right]$$

Transition Space captures variables changing over time. By aligning descriptions expressed in Transition Space, a program can systematically reconstruct causal relations. A Transition-Space rendering of the following sequence of events: “The distance did not change. The distance

1 PATHELEMENTS are not part of Jackendoff's LCS, but are added here as a convenient way to represent PATHS with multiple PATHFUNCTIONs.

decreased. The distance vanished."



Thread Memories capture categorical information flexibly and conveniently. Threads are acyclic chains of semantic symbols that are linked to keys. A Thread Memory of a particular mallard duck follows.

mallard → living → animal → bird → duck → mallard

Each of the three exemplary representational frameworks is detailed in Section 4.

2.2 Meta-Representation and Notation

Lance relies on a meta-representation to describe the representational frameworks that, in turn, describe the meaning of phrases and sentences. The advantages of using such a meta-representation are threefold: it simplifies code maintenance, it enables structural comparison between disparate representations, and it provides a convenient way to visualize representational frameworks. A meta-representation simplifies code maintenance because it obviates hard-wiring the mechanisms specific to each representation into the program. By the same token, meta-representations allow improvements to apply to all representations without repeated effort on behalf of each representation. The ability to execute structural comparisons without regard to the representational mechanics of the particular descriptions involved is essential to Lance's learning mechanism, as expounded later in this section. Finally, the constituents of the meta-representation are presented here, rather than in the section devoted to implementation, because they lend themselves to a concise notation with which to illustrate the examples in this section.

The meta-representation I employed was designed by Patrick Winston [Winston et al. 2003]. It consists of a hierarchy of structure types, in which the overarching type is `Thing`, which subsumes the special types `Derivative`, `Relation`, and `Sequence` as depicted in Figure 2-1. I adhere to the typographical convention in order to distinguish between meta-representational items from items such as THINGS in the LCS framework.

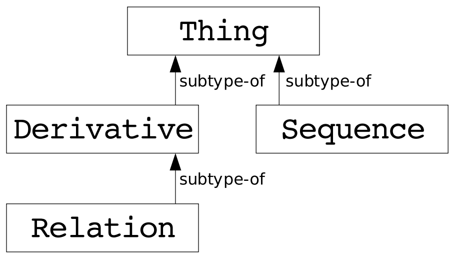


Figure 2-1: Meta-Representation Type Hierarchy

Instances of the `Thing` type have a label and zero or more instances of Thread Memories. A `Derivative`, so called because it describes a property derived from a `Thing`, contains a `Thing` in addition to its own label and bundle. `Relation`s contain two `Thing`s and capture the connection between them. `Sequence`s contain a list of `Thing`s and capture the connections among them. Figure 2-2 illustrates how Jackendoff's and Borchardt's frameworks map onto the meta-representation.

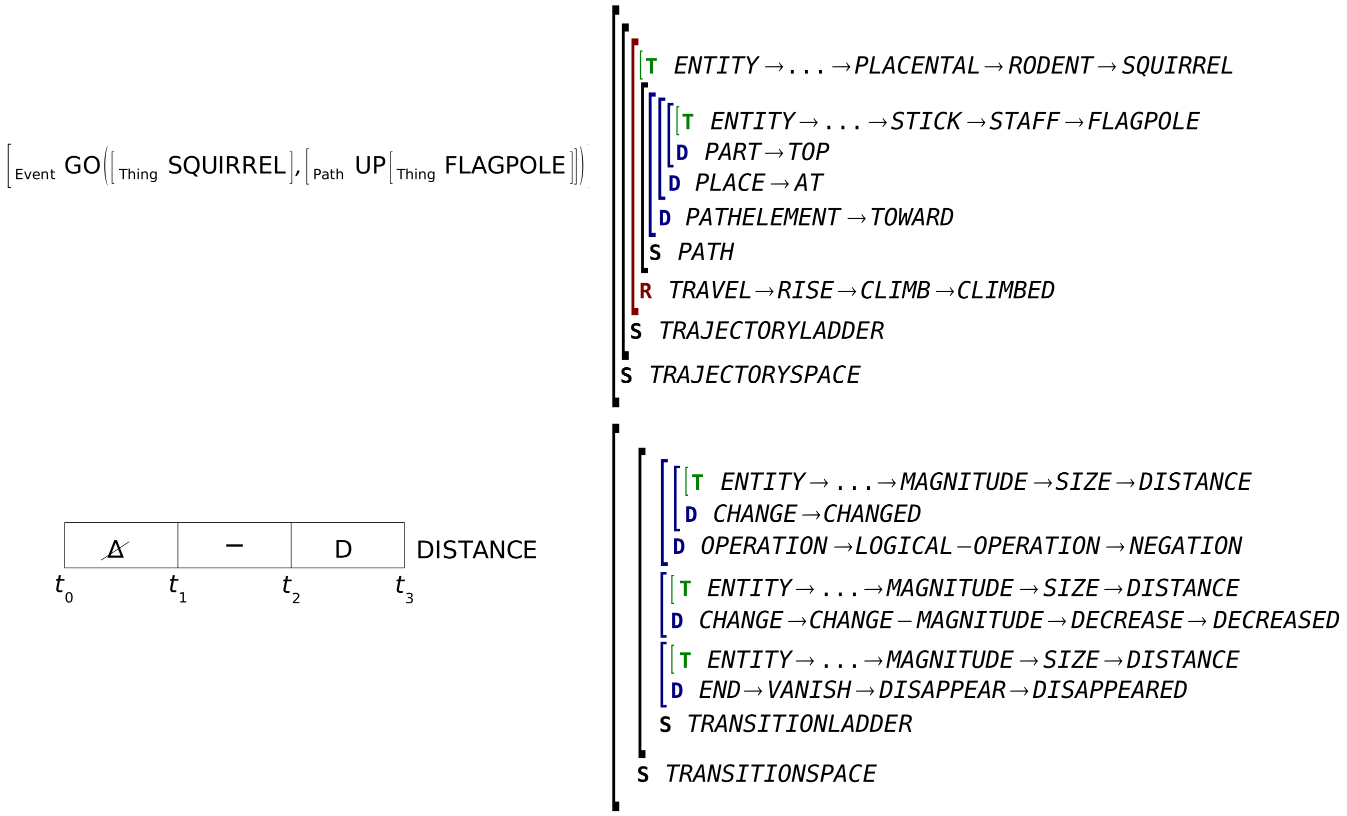

Event GO Thing SQUIRREL Path UP Thing FLAGPOLE
T ENTITY → . . . → PLACENTAL → RODENT → SQUIRREL
T ENTITY → . . . → STICK → STAFF → FLAGPOLE
D PART → TOP
D PLACE → AT
D PATHELEMENT → TOWARD
S PATH
R TRAVEL → RISE → CLIMB → CLIMBED
S TRAJECTORYLADDER
S TRAJECTORYSPACE
D
DISTANCE
t0 t1 t2 t3
T ENTITY → . . . → MAGNITUDE → SIZE → DISTANCE
D CHANGE → CHANGED
D OPERATION → LOGICAL − OPERATION → NEGATION
T ENTITY → . . . → MAGNITUDE → SIZE → DISTANCE
D CHANGE → CHANGE − MAGNITUDE → DECREASE → DECREASED
T ENTITY → . . . → MAGNITUDE → SIZE → DISTANCE
D END → VANISH → DISAPPEAR → DISAPPEARED
S TRANSITIONLADDER
S TRANSITIONSPACE

Figure 2-2: Meta-Representation rendering of LCS and Transition-Space descriptions

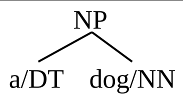
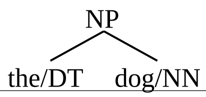
As illustrated in Figure 2-2, the renderings of Transition Spaces and trajectories in the meta-representation have both superficial symbolic differences and structural differences from the representations upon which they are based. Symbolic differences, such as the substitution of TRAVEL for GO in the trajectory description result from using WordNet, an English lexical database[Fellbaum 1998], to derive threads corresponding to nouns, verbs, and adjectives. Use of WordNet is discussed further in Section 3. Inclusion of lexical information pertaining to the verb *climb* in the trajectory is redundant but does not compromise the simplicity LCS because of the way this information is captured in a thread. The most notable structural difference is the introduction of PATH-ELEMENT to the basic LCS. PATH-ELEMENTs represent modular segments that make up PATHS, clarifying the notion of PATHS with more than one PATH-FUNCTION[Bonawitz 2003].

2.3 Setting the Stage

Presented here are examples of the most basic language-description pairs that Lance must recognize. I emphasize several troublesome scenarios in which subtle variations in parse-tree structure or other features complicate the matter of semantics extraction.

2.3.1 THINGS, PLACES, PARTS, and PATHELEMENTS

Table 2-3 introduces several ways in which noun phrases give rise to THINGS, the most basic elements of the representational framework.

Fragment / Parse Tree	**Semantic Description**
a dog 	[T *ENTITY → . . . → CARNIVORE → CANINE → DOG*
the dog 	[T *ENTITY → . . . → CARNIVORE → CANINE → DOG*

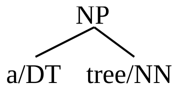
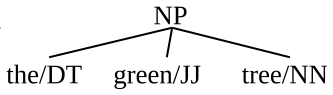
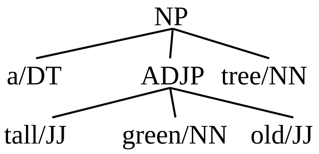
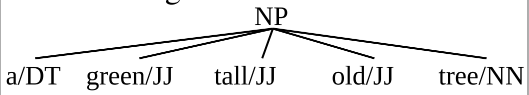
a tree 	[T ENTITY→...→VASCULAR–PLANT→WOODY–PLANT→TREE
the green tree 	[T ENTITY→...→VASCULAR–PLANT→WOODY–PLANT→TREE FEATURE→GREEN
a tall green old tree  NP a/DT ADJP tree/NN tall/JJ green/NN old/JJ	[T ENTITY→...→VASCULAR–PLANT→WOODY–PLANT→TRE FEATURE→GREEN FEATURE→TALL FEATURE→OLD
a green tall old tree 	[T ENTITY→...→VASCULAR–PLANT→WOODY–PLANT→TRE FEATURE→GREEN FEATURE→TALL FEATURE→OLD

Table 2-3 Representation of THINGS

As illustrated by the difference between "a tall old green tree" and "a green tall old tree" in Table 2-3, slight variations in word choice and order can significantly alter the parse tree structure. The variation in tree structure is an artifact of the Wall Street Journal corpus [Marcinkiewicz, et al. 1993] used to train the Stanford Parser, and does not lend itself to any concise model. Instead, it often appears that the complex interactions among productions within the parser's PCFG provide the only means to account for the observed variation in tree structure. A system that maps these parse trees onto semantic descriptions, in which any variation must add value to the interpretation, must either recognize semantic uniformity despite surface variation or provide a means to transform the parse trees to reduce spurious variation.

Table 2-4 depicts the opposite scenario in which the syntax of "to the left of <Noun Phrase>" is not ambiguous, in that the parser will always produce a tree like the one shown. The semantics of this type of phrase is sensitive to context, however, because such a phrase may just as easily refer to a PATHELEMENT, e.g. "The dog ran to the left of the tree," as a PLACE, e.g. "To the left of the desk stood a potted plant." To distinguish between the PATHELEMENT and PLACE interpretations of such constructions as "to the left of the table" requires knowledge of the surrounding context. This has an important implication for semantics extraction: any purely bottom-up recognition process that assigns semantic descriptions to subordinate parse trees first must occasionally account for several potential semantic interpretations of those trees, because it would not yet have the context information required to choose the appropriate interpretation.

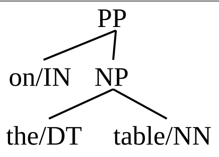
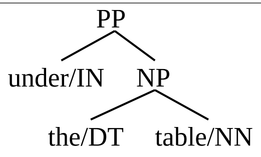
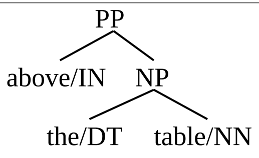
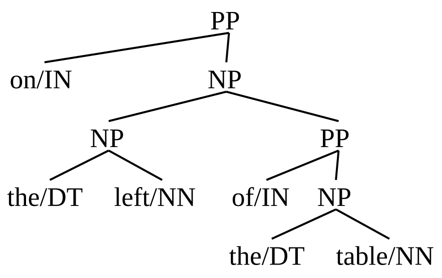
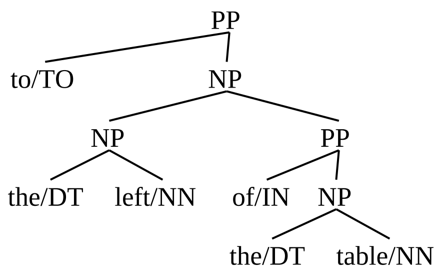
Fragment / Parse Tree	Semantic Description
on the table 	[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→ON*
under the table 	[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→UNDER*
above the table 	[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→ABOVE*
on the left of the table 	[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→LEFTOF*
to the left of the table 	[[[[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→LEFTOF* D *PATHELEMENT→TO* S *PATH* D *PLACE→ON* **Alternative:** [[[T *ENTITY→...→FURNISHING→FURNITURE→TABLE* D *PLACE→LEFTOF* D *PATHELEMENT→TO*

Table 2-4: PLACES and Semantically Ambiguous Prepositional Phrases

To shed more light on the type of semantic ambiguity present in “to the left of the table,” consider examples in which *left* is replaced with a superficially similar spatially-descriptive word, *top*:

The bird sat to the left of the tree.

The bird flew to the left of the tree.
The bird flew to the top of the tree.
*The bird sat to the top of the tree.

The difficulty with "The bird sat to the top of the tree" seems to arise from a quality of words such as *top, bottom,* and *underside,* that necessarily identifies a part of an object, whereas words such as *left, right,* and *North*, refer to directions relative to a reference object. Direction words in a PATHELEMENT construction cause the PATHELEMENT to be of a directional nature regardless of the preposition used. As noted in Section 2, directions are paths in which the reference object does not fall on the path itself. It stands to reason that directions can also identify abstract places because every place conforming to the following production, in which *x* is a direction, shares a common spatial property relative to the reference object:

$$[\text{PLACE}] \rightarrow \left[_{\text{Place}}\ \text{ON}\left(\left[_{\text{Path}}\ x\right]\right)\right]$$

Words such as top, bottom, and side, on the other hand, most often refer to concrete parts of objects. When these words participate in PATHELEMENT constructions, they are not restricted to directional paths but may refer to bounded paths or routes as well. Bounded paths and routes appear to be less apt to identify PLACES per se, because the absolute position relative to the reference object is not optional when using a bounded or route PATHELEMENT to identify a place. This is borne out by example:

*The bird perched from the top of the tree.
The bird perched three feet from the top of the tree.

Furthermore, by forcing PATHELEMENTS built upon *part* words to be of the direction type, the absolute position along the path again appears optional, as the following pair of examples illustrates:

*The bird perched from the top of a stone.
The bird perched toward the top of a stone.

As a final item of evidence for the unique ability of directional PATHELEMENTS to evoke PLACES, consider the example of forcing a direction word into a part role by using the word *part* explicitly:

*The lamp was to the left part of the desk.
The lamp was on the left part of the desk.

When the reference to a part becomes explicit, resulting in a bounded path rather than a directional path, PATHELEMENTS containing the direction words seem to lose the flexibility to refer to PLACES without further description of the position along the contained path.

To encapsulate the notion of a *part* in the meta-representation requires little effort because parts are naturally expressed in terms of `derivative`s in the meta representation. Rather than expanding the PATHFUNCTION vocabulary to account for directions, directions have been implemented in terms of PATHELEMENTS and the abstract places to which they refer, e.g. LEFTOF and RIGHTOF. Table 2-5 depicts examples of PARTS and PATHELEMENTS.

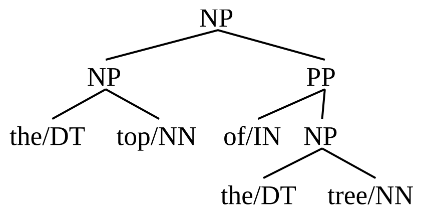
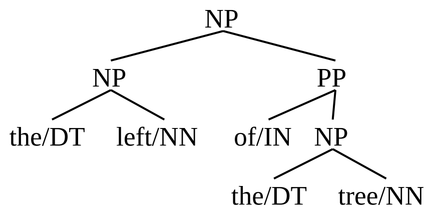
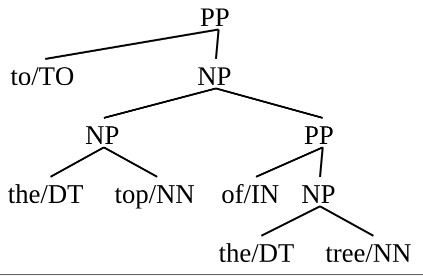
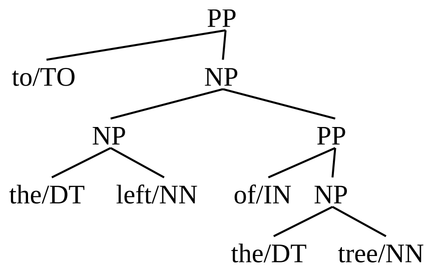
Fragment / Parse Tree	Semantic Description
the top of the tree 	T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PART → TOP
the left of the tree 	**Unclear Interpretation** T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PART → LEFT ?
to the top of the tree 	T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PART → TOP D PLACE → AT D PATHELEMENT → TO
to the left of the tree 	**Counterexample:** T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PART → LEFT D PLACE → AT X D PATHELEMENT → TO **(Refer to Table 2-4 for positive examples)**

Table 2-5: PARTS and PATHELEMENTS

The examples of Table 2-5 illustrate an important caveat pertaining to the semantics of *part* and *direction* words. It is unclear whether constructions of the form "the ___ of the <THING>" should be considered as identifying parts of the referenced THING when the blank is filled by a direction word. Of the following two sentences:

*?*The left of the room is painted blue.

The left side of the room is painted blue.

I prefer the form in which *side* or some other word indicating a part is stated explicitly. Given no additional context, however, I take the form in which no part of the room is mentioned explicitly to mean that a *leftward part* of the room is painted blue. Other native speakers of English may be more or less inclined to agree.

That the counterexample given in Table 2-5 for "to the left of the tree" is unacceptable follows from the nature of direction vs. part words. As noted, the flexibility of PATHELEMENT-type constructions formed with direction words like *left* and *right* to represent both PATHELEMENTS and PLACES seems to arise from the way in which direction words cause the paths to be unbounded. The interpretation of "to the left of the tree" in table 2-5 is a bounded path, however, so it should be excluded as a possible interpretation.

Table 2-6 illustrates yet another difficulty in extracting PATHELEMENTS: whereas often items representing a single element of semantic structure correspond to a unified syntactic structure, this is not always true. When the fragment "away from the tree" is included in a verb phrase, it produces two distinct branches of that verb phrase's parse tree: one for the prepositional phrase "from the tree" and another for the particle *away*. Further complicating matters is the fact that the Stanford Parser cannot produce a root parse tree with multiple branches, so that when "away from the tree" is parsed alone as a fragment, the parse tree necessarily differs from that of a sentence that includes this fragment. This precludes training a system to recognize this fragment out of context. The design of Lance addresses both the context-sensitivity problem and the disjoint-parse-tree problem simultaneously by requiring that "away from" and other constructions that suffer from these problems be learned as integral parts of semantics with broader scope, such as Trajectories.

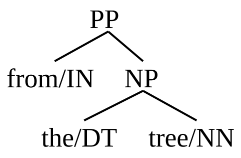
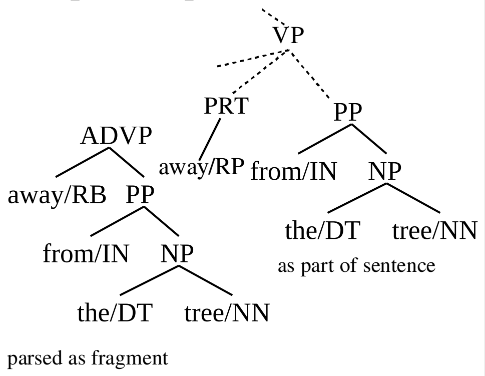
Fragment / Parse Tree	Semantic Description
from the tree 	[[T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PLACE → AT D PATHELEMENT → FROM
away from the tree **Two possible parse trees:**  VP PRT PP ADVP away/RP from/IN NP away/RB PP the/DT tree/NN from/IN NP as part of sentence the/DT tree/NN parsed as fragment	[[T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE D PLACE → AT D PATHELEMENT → AWAYFROM

Table 2-6: More PATHELEMENT structures

2.3.2 TRAJECTORY- and TRANSITION-SPACES

As noted in Section 2.2, a TRAJECTORY SPACE is roughly equivalent to certain EVENTs as described by LCS, and a TRANSITION SPACE corresponds to Borchardt's framework of the same name. The structures have been cast in the meta-representation so that they have clearly analogous parts, for reasons that do not pertain directly to this implementation but that concern the expert components of the Gauntlet System. Table 2-7 introduces basic correspondences between TRAJECTORY SPACEs, TRANSITION SPACEs, and the simplest syntactic structures that convey these forms.

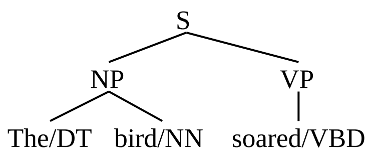
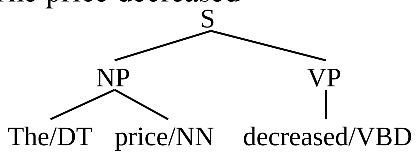
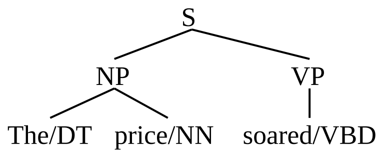
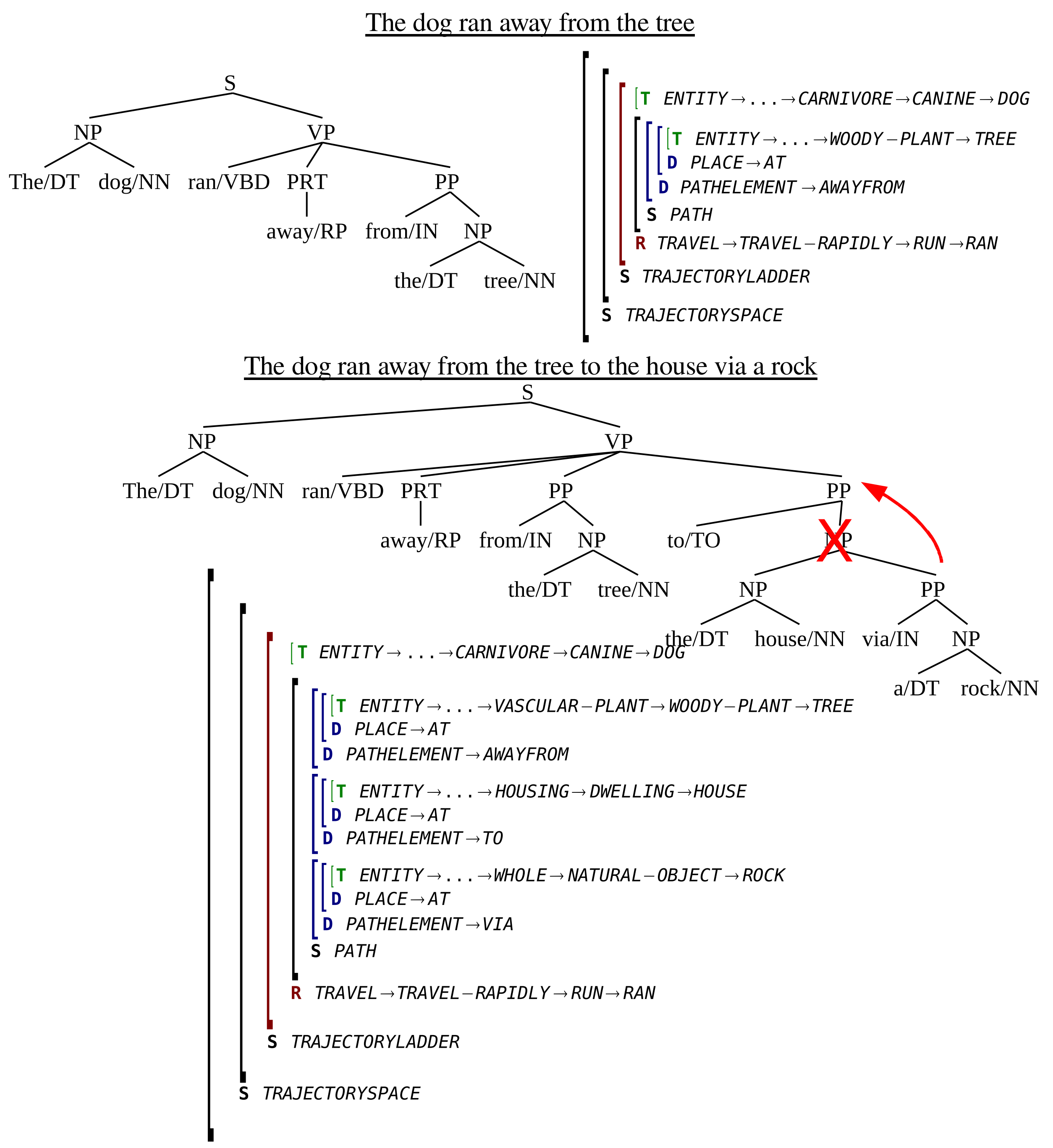
Sentence / Parse Tree	Semantic Description
The bird soared 	[T ENTITY → . . . → CHORDATE → VERTEBRATE → BIRD [S PATH R TRAVEL → FLY → SOAR S TRAJECTORYLADDER S TRAJECTORYSPACE
The price decreased 	[T ENTITY → . . . → VALUE → MONETARY_VALUE → PRICE D CHANGE → CHANGE – MAGNITUDE → DECREASE → DECREASED S TRANSITIONLADDER S TRANSITIONSPACE
The price soared 	[T ENTITY → . . . → VALUE → MONETARY – VALUE → PRICE D CHANGE → . . . → GROW → RISE → SOAR S TRANSITIONLADDER S TRANSITIONSPACE **Alternative:** [T ENTITY → . . . → VALUE → MONETARY – VALUE → PRICE [S PATH R TRAVEL → FLY → SOAR S TRAJECTORYLADDER S TRAJECTORYSPACE

Table 2-7: Basic TRAJECTORY- and TRANSITION-SPACE correspondences

As shown in Table 2-7, no single lexical item in these simple sentences completely determines whether the semantic descriptions are TRAJECTORY- or TRANSITION-SPACES. "The price soared" can be interpreted as a metaphorical trajectory, in which an imagined proxy for the abstraction *price* flies through an imagined stratosphere to symbolize its rising value. "The price soared" could just as easily represent a transition in which the monetary value grows over a period of time, because the word *soar* has an established sense meaning *rise* as well. Whether or not Lance should be required to produce all valid interpretations of a sentence is revisited in Sections 3 and 5, but it would be decidedly inappropriate for it to produce, for example, a trajectory in response to "the price decreased" or a transition in response to "the bird soared" for the particular senses of *bird, soar,* and *price* in the examples of Figure 2-7.

Figure 2-8 contains examples of TRAJECTORY SPACES with non-empty PATHS.



The dog ran away from the tree
S
NP
VP
The/DT
dog/NN
ran/VBD
PRT
PP
away/RP
from/IN
NP
the/DT
tree/NN
T ENTITY → . . . → CARNIVORE → CANINE → DOG
T ENTITY → . . . → WOODY − PLANT → TREE
D PLACE → AT
D PATHELEMENT → AWAYFROM
S PATH
R TRAVEL → TRAVEL − RAPIDLY → RUN → RAN
S TRAJECTORYLADDER
S TRAJECTORYSPACE
The dog ran away from the tree to the house via a rock
S
NP
VP
The/DT
dog/NN
ran/VBD
PRT
PP
PP
away/RP
from/IN
NP
to/TO
NP
the/DT
tree/NN
NP
PP
the/DT
house/NN
via/IN
NP
a/DT
rock/NN
T ENTITY → . . . → CARNIVORE → CANINE → DOG
T ENTITY → . . . → VASCULAR − PLANT → WOODY − PLANT → TREE
D PLACE → AT
D PATHELEMENT → AWAYFROM
T ENTITY → . . . → HOUSING → DWELLING → HOUSE
D PLACE → AT
D PATHELEMENT → TO
T ENTITY → . . . → WHOLE → NATURAL − OBJECT → ROCK
D PLACE → AT
D PATHELEMENT → VIA
S PATH
R TRAVEL → TRAVEL − RAPIDLY → RUN → RAN
S TRAJECTORYLADDER
S TRAJECTORYSPACE

Figure 2-8: Three Trajectories with non-empty paths

The Stanford Parser does not perform well when attaching prepositional phrases. Although the parse tree of “The dog ran away from the tree to the house via a rock” in Figure 2-8 is grammatically consistent, the literal interpretation of the last prepositional phrase subordinate to the verb phrase is “to the house that is via a rock.” This interpretation makes little or no sense. By eliminating the intermediate noun phrase and by moving the prepositional phrase “via a rock” to the level of the enclosing verb phrase, the interpretation of the parse is brought in line with the semantics shown in Figure 2-8. In general, it is not computationally feasible to try all attachment permutations when searching for a suitable interpretation. Even if such a brute-force approach were possible, it is unclear that it would be fruitful, because the problem of prepositional-phrase attachment may in itself be AI complete. To correctly attach the prepositional phrase in “The man saw a dog with a telescope” seems contingent on the commonsense knowledge that telescopes may be used to view objects, and that dogs do not know how to use telescopes. The implementation cannot address the problem of prepositional-phrase attachment in its entirety, but it can make headway towards a solution. In the example of Figure 2-8, the attachments chosen by the Stanford Parser do not lend themselves to *any* known semantic interpretation, let alone an interpretation that can be ruled out by common sense. It is reasonable to assume that a parse that lends itself to some interpretation constitutes a better choice than one which has no interpretation at all. The approach taken by the implementation, therefore, is to choose a prepositional-phrase attachment scheme that allows coverage of parse-tree elements by semantic elements. Further development of the process involved in testing various prepositional-phrase attachment schemes could lead to a more refined process in which other components of the Gauntlet System may iteratively refine the attachments.

2.3.3 CAUSE

Causal events described in terms of the meta-representation are not required to be of a uniform representational framework. It is possible, for example, to have a causal description in which the agent of the causal relation is a THING or TRAJECTORY SPACE and the outcome is a TRANSITION SPACE, as in “The distance from the boy to the tree decreased because the boy ran to the tree.”

Figure 2-9 illustrates a problem posed by causal sentences. The SBAR constituent, which indicates that the transition expressed in this sentence is caused by the trajectory, is subsumed by the verb phrase of the transition. This obscures the syntactic structure of the transition, complicating the matter of extracting a transition from this part of the sentence by pattern matching.

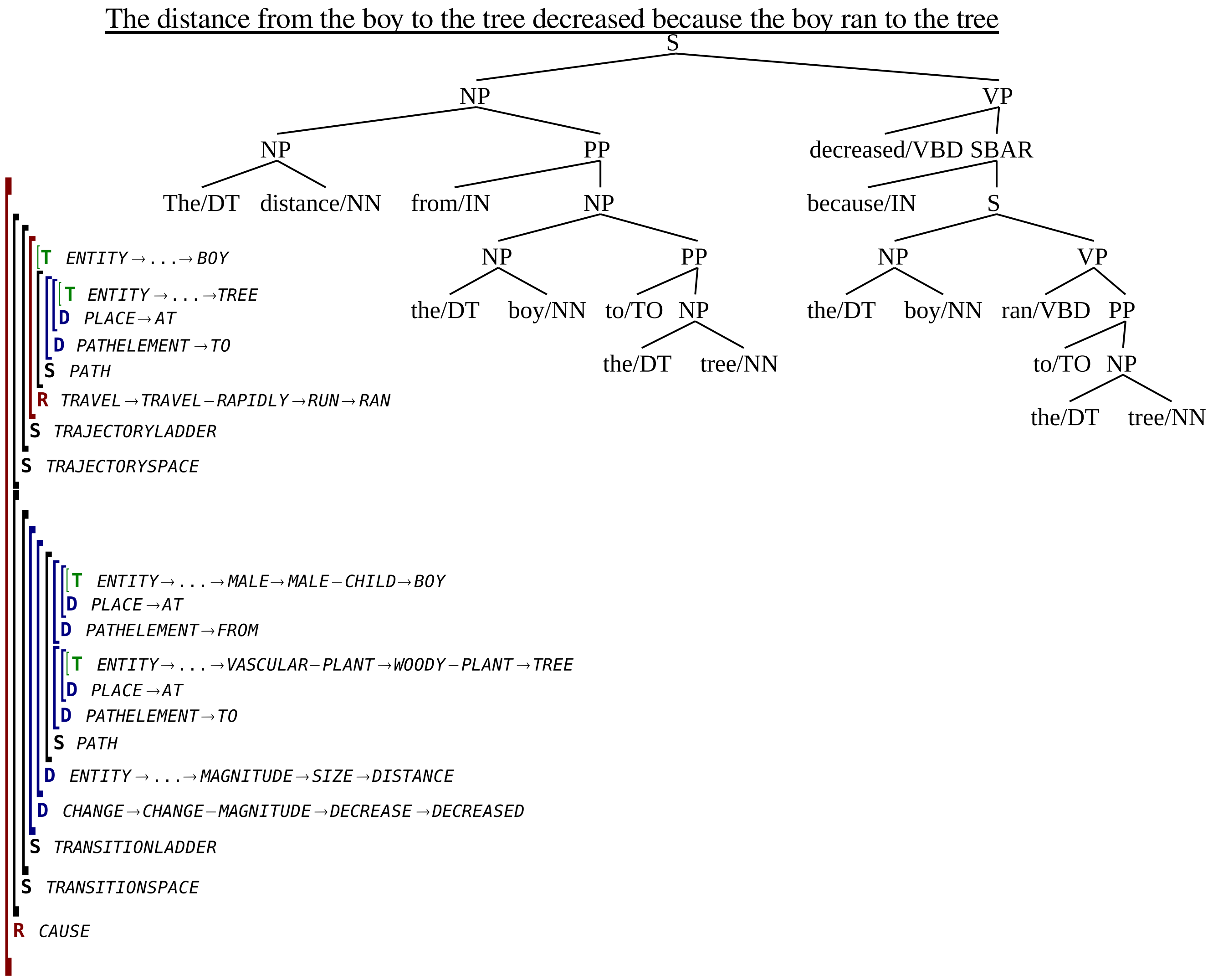
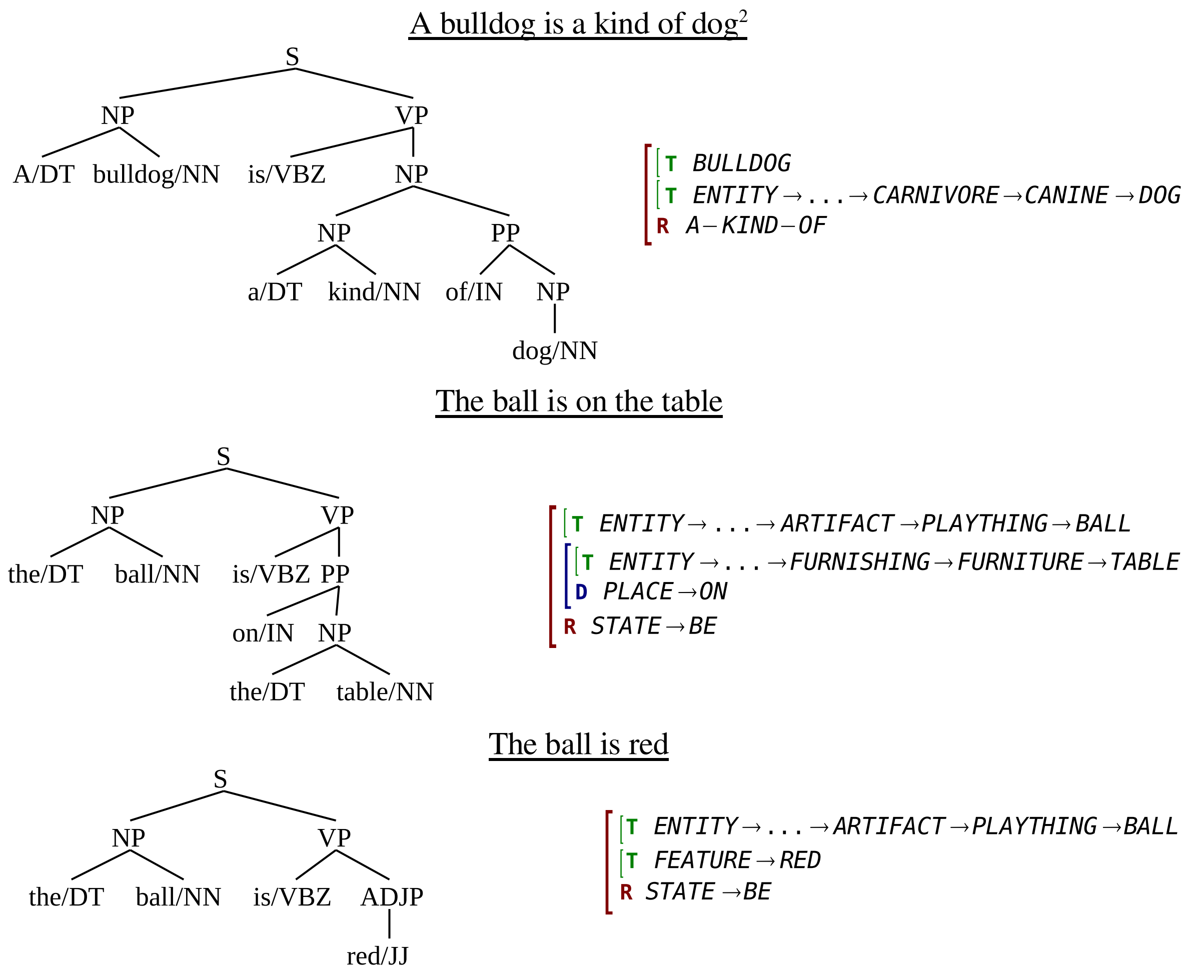


Figure 2-9: A Trajectory and a Transition participate in a Causal Relation.

2.3.4 States and Thread Memories

Declarative statements may convey information about location, condition, or other qualities that are represented by STATES in LCS. Alternatively, such statements can express categorical information of the type suited to Thread Memories. To recognize statements of the type "when I say *x* I mean *y*" as introducing a way to map new linguistic forms onto known semantic categories is an ability sought in the Gauntlet System[Winston 2007, 1], and the first step toward such flexibility is recognizing statements that expand existing categories of THINGS, e.g. "A bulldog is a kind of dog." A-kind-of and is-a statements differ from locative statements such as "the ball is on the table" or descriptive statements like "the ball is red." Locative and descriptive statements should evoke STATEs, rather than Thread Memories. Figure 2-10 identifies the semantics of simple declarative statements.



A bulldog is a kind of dog[2]
S
NP
VP
A/DT bulldog/NN is/VBZ NP
NP PP
a/DT kind/NN of/IN NP
dog/NN
T BULLDOG
T ENTITY→...→CARNIVORE→CANINE→DOG
R A−KIND−OF
The ball is on the table
S
NP VP
the/DT ball/NN is/VBZ PP
on/IN NP
the/DT table/NN
T ENTITY→...→ARTIFACT→PLAYTHING→BALL
T ENTITY→...→FURNISHING→FURNITURE→TABLE
D PLACE→ON
R STATE→BE
The ball is red
S
NP VP
the/DT ball/NN is/VBZ ADJP
red/JJ
T ENTITY→...→ARTIFACT→PLAYTHING→BALL
T FEATURE→RED
R STATE→BE

Figure 2-10: Semantics of Declarative Statements

Because Thread Memories comprise an important part of the meta-representation, I have introduced the A-KIND-OF and IS-A relations as an intermediate step to acquiring new threads. Mapping sentences such as “a bulldog is a dog” directly onto their Thread-Memory description would complicate Lance's implementation because it would force the process of associating syntax with semantics to scrutinize pieces of threads, rather than treating each thread in a template description as an immutable and self-contained pattern to match against input. Adding the A-KIND-OF/IS-A relation allows Lance to take full advantage of the descriptive ability of threads in the meta-representation without added complexity.

2 This example is structured as though *bulldog* were an unfamiliar word. In reality, WordNet contains a sense of this word already[Miller, G. 2006].

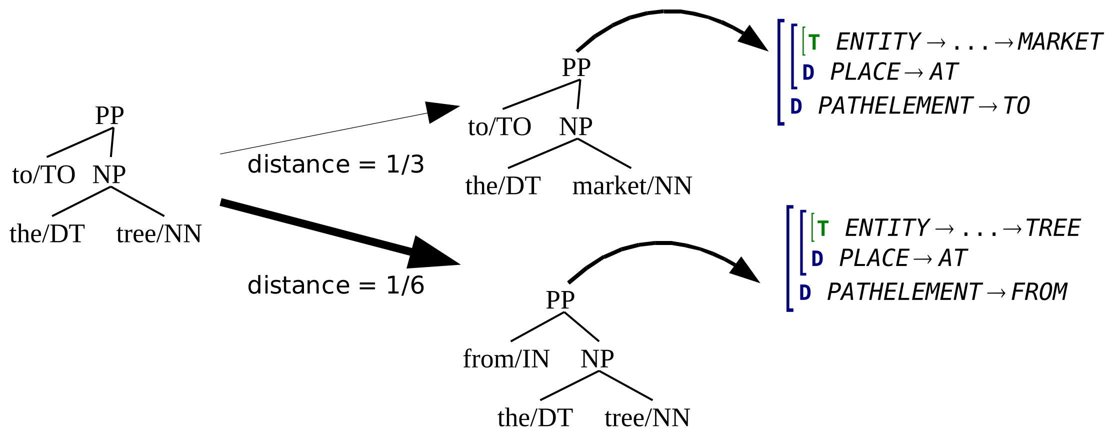
2.4 Extracting Regularity and Constraints from Examples

Learning from examples how to produce semantic descriptions of sentences and fragments requires building a model of the correspondences between syntax and semantics. A simplistic approach to developing such a model involves storing examples of fragment-description pairs used to train the system, and developing a scoring method that indicates the degree of difference between new sentence fragments and those previously encountered. The remembered fragment-description pair with the best score can then be selected as a template upon which to construct the semantics of a new fragment.

This way of learning by storing specific examples and then treating nearest neighbors, as decided by some distance metric, as templates, has the appeal of conceptual simplicity. Additionally, distance metrics are of particular interest in the Gauntlet System because they allow the expert components to store descriptions in Self-Organizing Maps (SOMs) [Kohonen 2001]. SOMs are outside the scope of this thesis but their prevalence in the Gauntlet System motivated use of distance metrics as a basis for evaluating similarity.

Despite its appeal as a conceptually simple way to extract descriptions that shares functionality with other components of the Gauntlet System, a nearest-neighbor approach based on distance metrics inadequately addresses some aspects of the language-representation problem. In particular, while such an approach is often proficient at identifying the best candidate from among a set of potentials, it cannot easily identify *all* candidates that meet criteria that are not easily expressed in terms of thresholds. Another problem with a model comprised of stored templates is that it becomes very difficult to analyze as the number of stored templates increases, because the constraints and regularities of the domain manifest though opaque distance scores instead of through explicit features of the model. Ad hoc adjustments to the distance metrics to adapt to nuances and exceptions in the language-representation domain become nearly impossible as the model grows, because the possible affects of such adjustments on each functional requirement of the system need to be considered separately.

Figure 2-11 caricatures the how process of matching a phrase against remembered examples can go wrong when there is sufficient asymmetry among the remembered examples that the input tree is closer, according to the metric, to the wrong remembered tree by virtue of irrelevant features. Of course, adjustments to the distance metric can always compensate for the peculiarities of a particular example, but such adjustments constitute a losing strategy in general.



	to	from	the tree	the market
to	0	1/3	1	1
from		0	1	1
the tree			0	2/3
the market				0

Distance Metric Rule: compute distance as an average of the fixed values in the table at left.

Figure 2-11: Caricature of the Weakness of Distance Metrics

Intuitively, the example of Figure 2-11 fails because the model does not capture the essential and seemingly obvious requirement that, in order for a prepositional phrase to represent a PATHELEMENT→TO, it must have the preposition *to*. Due to the disparity between the symbols that comprise the prepositional phrase and those that make up the semantic description, however, it would be an abuse of notation to equate the symbol *to* with the semantic symbol TO in order to facilitate comparison.

If the model is to capture the intuition about *to* and PATHELEMENT→TO, it must assimilate the information conferred in the difference between a prepositional phrase whose description is a PATHELEMENT-TO and a very similar counterexample, such as a PATHELEMENT→FROM in which the referenced PLACE is identical to that in the example. The essential feature of the counterexample is that it differ from the example in only one significant respect, thus constituting a *near miss.* The process of learning from near misses, first articulated by Patrick Winston as part of the Arch-Learning paradigm[Winston 1970], enables the following judgment, in terms of the comparison of a PATHELEMENT→TO example with a PATHELEMENT→FROM near miss: because the prepositional phrase of the example differs from that of the counterexample only in the preposition, it must be essential that a prepositional phrase contain *to* in order for it to represent a PATHELEMENT→TO. The new knowledge that *to* is required in the prepositional phrase of a PATHELEMENT→TO then augments the model of PATHELEMENT→TO.

Learning from near misses becomes more convenient when the near misses themselves are generated from examples of similar representations. If the representational space is sparse—that is, if only one semantic description typically suits a particular turn of phrase—then it becomes possible to use an example of one type of semantic representation as a near miss for models that would otherwise have matched the example. The procedure `update-all-models` generates near misses from examples when appropriate, in order to maximally improve all models by virtue of the new example. In the following pseudocode, a partially-described fragment is a parsed phrase that may or may not have a semantic description assigned to each of its subordinate clauses. In a training example, in which the top-level description of S must be provided by the trainer, any descriptions assigned to subordinate clauses of the partially-described fragment are guaranteed to be subcomponents of the top-level description.

```
Procedure update-all-models:
Inputs: {partially-described fragment S
         semantic description D}
     for each model M in memory:
          if type-of(M) equals type-of(D):
               generalize-model(M,D,S)
          else if qualifies-as-near-miss(M,S):
               specialize-model(M,S)
          else:
               do nothing
```

In the pseudocode, `specialize-model` is a procedure that applies near-miss learning heuristics to add more strict constraints to a model. The procedure `generalize-model` exploits regularity present in multiple descriptions to relax the constraints in a model. Models, in this context, specify the constraints that govern how sentence fragments map onto semantic descriptions. Such a language-representation model must have a single semantic representation at the root level of the parse tree. The type of the model is the same as the type of this root-level semantic model. Figure 2-12 depicts a schematic of a model of a TRANSITION SPACE, to illustrate the structural components of a nontrivial model.

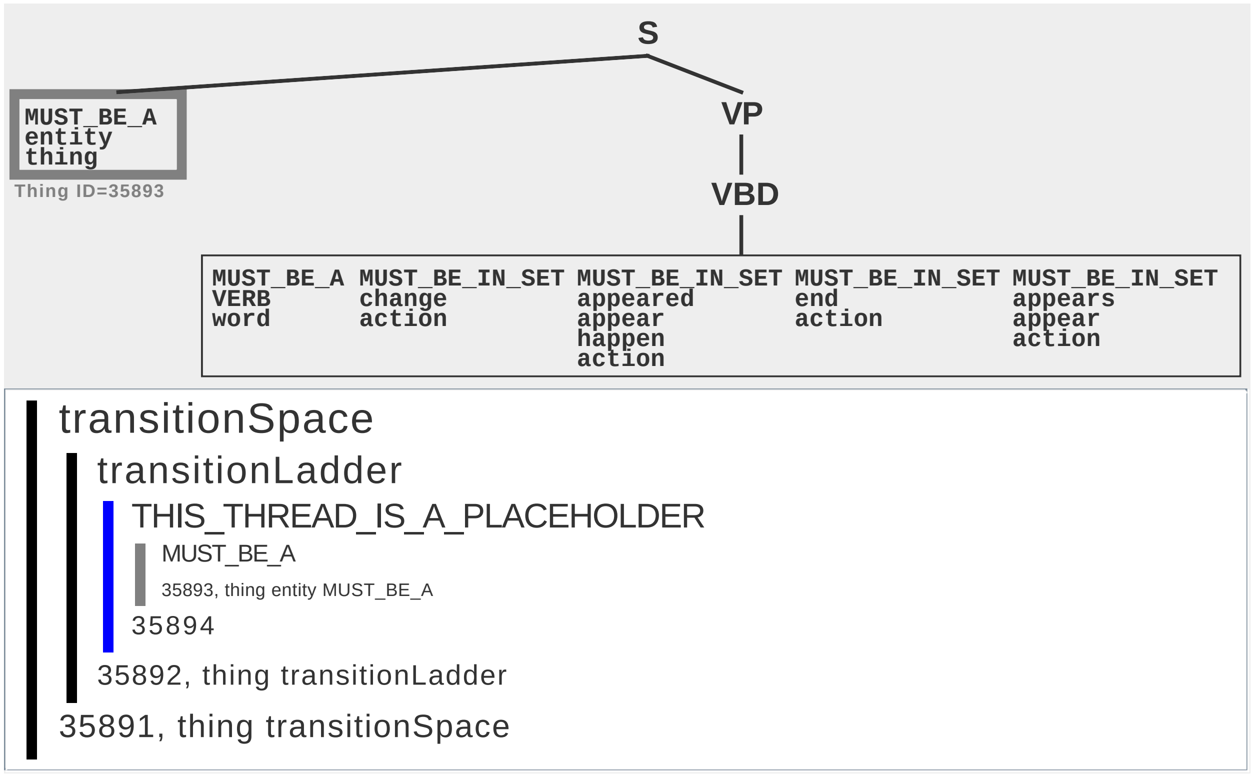

S
VP
VBD
MUST_BE_A entity thing
Thing ID=35893
MUST_BE_A VERB word
MUST_BE_IN_SET change action
MUST_BE_IN_SET appeared appear happen action
MUST_BE_IN_SET end action
MUST_BE_IN_SET appears appear action
transitionSpace
transitionLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
35893, thing entity MUST_BE_A
35894
35892, thing transitionLadder
35891, thing transitionSpace

Figure 2-12: A Transition-Space model

Of particular importance in Figure 2-12 are the explicit constraint labels added to both the parse tree and the semantic descriptions. MUST-BE-A labels specify that, in order for a subordinate semantic description or lexical item in a candidate tree to be consistent with the model, all threads in the model's bundle must be subsumed by threads in the corresponding part of the candidate tree. MUST-BE-IN-SET is similar to MUST-BE-A, but it identifies a set of possibilities rather than a single partial thread. By composing MUST-BE-IN-SET with MUST-BE-SUBCATEGORY, the model of Figure 2-12 captures the constraint that the verb in a sentence must be either a CHANGE, APPEAR, or END verb in order for the sentence to represent a TRANSITION SPACE. Section 4 contains the complete ontology of model constraints.

2.5 Refining the Language-Representation Model

The model introduced in Section 2.4 concisely captures meaningful constraints that govern how language maps onto representation. This model has a subtle problem, however, that arises in part from the nature of learning by near misses: the model does not easily capture constraints that hinge upon several simultaneous differences between examples and counterexamples. To motivate a discussion of how to address this problem, Figures 2-13 – 2-16 illustrate an excerpt of a training sequence that causes the learning mechanism to build a language-representation model of TRANSITION SPACES.

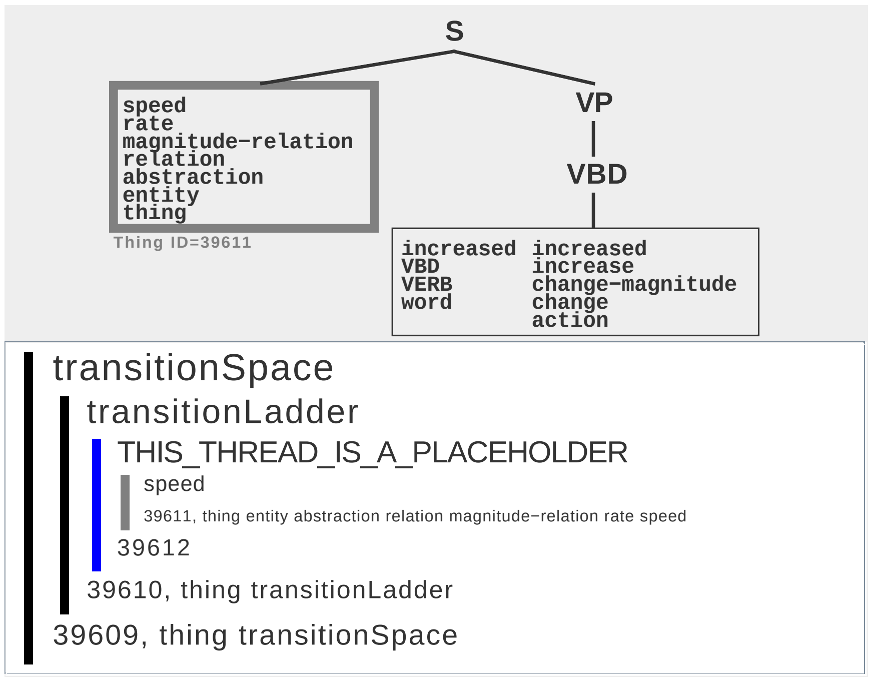


Figure 2-13: The Description Becomes the Initial Model

In figure 2-13, the learning mechanism has just received its first example of a TRANSITION SPACE. The parse tree of "The speed increased" has already had its noun phrase replaced with a THING, via a bottom-up match and replace procedure. Also, note that below the verb-phrase node there are two nodes instead of one compound word/tag node as in previous figures. The compound notation was a simplified notation; actual parse trees have an extra node that specifies the type of word in each leaf (the preterminal node). The thread contained by the preterminal node in Figure 2-13 means that the leaf is a past-tense verb, which is a verb, which is a word.

The first action of the learning mechanism is to link all semantic components in the description provided along with the parse tree to either partial descriptions found at subordinate parse-tree nodes, such as the THING description that has been attached to the noun phrase, or threads from lexical nodes, such as the thread CHANGE→CHANGE-MAGNITUDE→INCREASE, to their corresponding elements in the supplied description for the sentence as a whole. The PLACEHOLDER thread stores the association between the verb thread and the corresponding `derivative` in the semantic description.

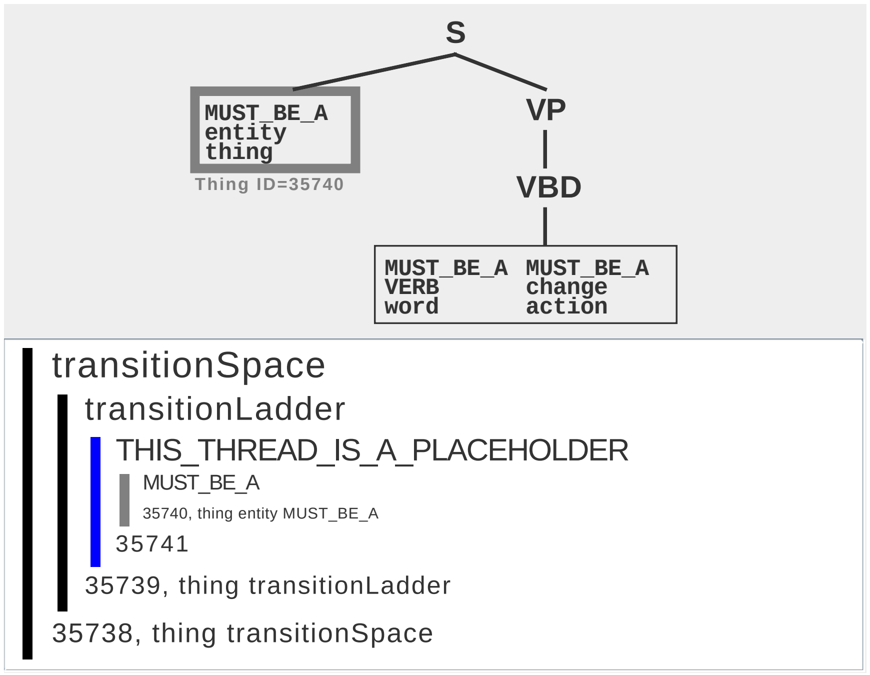


Figure 2-14: Generalize Model via "The bird mutates"

By giving the learning mechanism just one more example, the model is made much more general. Although near misses must differ in only one important respect in order to change the model, examples may have several differences, because all are assumed to act independently to generalize a portion of the model. In the example of Figure 2-14, the parse tree of "The bird mutates," along with the associated semantic description, causes three changes in the model. First, the type of word permitted in the verb phrase is generalized from a past-tense verb to any kind of verb. Second, the THING described by the noun phrase may now be any type of ENTITY, because the thread of *bird* diverges from that of *speed* in the first symbol after ENTITY. Finally, the verb is generalized to any type of CHANGE verb, for reasons analogous to the those governing the other two generalizations. The mechanism of the constraints on the THING and the transition verb is to add a MUST-BE-A symbol to the end of the thread of the relevant THING type. The mechanism of the constraint on the verb class is to add a MUST-BE-A constraint to the thread of parse-tree node itself.

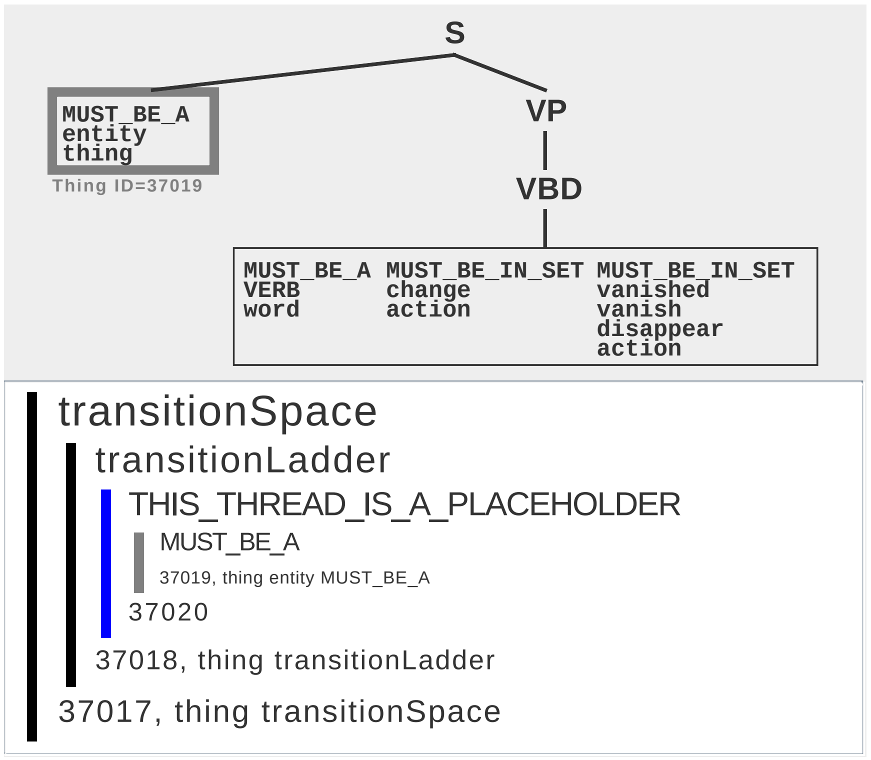
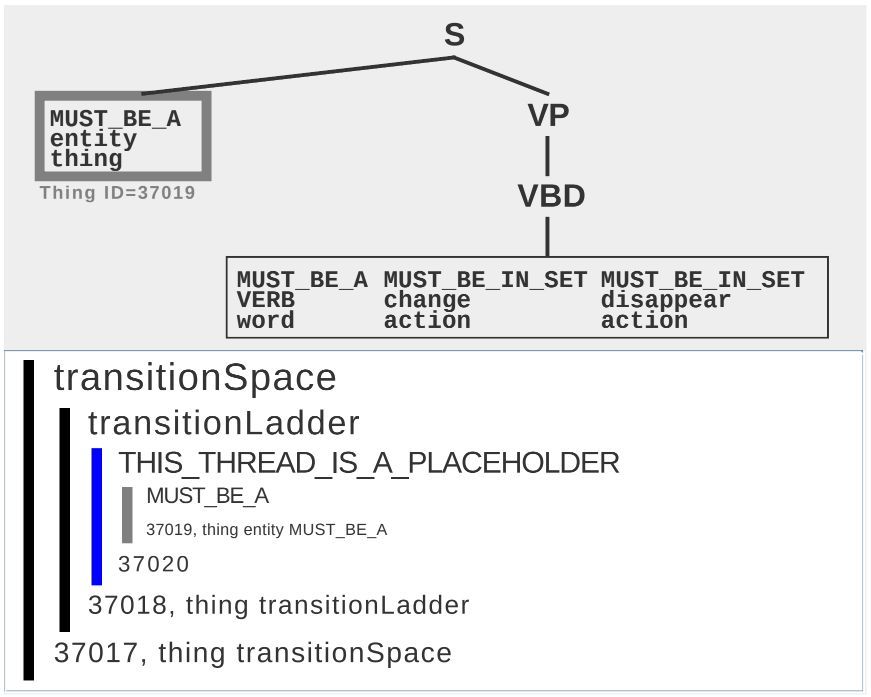

S
MUST_BE_A
entity
thing
Thing ID=37019
VP
VBD
MUST_BE_A VERB word
MUST_BE_IN_SET change action
MUST_BE_IN_SET vanished vanish disappear action
transitionSpace
transitionLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
37019, thing entity MUST_BE_A
37020
37018, thing transitionLadder
37017, thing transitionSpace

Figure 2-15: Generalizing by Expanding a Set

When the learner receives the partially-described parse tree of "A dog vanished" along with the associated TRANSITION-SPACE description, it cannot reconcile this example with the model because of the MUST-BE-A constraint added in the last step. The only option available that generalizes the model is to expand the set of possibilities. The constraint added in the step depicted in Figure 2-15 loosens the requirement on the verb: the verb may now be any CHANGE verb or exactly the verb *vanished*.


S
VP
VBD
MUST_BE_A entity thing
Thing ID=37019
MUST_BE_A VERB word
MUST_BE_IN_SET change action
MUST_BE_IN_SET disappear action
transitionSpace
transitionLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
37019, thing entity MUST_BE_A
37020
37018, thing transitionLadder
37017, thing transitionSpace
Figure 2-16: Generalizing *Died* and *Vanished* to *Disappear*

In the final step of Figure 2-16, the TRANSITION sentence is “a bird died.” The thread has the symbol DISAPPEAR in common with the verb vanish, so the MUST-BE-SUBCATEGORY constraint is applied to the model.

Consider what happens when the model developed in Figures 2-13 – 2-16 is tasked with assigning a semantic description to “A bird soared.” As noted in Section 2.3.2, a TRANSITION-SPACE interpretation of this sentence is dubious at best, and probably should not be admitted. WordNet, however, contains many interpretations of *soar*, as listed in thread form in Figure 2-17. Among the interpretations is a CHANGE action.

TRAVEL → RISE → SOAR
TOUCH → HANDLE → MANIPULATE → OPERATE → FLY → HANG−GLIDE → SOAR
TRAVEL → FLY → SOAR
CHANGE → CHANGE −MAGNITUDE → INCREASE → GROW → RISE → SOAR
TOUCH → HANDLE → MANIPULATE → OPERATE → GLIDE → SAILPLANE → SOAR

Figure 2-17: Thread Representations of the Senses of *Soar* in WordNet[Miller, G. 2006]

Because the training step shown in Figure 2-14 generalized the model such that it will admit any THING permitted by the senses of *bird*, nothing prevents the model from admitting the TRANSITION-SPACE interpretation of "A bird soared" that means literally that a feathered flying animal increased. We condone such absurdity because, presumably, the TRAJECTORY-SPACE model would admit the conventional interpretation of "A bird soared," and so it might seem reasonable to delegate the task of deciding which interpretation is best to Gauntlet's experts. This is not reasonable, however, because retaining all of the descriptions admitted by each model as Lance processes the parse tree from the bottom up would lead to combinatorial explosion. The unfortunate fact is that some interpretations of subordinate clauses must become pruned before the top-level models are even applied.

How should Lance evaluate which intermediate interpretations to prune? One path to answering this question is to modify the language-representation modeling framework so that the TRANSITION-SPACE model of Figure 2-16 can capture the constraint that, roughly stated, if the THING is a physical object then the transition verb has to be of a class that includes words such as *change, mutate,* and *vanish,* but not words such as *increase* and *decrease* that apply to abstractions like distance, volume, or value but not to physical objects. Not only would such a constraint be cumbersome to teach because of its subtlety, it would require models and learning methods that are sensitive to high-order interactions between different aspects of descriptions. Winston appealed to the scientific method to justify a learning method that largely hinges on the descriptive ability of first order constraints: "In science as a whole, each particular method for treating interacting effects is usually a major problem in itself and over-ambitions search for completely general methods is of low utility when premature."[Winston 1970 – p. 150] General methods for extracting high-order constraints would not only be difficult to devise in the learning method but would likely be computationally infeasible as well.

Presently, I do not have a way to handle constraints that the system cannot learn because they depend on too many variables simultaneously. I have employed a heuristic strategy for the time being and I suggest alternatives to this compromise in Section 5.

To compensate for constraints that Lance is yet unable to learn, I have brought the very distance metrics that performed so poorly at classifying the representational space back into service; this time to indicate a measure of familiarity. To see how this works, suppose that the counterexample given in Figure 2-18, which is actually an example of a TRAJECTORY SPACE, were stored in a list of example descriptions associated with the TRAJECTORY-SPACE model. Now, when Lance is presented with "A bird soared," both the TRAJECTORY- and TRANSITION-SPACE models admit several matches, each containing threads associated with the various senses of *bird* and *soar*.

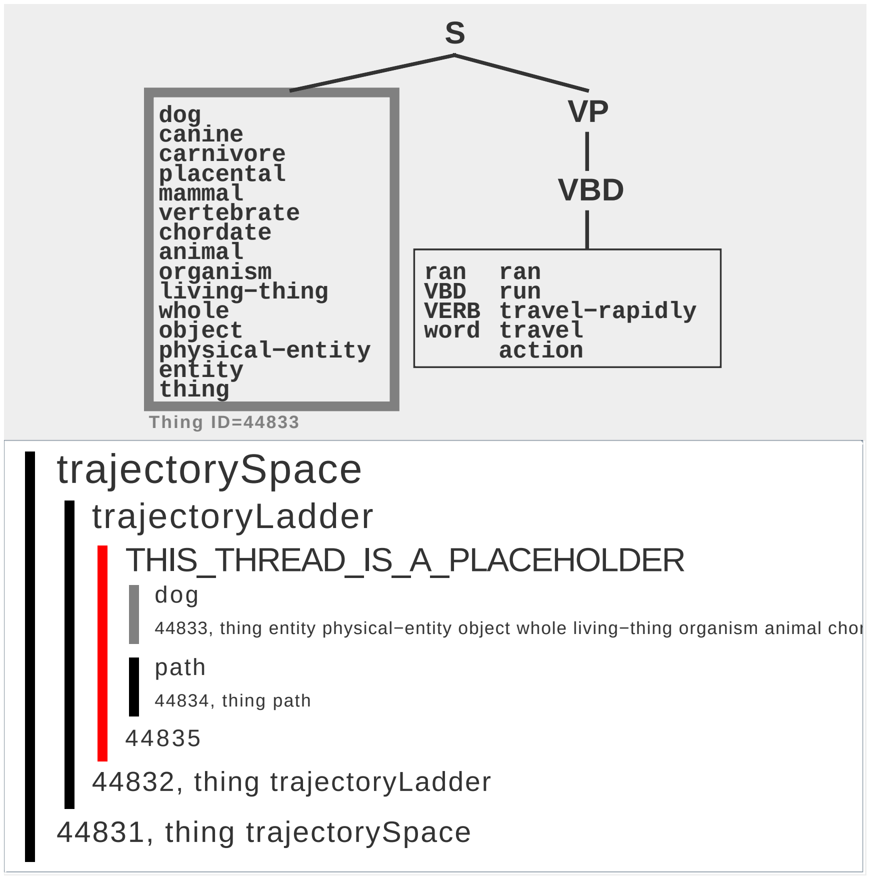


Figure 2-18: A Trajectory Example (Implicit Transition Counterexample)

Because the the distance between the TRAJECTORY-SPACE interpretation of “A bird soared” that describes an animal in an aerial trajectory is so close—speaking in terms of the distance metric—to the TRAJECTORY-SPACE interpretation of “A bird flew,” this interpretation would be ranked highest among all potential matches. The TRANSITION-SPACE interpretation that means literally “a feathered animal increased,” on the other hand, would be quite dissimilar to previously encountered TRANSITION SPACE examples by virtue of the disparity between ENTITY→PHYSICAL-OBJECT→...→BIRD and the subjects of other TRANSITION SPACES that entail increase/decrease verbs, which would presumably be ABSTRACTIONs rather than PHYSICAL-OBJECTs. As a result, the similarity score for the absurd TRANSITION-SPACE interpretation of “A bird soared” would be low. The process of finding the best description or descriptions at each stage of recognition can take on a beam-search quality: the first-order constraints of the language-representation models admit many possibilities, a few of which are chosen based on their preferred heuristic similarity scores.

The final topic not yet addressed is context. More often than not, interpretation of a sentence or phrase is governed not only by memories of past experiences but by the situation at hand. If the Gauntlet System is ever to form complex, multimodal concepts, it will surely need a way to combine cues from vision and perhaps from other recent linguistic events to influence the interpretation of sentences and fragments. The nature of how context influences understanding is fertile ground for research, and a complete solution is far beyond the scope of this thesis.

Failure to address context, nevertheless, would severely limit the utility of Lance as research into Gauntlet's concept-formation domain continues. In order not to cripple such endeavors as finding out how the visual apparatus contributes to language understanding, or discovering how to represent information conveyed across several sentences, Lance must provide a feedback channel that is as flexible as possible given the ill-defined parameters of the problem.

A simple, expedient approach to address context is to allow external components, such as Gauntlet's yet unrealized Cross-Representational Memory (refer to Figure 1-4), to place descriptions in a temporary buffer. When this buffer is not empty, Lance will preferentially assign the descriptions that it contains rather than those seen previously as training examples. By using this apparatus, an expert that anticipates that anticipates that an impinging phrase in the language stream will have to do with a certain PATHELEMENT, for example, may force Lance to choose that PATHELEMENT even if other interpretations of a phrase are possible.

3. Implementation

In this section I describe the implementation of Lance, a program that learns how to extract semantic descriptions from language by analyzing example pairs of phrases and descriptions. The three principles of Flexibility/Coverage, Correctness, ,and Transparency, distilled from the examples and analysis of Section 2, guide this implementation:

- **Flexibility/Coverage**: The representational frameworks introduced in Section 2 exemplify the type upon which the Gauntlet System will depend, but they are not comprehensive. Other representational frameworks will take hold as the system progresses. This means that Lance's learning mechanism must be flexible enough to acquire the mappings between language and these new representations in order to provide the best possible coverage of the representational space. In Section 2 I point out some situations in which more than one representation makes sense for a given example. Lance must be able to produce a description cast in at least one of the admissible representations so long as the model for extracting that description abides by the stipulation for the guarantee of correctness. Of secondary importance, it would be desirable to produce as many varieties as possible, as long as doing so would not compromise coverage of other representational domains, or the correctness or transparency of the system.
- **Correctness**: As long as a language-representation mapping can be understood in terms of first-order constraints, then there must be at least one finite training sequence such that Lance will be able to produce descriptions according to the mapping after processing that sequence. The first-order requirement arises per the discussion of Section 2.4. If a mapping depends on higher-order constraints then a best effort should be made to approximate it via heuristics but no strong guarantee can be made as to correctness.
- **Transparency**: The models generated by Lance should be understandable by a human interpreter. This is because the purpose of acquiring models by extrapolating from examples is to automate a design process that is mostly tedious and error-prone, but that

occasionally reveals subtle and interesting constraints that may lead to speculation about how language inspires representation or vice versa. While I harbor no pretense that my implementation can discover such subtlety more adeptly than its human trainer, I do not dismiss the value of a transparent internal representation as a tool to teach oneself, through the process of teaching Lance, about language and representation. Philosophical issues aside, transparent representations are extremely helpful in debugging training sequences because, without them, the only recourse when things go wrong is to trace manually through the program's execution—a process which would be significantly more painful and tedious than designing pattern-matching routines by hand.

Lance's design is guided by these three principles. In this section I present the following features of Lance's implementation:

- A preprocessing stage that draws lexical information from WordNet and modify the results of the Stanford Parser.
- A learning procedure that is based on Winston's Arch-Learning paradigm[Winston 1970]
- A recognition procedure that uses first-order models augmented by distance scores
- A priming mechanism that affords external components control over Lance's recognition process
- A graphical interactive training environment.

3.1 Overview

Lance is a Java program comprising 68 public classes and interfaces. I present in this section only the features of Lance that are especially interesting or essential to learning to model the correspondences between sentences and fragments.

Lance's external interface and many of the interfaces shared by its internal components conform to the Wire design pattern[Winston 2003, 2]. This pattern comprises abstractions above the interface mechanics of the Java programming language in which Lance and Gauntlet are written. Instead of the usual pattern whereby classes of components expose methods to other components, components that follow the Wire pattern expose ports, which can be likened to the terminals of an electrical component. The wire pattern simplifies the process of integrating components—the ports of the components to be integrated can simply be connected by wires, without regard to any details of each component's function.

Figure 3-1 is a block diagram of Lance depicting the external interfaces and the interfaces between components. There is a single port for training the system and for recognizing new inputs; the behavior depends on the type of input received. A port controls the priming mechanisms that allows external components to influence Lance's preference for certain representations. A single port delivers recognized descriptions from Lance to Gauntlet's expert components, although a demultiplexor may be connected to this port to send descriptions to different destinations depending on the type of the description. Finally, a privileged interface connects directly to Lance's learning module, allowing the interactive training component to access Lance's learning mechanisms.

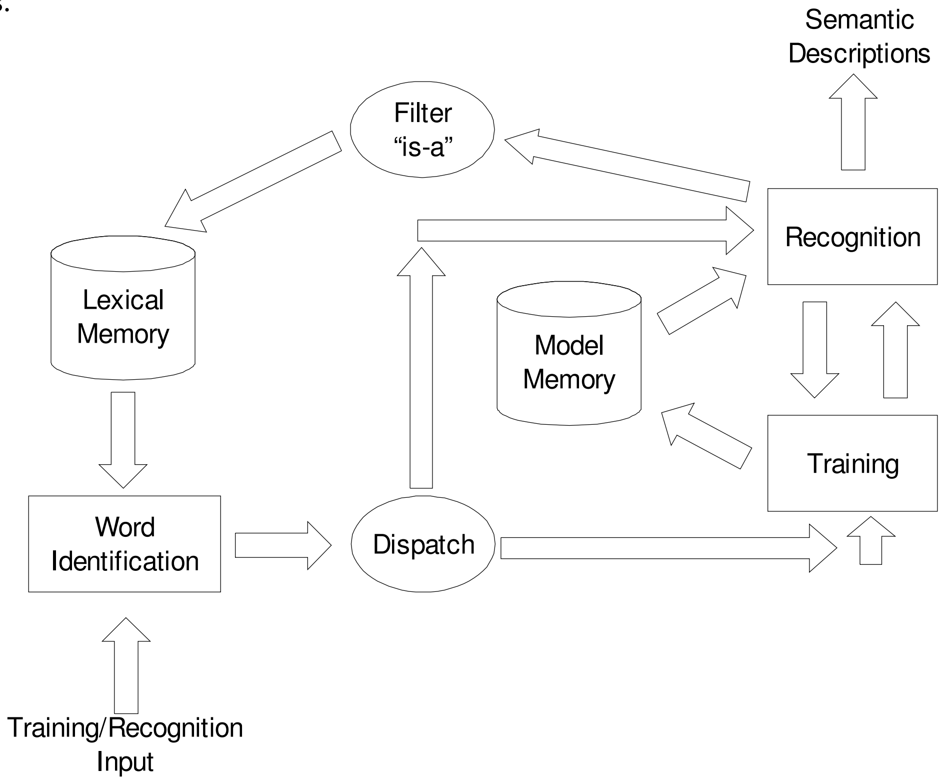


Figure 3-1: Block Diagram of Lance

3.2 Preprocessing Stage

An input on the main input port may indicate that one of three types of events has occurred: receipt of a training pair, receipt of a sentence to recognize, or receipt of a description to render as a sentence or fragment. These three event types are uniquely distinguishable by the types of information supplied. To provide the greatest flexibility, the inputs are accepted in a variety of forms.

A character string received on the main input port is assumed to be a sentence or fragment, and so it must be parsed. An instance of the Stanford Parser, which is discussed in Section 4, performs this parsing. The Stanford Parser used in Lance has a modified interface so that it can accept pre-tagged words, which may be specified by adding /TAG after the word without any whitespace, where TAG is one of the Penn Treebank tags[Marcinkiewicz et al. 1993]. Pre-tagging one or two words in a phrase is often all that is required to force the Stanford parser to treat the entire phrase differently than it would if it were allowed to derive the tags on its own. This is especially useful when training Lance on sentence fragments, because a Stanford Parser trained on the Wall Street Journal corpus[Marcinkiewicz et al. 1993] tends to be better at parsing sentences than fragments. Once parsed, the sentence is treated in the same way that incoming Stanford parse trees are treated.

If the input is a Stanford parse tree, it is converted to a different form that I call a flexible tree. Flexible trees differ from the Stanford trees in four important ways: the nodes can have constraint labels such as MUST-BE-A, bundles of Thread Memories can augment any node label, any node can have a set of descriptions assigned to it, and an interface for rearranging a tree (e.g. moving prepositional-phrase attachments) is provided.

Any input that is an instance of the `Thing` type or a subcategory of this meta-representation type is treated as a description with no further processing.

If the input is a Java HashMap, it must contain exactly one of the strings `stanford-tree`, `flexible-tree`, or `sentence-fragment` as a key, along with the appropriate value, and at least one `Thing` instance in the slot of some other key. HashMap inputs may have any number of `Thing` instances; in fact, they must have as many as apply to the included fragment, or else the assumption of a sparse representational space may lead Lance to generate erroneous counterexamples. HashMap inputs may also carry along arbitrary baggage, such as images, audio, etc. These objects are stored in a vector that accompanies each training example, so that when a model admits a match, any objects that accompanied the closest example to that match are packaged with that match before it is emitted to the experts.

The penultimate preprocessing activity that Lance performs before sending the repackaged input along to the recognition or learning stage is looking up the bundles of threads corresponding to each word and each preterminal node in the flexible tree. The threads of preterminal nodes are hard-wired, because they can be of a very limited range of possibilities. First, a lexical memory component is searched. If the lexical memory does not contain an entry for the word in question Then the search proceeds to WordNet[Fellbaum 1998], accessed via the Java WordNet Interface [Finlayson 2007]. The two-step search allows the modifications to the senses in WordNet if they are unsatisfactory. There may be several approaches to constructing threads from WordNet; my approach is essentially to walk along hypernym chains and collect the representative symbol from each synset visited, while maintaining a visited-synset list to avoid getting stuck in loops, and adhering to some additional precautionary measures to curtail excessive branching.

A final stage of preprocessing generates all prepositional-phrase attachment permutations according to a set of attachment rules, when operating in recognition mode. This is a naïve approach to dealing with prepositional phrases, because in the general case a simple generate-and-test is computationally infeasible. If the number of prepositional phrases per sentence is assumed to be small, the added computational burden is small. The prepositional-phrase attachment permutations are stored in a queue and tried by the matcher until one of the attachments produces a description of the top-level node of the tree. The permutations are generated according to a small group of transformations that govern to which noun phrases and verb phrases a given prepositional phrase may attach without affecting the word order in the sentence or fragment that was originally parsed. Figure 3-2 is a view from a prepositional-phrase reattachment tool that is part of Lance's graphical training-set editor. A prepositional phrase is highlighted in gray, and its possible points of attachment according to the rules are highlighted in red.

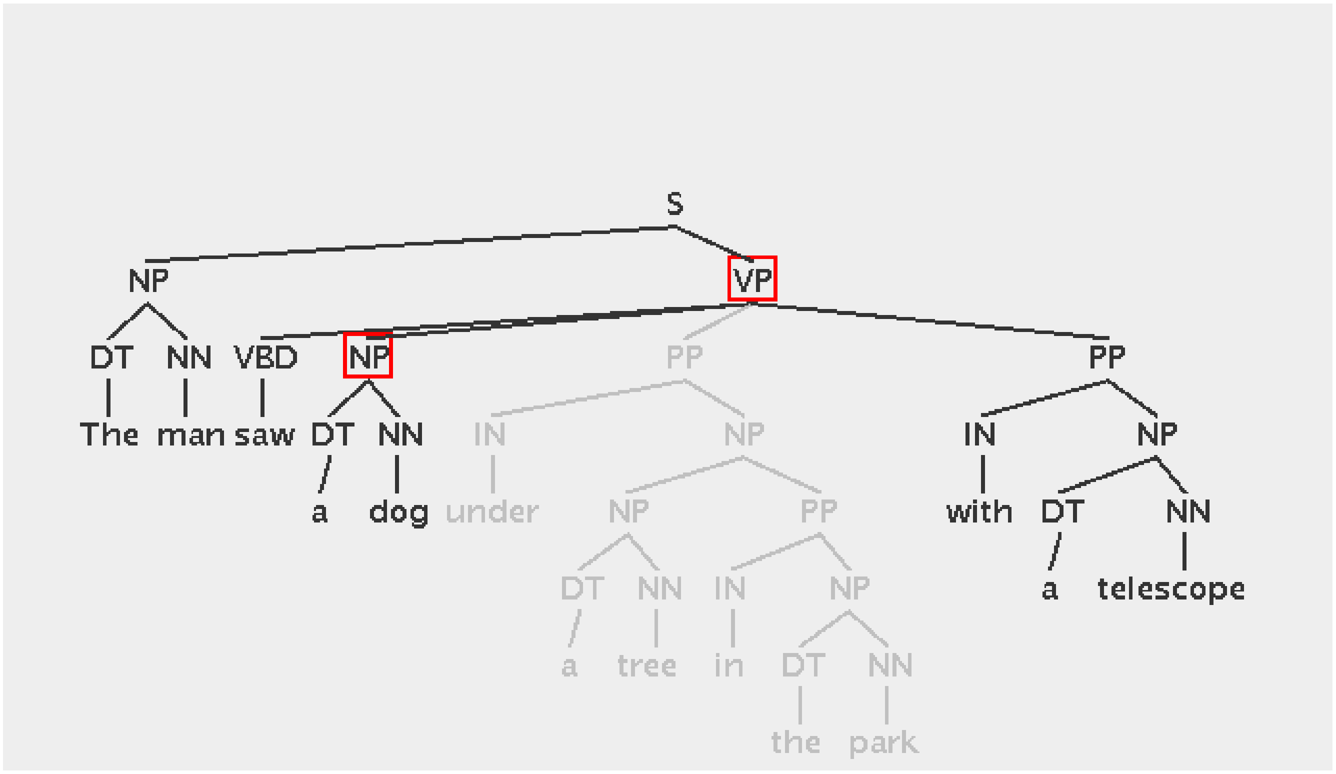


Figure 3-2: Lance's Graphical Prepositional-Phrase Attachment Editor

3.3 Model Building

If a language input is accompanied by a semantic description, it constitutes a training example. Section 2.4 illustrates acquisition of a TRANSITION-SPACE model through a training sequence. This section presents a detailed overview of the procedures and representations that interact to facilitate this modeling.

In order to successfully acquire the constraints embodied in a training pair, Lance must already have learned the constraints governing the constituents of the sentence or fragment of the new training pair. To learn about trajectories, for example, Lance must have acquired the constraints that govern descriptions of objects and path elements, so that learning about trajectories is reduced to learning a few simple constraints, which in the case of trajectories are the kinds of objects can participate, the limitations on the motion verb, and the optional inclusion of several PATHELEMENTS. Lance's learning procedure and recognition procedure are intertwined: recognition closely follows the structure of the constituency tree, and assigns descriptions to subordinate clauses before learning can acquire the description of the highest-order node of the parse tree. The recognition process is self-referential as well, because descriptions cannot be assigned to clauses until subordinate clauses have descriptions that match the patterns in the recognition rules for these subordinate clauses.

The interdependence of learning and matching grounds out when both the sentence fragment and the representation provided to Lance have no internal structure. Many noun phrases lack phrasal structure below the root node, and so noun phrases may serve as such a base case.

The first action performed upon receiving a new training pair after subordinate matching has completed is to resolve the references between the threads attached to the leaves of the parse tree, which represent words, and the threads present in the semantic description. If the system were given the phrase “a dog,” as a first training example, along with a THING description including the thread:

entity physical-entity object whole living-thing organism ... canine dog[Miller, G. 2006],

then it must determine that the thread in the word's bundle refers to the thread in the semantic bundle. If the description has no internal structure beyond its group of threads then this reference resolution is straightforward. The references are preserved explicitly in the just-instantiated model of the correspondences between noun phrases and semantic descriptions, though, because the learning process will eventually modify the threads, making equality comparisons inadequate to accurately determine reference. Once the reference between the *dog* thread in the word's bundle and that in the semantic description has been established, the system attempts to discard any lexical information that is irrelevant to the description so that it does not interfere with learning. The word *dog* has at least seven distinct senses in WordNet[Miller, G. 2006], and we would not want the system to make inferences about the similarity and difference between the unintended senses of *dog* and those of other nouns, and use such inferences to constrain the model.

After the *dog* example is processed, the system has one model, depicted in Figure 3-3, that consists merely of a parse-tree fragment, a description, and a set of references between threads. This model is a seed, and requires generalization before it may be used in matching. Now suppose that the system encountered the phrase “a man,” paired with the appropriate semantic description. This example is determined to be of the same type embodied by the model, because its semantic description's only thread is similar enough to the the model's only thread, and because the two parse trees can be completely aligned. Generalization takes place, and the model, shown in Figure 3-4 (a), is now capable of matching any phrase that has the word *a* followed by a noun that has a sense as a type of organism. In Figure 3-4(c), the training example “the rock” causes further generalization, and subsequently the model would match any determiner followed by a noun that has a sense as a whole object.

When the system receives “the price,” the thread is sufficiently different from the extant model of physical things that a new model is generated. I have, somewhat arbitrarily, imposed the constraint for `thing`s that the first three symbols of the item's prime thread must match in order to qualify a model match. The higher the number of symbols that must match, the more models will be generated to accommodate the taxonomy of THINGS. Having slightly more models than necessary may affect the ability of the system to perform efficiently but it does not seriously limit its capabilities. Instantiating far too many models by way of an overly strict similarity threshold would make the training tedious—if what constraints have been learned about *dog* noun phrases must be re-learned for *cat* noun phrases, there would be little benefit in using this system. Allowing too few models would make learning impossible. In the extreme case where only one model were permitted by a similarity threshold that is too relaxed, the model would simply degenerate to the point where it admitted every turn of phrase under one uniform, but meaningless, semantic category.

Next, consider the PLACE description of the prepositional phrase “on the table.” Prepositional phrases subsume noun phrases, for which the system must have already acquired models if the training is to be successful. The first stage of learning “on the table” is to recognize the subordinate noun phrase “the table.”

As was the case with dog, WordNet has many senses of *table*. Clearly, though, if the trainer has provided a semantic description including a THING description of table that indicates a piece of furniture, that is the only description that must be considered in matching. The approach taken by Lance when learning from new training examples is to invoke its own priming mechanism, which is a filter that causes Lance to prefer descriptions found within a set of provided descriptions, during any stage of recognition. The set of provided descriptions is derived from the training example by decomposing its description into all of that description's component parts.

Recognition of “the table” is a three-step process: first, all models are presented with the parse-tree fragment, and a subset of models will respond with a positive match. Then, each of these matching models produces a new description by aligning the new parse tree with the model's parse tree, and replacing all elements of a copy of the model's semantic description with the corresponding elements of the new parse tree, as determined by the alignment of the parse trees and the references that were stored when the model was instantiated. Finally, the outputs of all matching models are filtered, in this case by the priming mechanism, and only those descriptions that pass through the filter are retained. In the case of “the table,” only the THING included in the PLACE that was given as a training example is retained.

After all subordinate clauses have been assigned, the training of the new semantic category PLACE→ON may begin. Because this new description and its accompanying fragment match no previously learned models, a new model is instantiated. The thread of the preposition, *on*, matches no threads in the semantic description so no association is formed. The subordinate THING description, however, which is present in the parse-tree fragment because it was attached during the recognition step, is equivalent to the subordinate THING in the PLACE description, so a reference is formed between the two.

Subsequent training with the the PLACE “on the bird” will cause the model to become generalized. There is an implicit requirement that the teacher provide information that is relevant and informative. “On the bird” is a particularly good choice for teaching PLACE→ON after the first example, “on the table,” because it conveys the information that the THING represented by the subordinate clause doesn't have to be inanimate. The learner applies a method analogous to the climb-tree heuristic of Winston's Arch-Learning methods[Winston 1970] by generalizing the threads for *table* and *bird* to a new model thread that embodies the constraint that, in order for a prepositional phrase to be a PLACE→ON, its subordinate THING must be a whole object. Both training examples contain the word *on*, and this is assumed to be relevant: a MUST-BE constraint is placed in the word bundle to assure that only prepositions containing the word *on* are admitted as PLACE→ONs. It should be noted that these constraints are not final—in the hypothetical scenario in which the trainer wanted the system to learn that "atop the table" meant the same thing as "on the table," presentation of a new training example of "atop the table" paired with a PLACE→ON would cause the system to apply the climb-tree heuristic to the word thread, generalizing it from the specific word *on* to any preposition. Counterexamples could then do the job of eliminating undesired prepositions.

The process of providing counterexamples that prevent sentences and fragments from being recognized as members of the wrong semantic categories would be prohibitively tedious were it not the case that training examples of one category automatically serve as counterexamples of categories that would otherwise have matched the parse fragment of the training example on the basis of its structure and partial description. This ability to cross reference examples with counterexamples is unique to sparse representational spaces. The goal of the representations of the Gauntlet System is coverage, so overlap among representational spaces is not forbidden. Based on qualitative observations of TRANSITION- and TRAJECTORY-SPACES, PLACES, PATHELEMENTS, and linguistic forms that represent thread memories, though, it seems plausible that the assumption of a sparse representational space will serve well in most instances.

Discovering which models treat an example of a different category as an counterexample is easy, but uninformative. The counterexample must also be a near miss in the Arch-Learning sense; one which is similar enough to an example that it conveys information about a new constraint. How similar is similar enough? A noun phrase would under no circumstances be considered a potential match for a PLACE, because of the structural mismatch between the parse trees of the items in question. This suggests that the first rule for evaluating whether an example is a near-miss should be that the parse trees must be structurally consistent. The precise definition of structural consistency in this context is that, evaluating from the root of each parse tree onward, every corresponding pair of nodes either has a semantic description attached to both nodes or has the same number of children, except in the case that a child's semantic description references an element of a `sequence` in the meta-representation notation of the top-level semantic description of the model, in which case any number of such children are permitted. For the purposes of clear discussion, I will consider only cases in which the branching factors of the example and the model are the same[3].

Structural consistency alone is not enough to merit consideration as a near miss. If the system had acquired a new model from the PLACE description of “on the table,” and was then given “to the floor” as a PATHELEMENT, and hence a potential near-miss for the PLACE model, it would be unwise to treat it as a near miss. The reason is that, because it differs from the PLACE model in both the type of THING it subsumes and the preposition used, there is not enough information to conclude that one or the other of these differences is what distinguishes the PATHELEMENT from the PLACE. If, on the other hand, the model had already been generalized to accept any THING represented by the noun phrase, it would be safe to conclude that the different preposition is the important feature. The second rule for evaluating whether an example may be used as a near miss, therefore, is that it be different from the model in only one significant way.

A third rule that the system uses to decide whether to treat an example as a near miss is that the example must not already be rejected by the model in question. In general, the constraints created by near-miss learning add negative qualifications, e.g. MUST-NOT-BE-A. If a category is already rejected by a model, adding such negative constraints is redundant.

The fourth and final rule for admitting near misses arises as a precaution to guard against over-constraining the models. If the trainer provides multiple descriptions to accompany a sentence that fall into the categories of different models, it may very well be the case that these should serve as complimentary near misses; that is, this compound training example might be

3 In section 3.6 I discuss an implementation detail pertaining to `sequences`. In the present implementation of Lance, all training examples pertaining to forms that may have a variable number of elements must have exactly one of the type of element that may be multiplied, and the semantic form corresponding to such an element must be a member of a `sequence`. For the purpose of clarity in the present discussion, I omit this detail.

treated just as would a sequence of training examples, each containing the sentence and one of the descriptions. This is not the behavior I chose, though, because if there is no difference between the word senses used in the compound description (and occasionally even if there are differences) then the same partially-described sentence may end up being presented as an example and as a counterexample to a single model. This behavior would violate the felicity conditions upon which the Arch-Learning paradigm rests, and so it cannot be permitted.

Once a near miss has been identified according to these rules, the pair thread bundles associated with the two nodes containing the critical difference are passed to a procedure `specialize`. In a similar fashion, when examples of a category are presented, all nodes are aligned and the pairs of bundles are each presented to the procedure `generalize`. The procedures `generalize` and `specialize` are outlined below.

```
Procedure generalize:
Inputs: Bundle model, Bundle example
      if all constraints in model are satisfied by example:
            return, making no modifications
      else if the model is unconstrained:
            [Use a combinatorial optimization technique to map all of
            the threads in model to a thread in the example, such that
            the total edit distance between bundles is minimized. climb
            tree on each of the pairs in this map, and convert the
            result of each tree climbing to a MUST_BE_A or
            MUST_HAVE_FEATURE]
      else:
            build a Map, m, of the violated constraints and
                  the threads in example that violate them, if
                  applicable.
            for each model thread, mt, and example thread, et, in m:
                  get the constraint, c, that modifies mt.
                  if c is MUST_BE_A or MUST_HAVE_FEATURE:
                        [climb tree heuristic if possible, otherwise
                        expand set. if it makes no sense to expand the
                        set just drop the constraint all together (this
                        not exactly like drop-link, because we never
                        know if there is an exhaustive set)]
                  else if c is MUST_NOT_BE_A or MUST_NOT_HAVE_FEATURE:
                        remove mt from model
                  else if c is MUST_HAVE_FEATURE_IN_SET or
                               MUST_BE_IN_SET :
                        [expand set, either by adding another constraint
                        thread or by generalizing (via climb-tree) an
                        extant constraint thread.]
```

```
                else:
                      error: unknown constraint.
          end for
     end if
end procedure

Procedure specialize:
Inputs: Bundle model, Bundle counter
      if counterexample is already rejected by the model:
            return, performing no specialization.
      else if the model is yet unconstrained:
            [possibilities:
            1: The counterexample contains one additional thread that
            the model lacks.  Reduce the model to a MUST_BE on all
            threads and a MUST_NOT on the lacking one.
            2: The counterexample lacks a thread that the model
            contains.  reduce the model to a must-be on all threads
            3: the counterexample has one thread that is somehow
            different from a thread in the model. find the fork point
            on the disparity, add a must-be on one symbol up on the
            model side, and add a must-be for all other threads in the
            model.  Alternatively, if the fork point occurs at the end
            of the model's thread, add a MUST_NOT constraint one symbol
            up on the counterexample side.]
            4: otherwise, the counterexample is not a near miss
      else:
            Obtain a list, l, of threads that are not explicitly
            sanctioned by constraints in the model.
            if l has more than one element:
                  return, because the counterexample is not a near miss
            else if the l has exactly one element:
                  add to the model a MUST_NOT_BE or
      MUST_NOT_HAVE_FEATURE
                  thread for the element of l
            else:
                  Obtain a list l2, of threads that are not *identical* to
                  some positive-constraint thread in the model, minus
                  the constraint label itself.
                  if l2 has more than one element:
                        return, because the counterexample is not a near
                        miss
                  else if l2 has exactly one element:
                        add to the model a MUST_NOT_BE or
                        MUST_NOT_HAVE_FEATURE thread for the element of
                        l2
                  else:
                        [here, the counterexample is exactly the same as
                        the model.  depending on whether a fail-soft or
                        fail-fast behavior is desired, we may choose to
                        label the model as destroyed (for fail-fast) or
                        do nothing (for fail-soft)]
                  end if
            end if
```

```
        end if
    end procedure
```

In the pseudocode of `generalize` and `specialize`, I have referred to six explicit constraints: MUST_BE_A, MUST_NOT_BE_A, MUST_BE_IN_SET, MUST_HAVE_FEATURE, MUST_NOT_HAVE_FEATURE, and MUST_HAVE_FEATURE_IN_SET. The feature constraints are identical to their counterparts in all respects except that feature constraints apply to threads whose first symbol is FEATURE. Such threads are used to augment description bundles without incurring ambiguity as to the dominant type of the bundle. MUST_NOT constraints indicate that a certain thread (or any subcategory thereof) may not be present in a bundle in order for the constraints to be satisfied. Positive constraints indicate the contrary; that a certain thread or subcategory must be present. The SET constraints indicate that a bundle must have at least one of the threads, or a subcategory thereof, in order to qualify. The set of FEATUREs is considered independently from the set of other types.

The procedures for learning and recognition in the base case are easily generalized to apply to compound phrases and descriptions. As mentioned in the context of learning PLACE→ON, the first generalization is that the process of reference resolution that occurs when the model is instantiated honors equivalence of entire substructures cast in the meta-representation rather than just bundles of threads. During matching, these entire substructures are replaced by the corresponding substructures in the partially-described tree rather than just their bundles. When generalizing and specializing, the same procedures outlined above in the context of bundle generalization and specialization are applied only to the top-level bundle of the stored description. This is a compromise that I had to make in order to facilitate debugging a complicated system; using only the root-level bundle in model-building procedures precludes, for example, developing a model with the constraint: *the path element described by the prepositional phrase that modifies the verb phrase must have a geological formation as the reference object*. While it is conceivable that a certain representation may require distinctions on such a fine granularity, my system is currently incapable of judgments on this level of specificity. My implementation is perfectly capable, however, of capturing constraints such as: *the path element described by the prepositional phrase* *that modifies the verb phrase must have the path function FROM*, because the path function is specified in the top-level bundle of a PATHELEMENT.

3.4 Approximating High-Order Constraints.

As discussed in Section 2, the Arch-Learning approach to learning from examples and near misses is best suited to learning first-order constraints. When constraints span many variables in a model, I augment the first-order model by approximate methods.

The mode of this augmentation is to use distance-like[4] scoring methods to measure similarity between a description generated by a first-order model and the examples that were used to train that first-order model. All descriptions emitted by such an augmented model have a distance score that intimates how plausible that description might be, based on prior experience.

My method for computing a distance-like score between two descriptions cast in the meta-representation is cast in terms of combinatorial optimization and is largely empirical. The basic procedure is outlined as follows:

```
Procedure find-distance:
Inputs: Description D0, Description D1
      if optimize(thread-dist, D0.bundle, D1.bundle ) equals 0:
            if D0, D1 both have subcomponents:
                  return optimize(find-distance,
                                    D0.subcomponents
                                    D1.subcomponents )
            else:
                  return 0
            end if
      else if D0, D1 are both Sequences:
            if optimize(find-distance,
                  D0.subcomponents
                  D1.subcomponents) < SEQUENCE-THRESHOLD :
                  return optimize(thread-dist,D0.bundle,D1.bundle)
            end if
      else if D0, D1 are both Relations:
            if find-distance(D0.subject,D1.subject)
                        < DERIVATIVE-THRESHOLD
                        AND
```

4 My scoring methods do not conform to the subadditivity constraint required of true distance metrics.

```
                find-distance(D0.object,D1.object)
                        < RELATION-THRESHOLD:
                return optimize(thread-dist,D0.bundle,D1.bundle)
            end if
        else if D0, D1 are both Derivatives:
            if find-distance(D0.subject,D1.subject)
                        < DERIVATIVE-THRESHOLD :
                    return optimize(thread-dist,D0.bundle,D1.bundle)
            end if
        else if D0,D1 are both plain Things:
            return optimize(thread-dist,D0.bundle,D1.bundle)
        end if
        return MAX-DISTANCE
    end procedure find-distance
```

In the pseudocode, the procedure `optimize` is a combinatorial-optimization algorithm that takes two sets of objects to compare and a cost function, and returns the average cost of the optimal assignment. The procedure `thread-dist` returns `MAX-DISTANCE` multiplied by the ratio of the number overlapping symbols in both threads (counting from general to specific) to the sum of the lengths of the two threads.

4.5 Priming Mechanisms

By sending collections of descriptions to Lance's priming port, external components may influence the behavior of Lance as it recognizes sentences. If Lance's recognition mechanism contains a set of descriptions (the “primed” state) then it will preferentially assign descriptions from that set to phrases when it can. This can prevent unlikely interpretations of subordinate clauses from being pruned by Lance's recognition process—which has no other knowledge of context—if they are regarded more plausible by expert components that have access to the cross-representational domain. This ability to influence the recognition process based on partial or anticipated knowledge will be a much-needed feature in the Gauntlet System as research in concept-formation progresses.

In addition to permitting interpretations that are unlikely based on training experience alone, priming mechanisms may prove necessary to derive even those descriptions that the system has encountered before, if the representational space is not sparse as I have assumed, or if first-order constraints are not as well represented in future future additions to Gauntlet's repertoire of representational frameworks as they are in the examples that I have chosen.

If the representational space is not sparse, or if first-order constraints are not strong enough to eliminate the majority of irrelevant semantic descriptions of a phrase, then each model that admits a surface form is likely to be active at a recognition step that concerns that surface form, regardless of how many such models are present. The process of recognition from the bottom up would, in this scenario, have an amplifying rather than the desired attenuating effect on the number of descriptions assigned to each successive subsuming phrase. This amplification, coupled with the word-sense ambiguity problem inherent in natural language, would certainly lead to unacceptable computational performance on sentences of modest length.[5] In such a scenario, it is infeasible to produce all valid results, and unlikely, due to the approximate nature of the filtering that occurs at each recognition stage, that the intended meaning of the sentence would make its way to the final stage of recognition. Priming could provide a way to keep the search headed in the right direction in situations when excessive branching might occur.

3.6 Example Training Exercise

This section consists of annotated screen captures from Lance's interactive training environment that depict a complete training sequence in which Lance acquires models for TRAJECTORY-SPACES, TRANSITION-SPACES and their subcomponents.

Figure 3-3 is a view from Lance's graphical training-sequence editor. The panel on the left contains semantic descriptions paired with images, The panel at the right may display an editor in which descriptions may be paired with parse trees, or it may display a condensed view of the entire training sequence. Any semantic description may be augmented by an optional image or text.

5 Assuming that the parse tree is of uniform depth, and that every recognition stage yields a constant number, greater than one, of descriptions for every permutation of inputs, the performance would be of roughly exponential complexity in the length of the sentence.

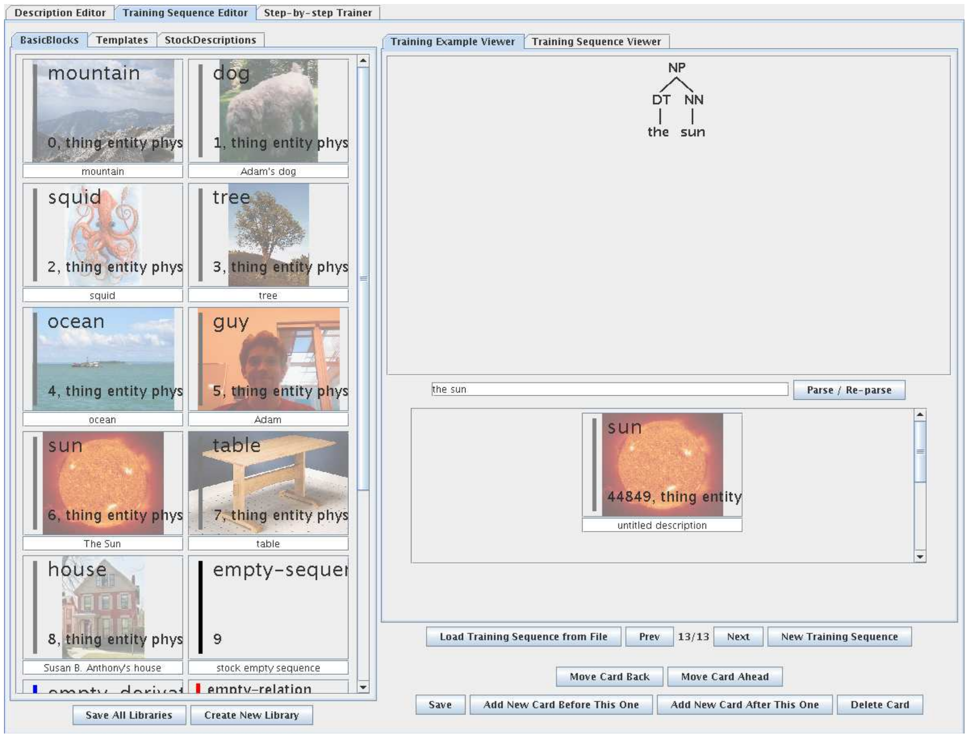


Figure 3-3: Lance Training-sequence editor, showing single training example

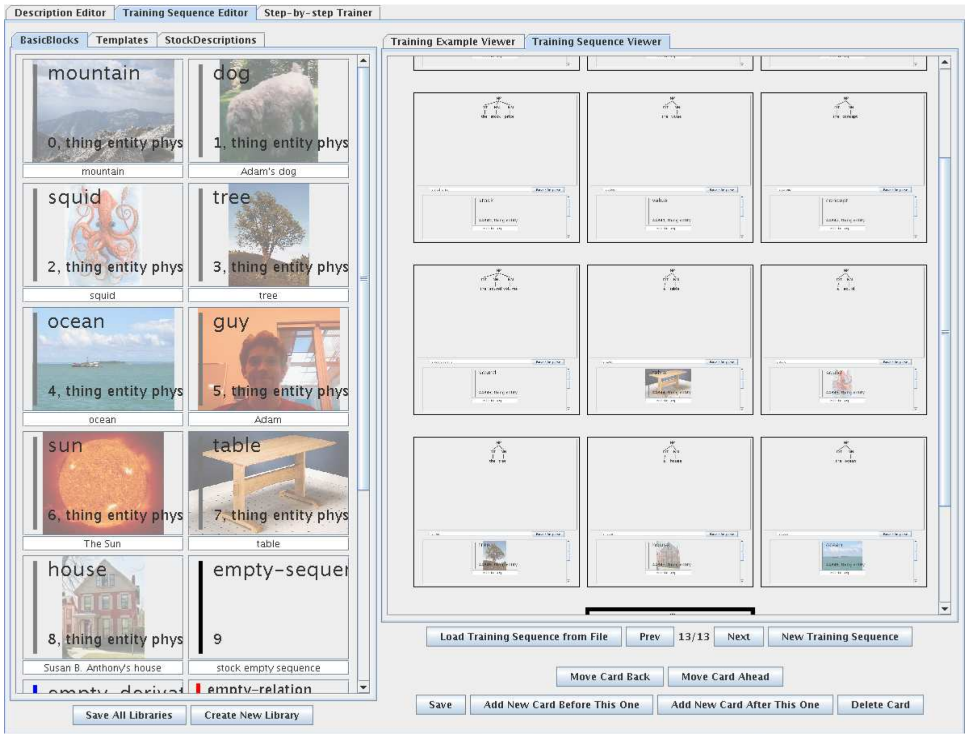
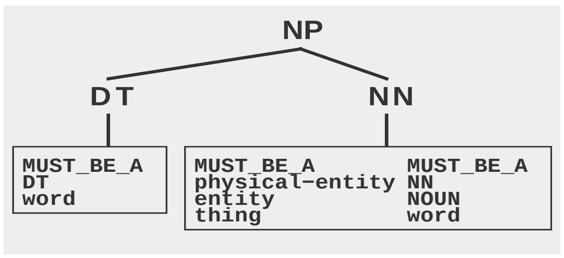
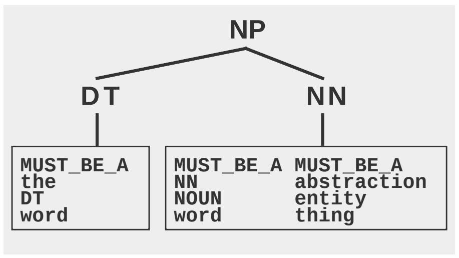


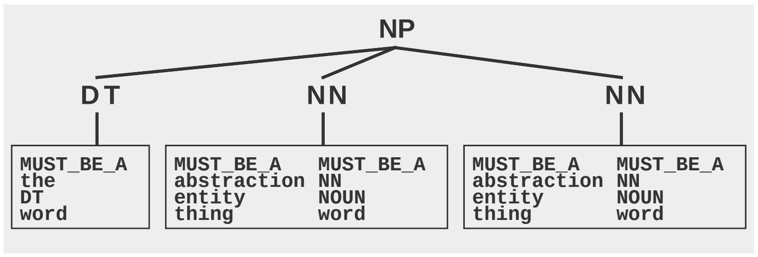
Figure 3-4: Lance Training-sequence editor, showing entire training sequence

After training on all of the examples in a training sequence consisting of noun phrases, Lance acquires the models shown in Figure 3-5. Figure 3-5 (a) is a model of physical entities represented by noun phrases, whereas Figures 3-5(b) and (c) are both models of abstractions. The mapping between the syntax and semantics in 3-5 (a) and 3-5 (b) is the same; it is redundant, therefore, to acquire separate models for each of these mappings. It would not be appropriate to lower the similarity threshold such that these models would be joined, however, because, as noted in this Section, such generality would lead to ambiguity in models of more complex mappings.





(a) (b)



(c)

Figure 3-5: Models of Noun Phrases

After acquiring models of THINGS or the type represented by noun phrases, and related expressions such as compositions of noun phrases and prepositional phrases that represent parts of objects, e.g. “the top of the tree,” the system is in a position to learn about PLACES, of the sort often represented by prepositional phrases such as “on the fence,” or “near the top of the tree.” There are 10 places currently in the ontology: NEXT-TO, FAR-FROM, NEAR, LEFT-OF, RIGHT-OF, ON, UNDER, ABOVE, IN, and AT. Shown in Figure 3-6 is one example of a PLACE model.

Figure 3-6: Model of phrases meaning UNDER

As noted in Section 2, the Stanford Parser handles certain PLACE syntax differently depending on whether it is part of a larger context or parsed alone. For example “next to the table” produces an adverbial phrase when parsed alone, and a prepositional phrase with a preceding particle when parsed as part of a verb phrase. To address this problem, I have simply transformed all of the output of the Stanford Parser to eliminate the ambiguity. The implementation searches the Stanford Parser's output for the particle, prepositional-phrase constructions and replaces any instances with the adverbial-phrase form.

The system may now acquire models of PATHELEMENTS. There are five PATHELEMENTS presently in the ontology: TO, FROM, TOWARD, AWAY-FROM and VIA. Most PATHELEMENTS have an implicit PLACE, so it is unnecessary to have acquired models of explicit PLACE prepositions in order to acquire models of PATHELEMENTS. Figures 3-7 and 3-8 show two models of surface forms corresponding to PATHELEMENT→TOWARD semantic descriptions. Figure 3-7 shows the typical form, containing the word *toward*. In Figure 3-8, the PATHELEMENT→TOWARD is implicit in the word *down*.

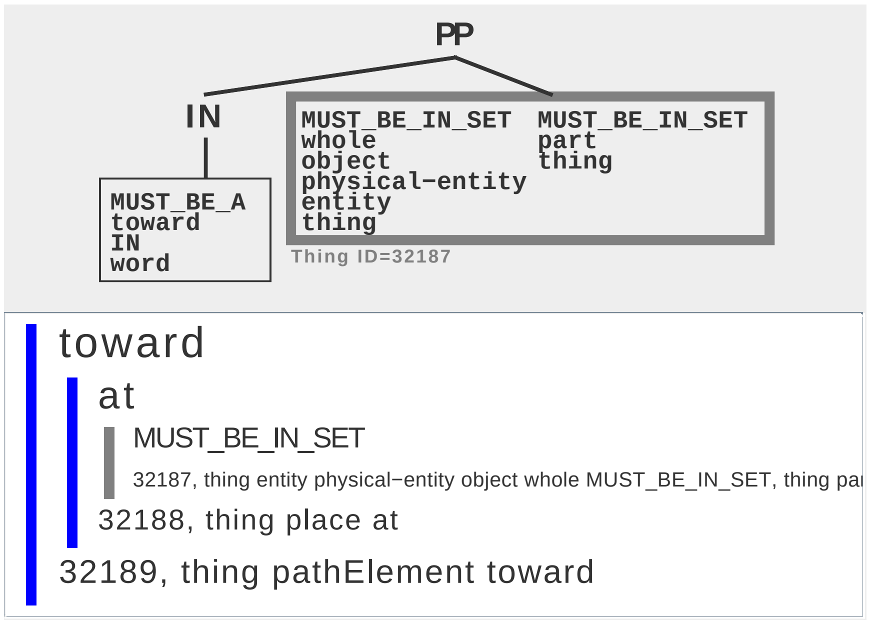
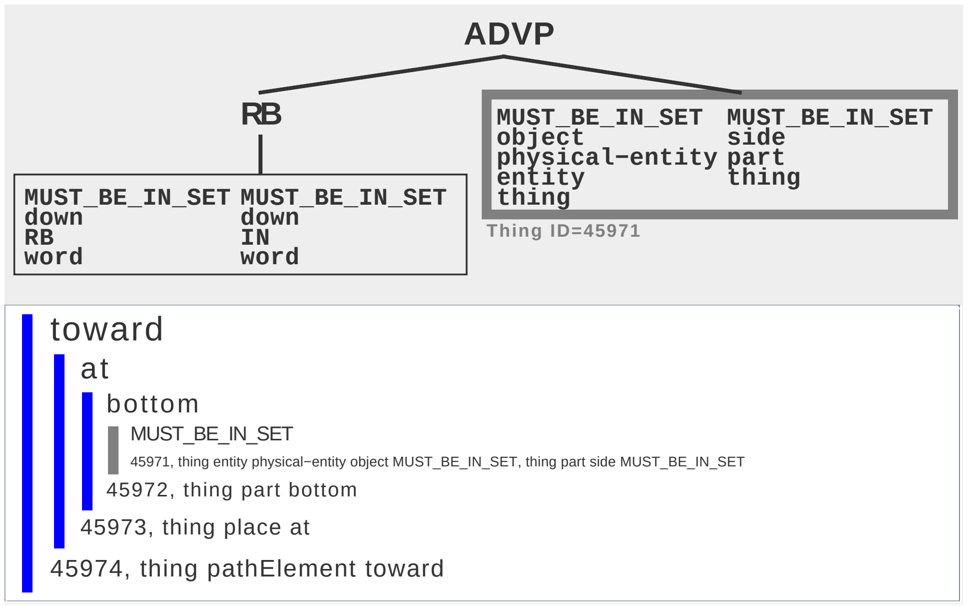


Figure 3-7: Model of explicit PATHELEMENT→TOWARD phrases



ADVP
RB
MUST_BE_IN_SET down RB word
MUST_BE_IN_SET down IN word
MUST_BE_IN_SET object physical-entity entity thing
MUST_BE_IN_SET side part thing
Thing ID=45971
toward
at
bottom
MUST_BE_IN_SET
45971, thing entity physical−entity object MUST_BE_IN_SET, thing part side MUST_BE_IN_SET
45972, thing part bottom
45973, thing place at
45974, thing pathElement toward

Figure 3-8: Model of implicit PATHELEMENT→TOWARD using the word *down*

In Figure 3-8, note that the model permits two interpretations of *down*, both as an adverb and as a preposition. This results from peculiarities of the training data: the Stanford Parser parses the phrase "down the hill" as an adverbial phrase, whereas it parses "down the side of the mountain" as a prepositional phrase. Despite this structural ambiguity, Lance has distilled the essential information from the training examples: if a phrase contains the word *down* followed by a phrase that represents a physical object or the side of such an object, then the compound phrase represents a PATHELEMENT→TOWARD in which the destination is a PART→BOTTOM of the entity represented by the subsumed phrase.

It is relatively straightforward to acquire TRAJECTORY- and TRANSITION-SPACES. In its present configuration, Lance requires surface forms that may contain a variable number of elements to be presented in a simplified form—TRAJECTORY-SPACES, for example, which may have zero or more PATHELEMENTS, must be trained with exactly one modifying pathelement. This limitation applies to training only; once it has acquired TRAJECTORY-SPACE forms, Lance may recognize forms with any number of modifying PATHELEMENTS.[6] Figure 3-9 depicts the

6 This behavior results from a rule that I have hard-wired into the system: if a training parse tree has one branch whose description corresponds to an element of a `sequence`, then during recognition any number of branches are permitted in the `sequence`, provided that they all match the pattern of the single branch of the example. I believe that modifying this behavior to permit flexibility in training as well as in recognition would not present any difficult problems but may compromise the clarity of the training procedures.

model of TRAJECTORY-SPACES and Figure 3-10 depicts the model of TRANSITION-SPACES.

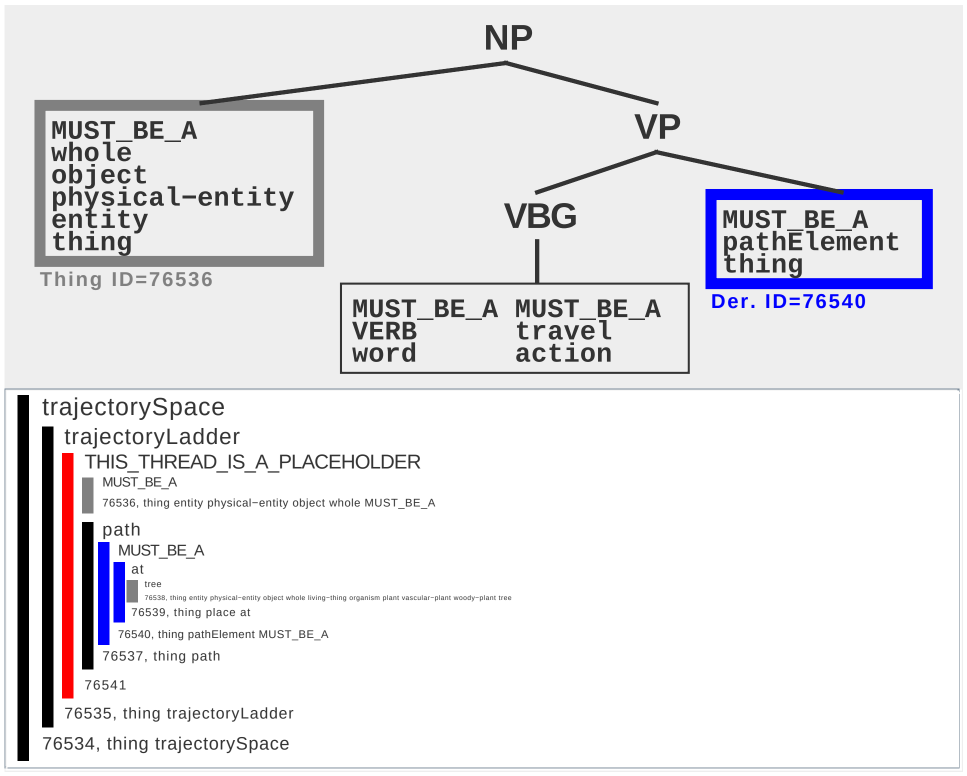

NP
MUST_BE_A
whole
object
physical-entity
entity
thing
Thing ID=76536
VP
VBG
MUST_BE_A
VERB
word
MUST_BE_A
travel
action
MUST_BE_A
pathElement
thing
Der. ID=76540
trajectorySpace
trajectoryLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
76536, thing entity physical−entity object whole MUST_BE_A
path
MUST_BE_A
at
tree
76538, thing entity physical−entity object whole living−thing organism plant vascular−plant woody−plant tree
76539, thing place at
76540, thing pathElement MUST_BE_A
76537, thing path
76541
76535, thing trajectoryLadder
76534, thing trajectorySpace

Figure 3-9: Model of a typical TRAJECTORY-SPACE form

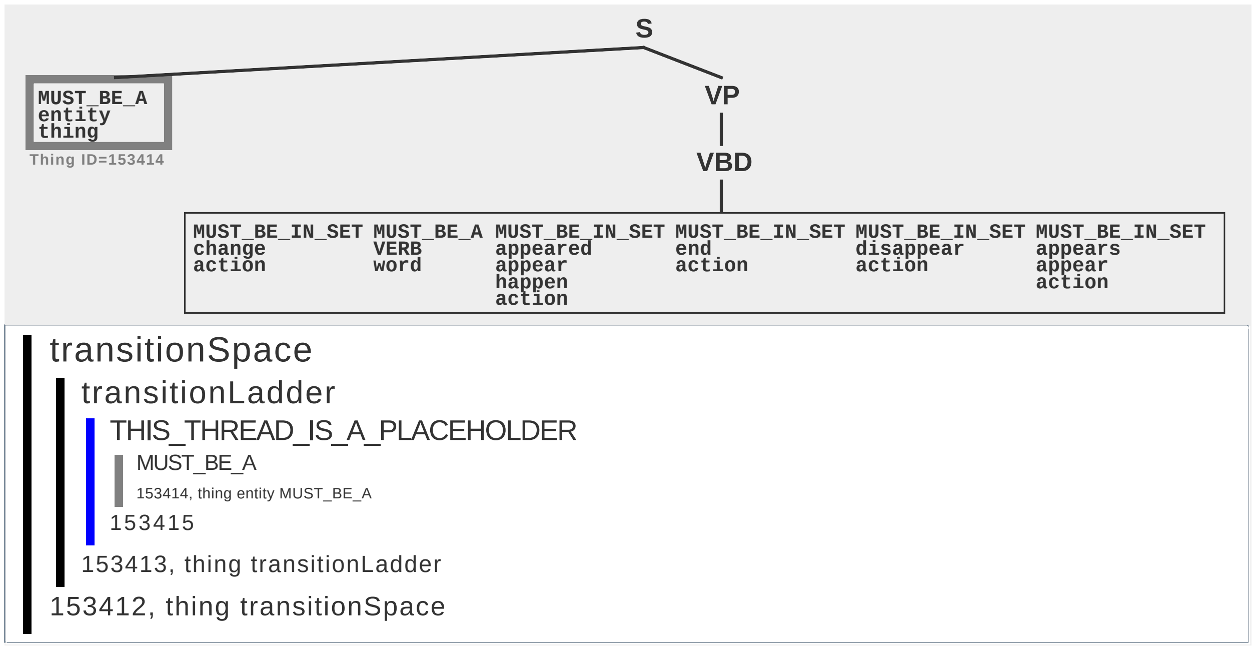

S
VP
VBD
MUST_BE_A
entity
thing
Thing ID=153414
MUST_BE_IN_SET change action
MUST_BE_A VERB word
MUST_BE_IN_SET appeared appear happen action
MUST_BE_IN_SET end action
MUST_BE_IN_SET disappear action
MUST_BE_IN_SET appears appear action
transitionSpace
transitionLadder
THIS_THREAD_IS_A_PLACEHOLDER
MUST_BE_A
153414, thing entity MUST_BE_A
153415
153413, thing transitionLadder
153412, thing transitionSpace

Figure 3-10: Model of a TRANSITION-SPACE form

Note that, in Figure 3-9, the top-level label of the phrasal structure corresponding to the TRAJECTORY-SPACE model is that of a noun phrase, rather than that of a sentence. This is an artifact of the training data; the first example of a TRAJECTORY-SPACE was “the dog running to the tree,” in which the gerund form of the verb forces a noun-phrase parse of the construction. Subsequent examples, containing past- or present-tense verbs rather than gerunds, generalized the verb form to any word that is of the subsuming class VERB. Because Lance ignores the Stanford Parser's phrase tags, however, no generalization caused the NP tag in the model to change when Lance processed the example in which the top-level phrasal label was S.

Having learned about TRAJECTORY- and TRANSITION-SPACES, Lance is in a position to learn about CAUSE. As noted in Section 2, typical sentences expressing causal relationships contain the word *because* in conjunction with an SBAR clause that obscures the verb-phrase of the TRAJECTORY, TRANSITION, or other construction that precedes it. I simply circumvent this difficulty by transforming SBAR constructions according to a rule that moves the two logical parts of such expressions into positions such that they are easily recognized by Lance. Figures 3-11 and

3-12 depict two CAUSE models, for different syntax structures.

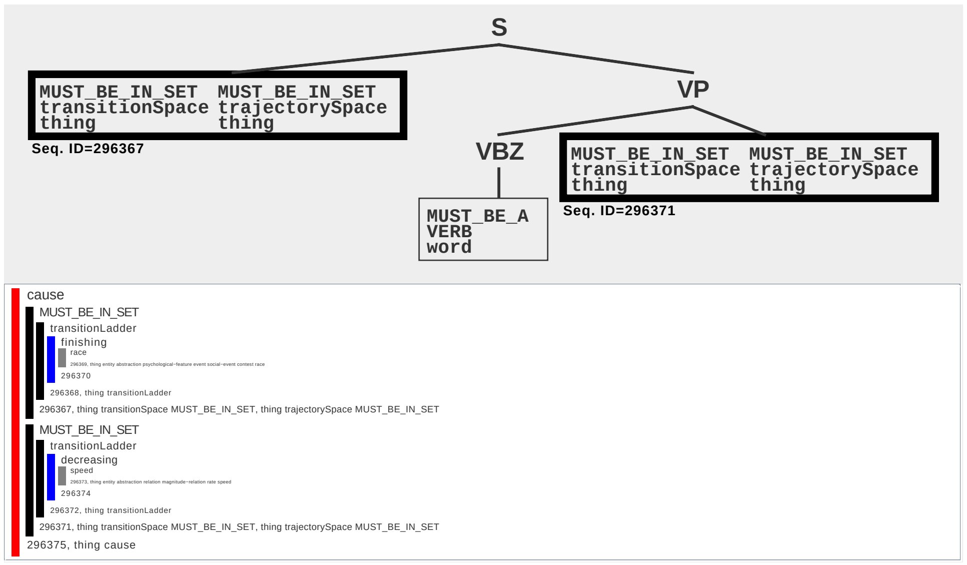

S
MUST_BE_IN_SET transitionSpace thing
MUST_BE_IN_SET trajectorySpace thing
Seq. ID=296367
VP
VBZ
MUST_BE_A VERB word
MUST_BE_IN_SET transitionSpace thing
MUST_BE_IN_SET trajectorySpace thing
Seq. ID=296371
cause
MUST_BE_IN_SET
transitionLadder
finishing
race
296369, thing entity abstraction psychological−feature event social−event contest race
296370
296368, thing transitionLadder
296367, thing transitionSpace MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET
MUST_BE_IN_SET
transitionLadder
decreasing
speed
296373, thing entity abstraction relation magnitude−relation rate speed
296374
296372, thing transitionLadder
296371, thing transitionSpace MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET
296375, thing cause

Figure 3-11: A CAUSE model

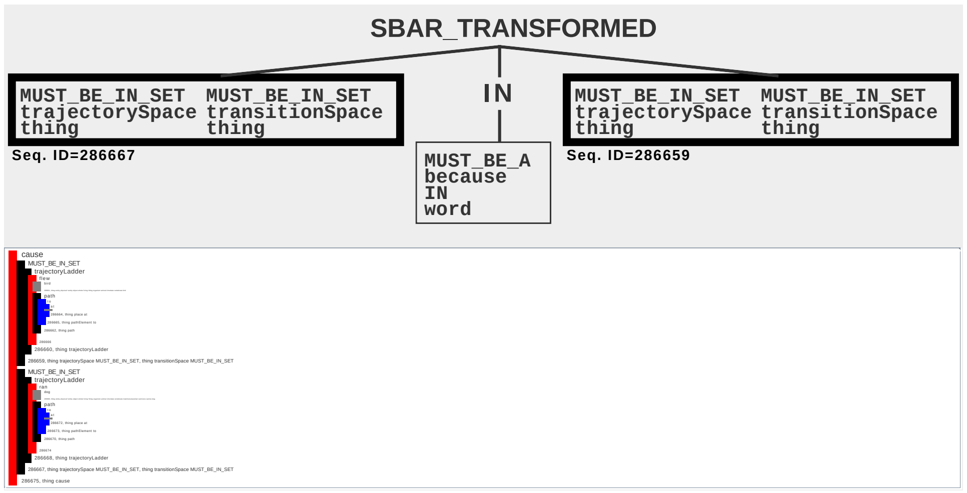
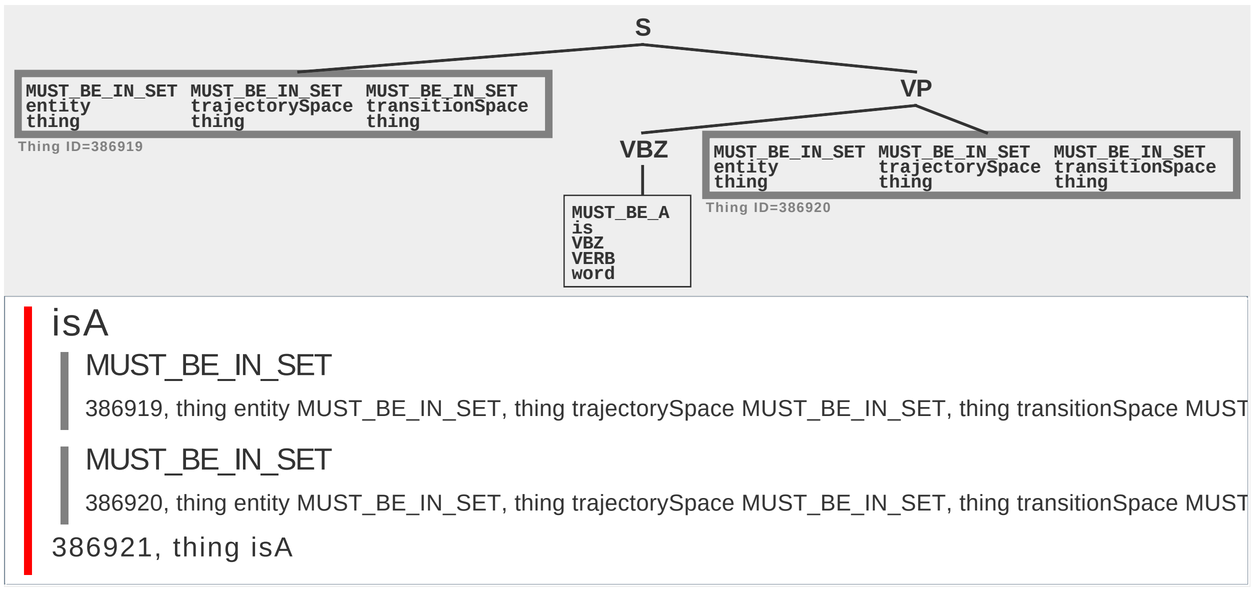

SBAR_TRANSFORMED
MUST_BE_IN_SET trajectorySpace thing
MUST_BE_IN_SET transitionSpace thing
Seq. ID=286667
IN
MUST_BE_A because IN word
MUST_BE_IN_SET trajectorySpace thing
MUST_BE_IN_SET transitionSpace thing
Seq. ID=286659
cause
MUST_BE_IN_SET
trajectoryLadder
flew
bird
path
to
at
286664, thing place at
286665, thing pathElement to
286662, thing path
286666
286660, thing trajectoryLadder
286659, thing trajectorySpace MUST_BE_IN_SET, thing transitionSpace MUST_BE_IN_SET
MUST_BE_IN_SET
trajectoryLadder
ran
dog
path
to
at
286672, thing place at
286673, thing pathElement to
286670, thing path
286674
286668, thing trajectoryLadder
286667, thing trajectorySpace MUST_BE_IN_SET, thing transitionSpace MUST_BE_IN_SET
286675, thing cause

Figure 3-12: Another CAUSE model

Note the overgeneralized word that the word that joins the two parts of the CAUSE model in Figure 3-11. This model resulted from two training examples: "the dog running caused the bird flying to the tree" and "the value decreasing causes the price decreasing." Because of the difference in verb tense between *caused* and *causes,* the climb-tree heuristic forced the model to include the intersection of these two threads. Because these are WORD threads—which pertain to part of speech rather than semantic information—the point of intersection is of the general category VERB. Such overgeneralized threads are occasionally unavoidable. In this case, because the system has no knowledge of which semantic sense of the word *cause* pertains to the model, it has only the WORD threads to manipulate. If this became a problem, the CAUSE model could be modified to take a lexical thread into the bundle of the root `relation`, rather than using a symbol that is not related to a lexical item. For the purpose of this example training exercise I leave it as is, because it serves to showcase the specialization routine in the example of IS-A models.

There are several situations in which we would present Gauntlet with sentences of the form "A ___ is a ___," in which the blanks may be occupied by simple noun phrases such as "a dog" or compound phrases such as "a man hiking." We may wish to teach the system about a new type of dog called a bulldog, in which case we would want the system to recognize "A bulldog is a dog" as a cue that triggers formation of a Thread Memory. In another scenario, we might teach Gauntlet what the trajectory verb *hike* means by issuing the declarative statement, "A man hiking is a man walking up the side of a mountain." These two scenarios call for a general form of description called IS-A, depicted as a model in Figure 3-13.


S
MUST_BE_IN_SET entity thing
MUST_BE_IN_SET trajectorySpace thing
MUST_BE_IN_SET transitionSpace thing
Thing ID=386919
VP
VBZ
MUST_BE_A is VBZ VERB word
MUST_BE_IN_SET entity thing
MUST_BE_IN_SET trajectorySpace thing
MUST_BE_IN_SET transitionSpace thing
Thing ID=386920

```
isA
  MUST_BE_IN_SET
  386919, thing entity MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET, thing transitionSpace MUST
  MUST_BE_IN_SET
  386920, thing entity MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET, thing transitionSpace MUST
386921, thing isA
```

Figure 3-13: Model of IS-A relationships

Some of the sentences admitted by the model of Figure 3-13 would also be admitted by the CAUSE model of Figure 3-12. As noted, this results from the overgeneralized WORD thread in the CAUSE model. The specialization feature of Lance aims to compensate for such overgeneralized threads by generating near-miss counterexamples from examples of another category. In order for a training pair of one category *A* to be a near miss for another category *B*, it must:

- admit examples that *B* ordinarily admits
- consist almost entirely of nodes that satisfy *B*'s constraints exactly, that is, by virtue of thread equality not simply by thread overlap.
- contain exactly one node that satisfies *B*'s constraints by virtue of overlap, not equality. This node is the near-miss node.

In the example of Figures 3-12 and 3-13, the near-miss node is the verb that connects the two elements of the CAUSE or IS-A description. During the same training step that generalized the model of Figure 3-13, Lance specialized the CAUSE model to include the additional constraint in Figure 3-14.

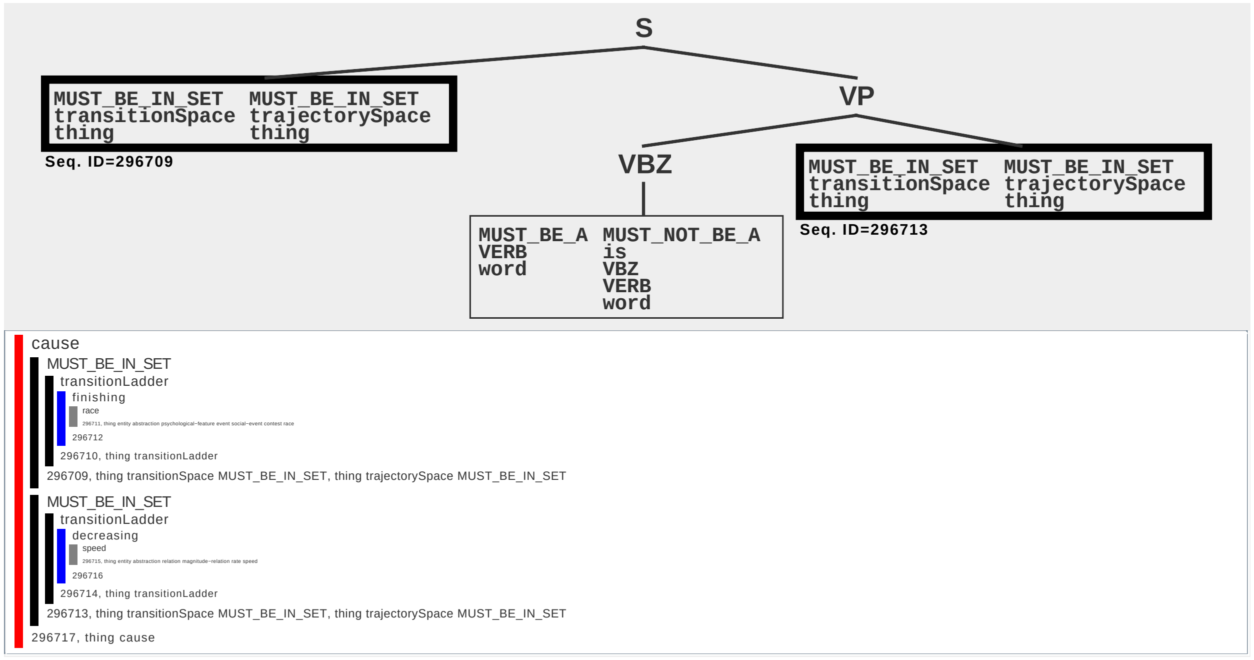

S
VP
VBZ
MUST_BE_IN_SET transitionSpace thing
MUST_BE_IN_SET trajectorySpace thing
Seq. ID=296709
MUST_BE_A VERB word
MUST_NOT_BE_A is VBZ VERB word
MUST_BE_IN_SET transitionSpace thing
MUST_BE_IN_SET trajectorySpace thing
Seq. ID=296713
cause
MUST_BE_IN_SET
transitionLadder
finishing
race
296711, thing entity abstraction psychological−feature event social−event contest race
296712
296710, thing transitionLadder
296709, thing transitionSpace MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET
MUST_BE_IN_SET
transitionLadder
decreasing
speed
296715, thing entity abstraction relation magnitude−relation rate speed
296716
296714, thing transitionLadder
296713, thing transitionSpace MUST_BE_IN_SET, thing trajectorySpace MUST_BE_IN_SET
296717, thing cause

Figure 3-14: Specialization in the CAUSE model in response to a near miss

4. Background

This section is a survey of natural-language science and technology as it applies to the design of Lance. Section 4.1 pertains to parsing theory and practice, with emphasis on the particular technology upon which Lance is built. Section 4.2 is devoted to a small selection of representational frameworks that exemplify the types of descriptions that Lance is equipped to learn to extract from language. A thorough treatment of either of these topics would overwhelm the subject matter of this thesis. The purpose of this section, therefore, is to introduce just those features of each topic that factor prominently into the design of Lance.

4.1 Parsing Theory and Technology

In order to model the ways in which the structure of language gives rise to semantic descriptions, it is essential to have methods to extract the underlying structure from the surface form of language. Parsing is the process of extracting this structure. To motivate a brief discussion of the challenges that parsing poses and the technology that has been applied to meet these challenges, consider the example of a sentence and its associated parse trees in Figure 4.1.

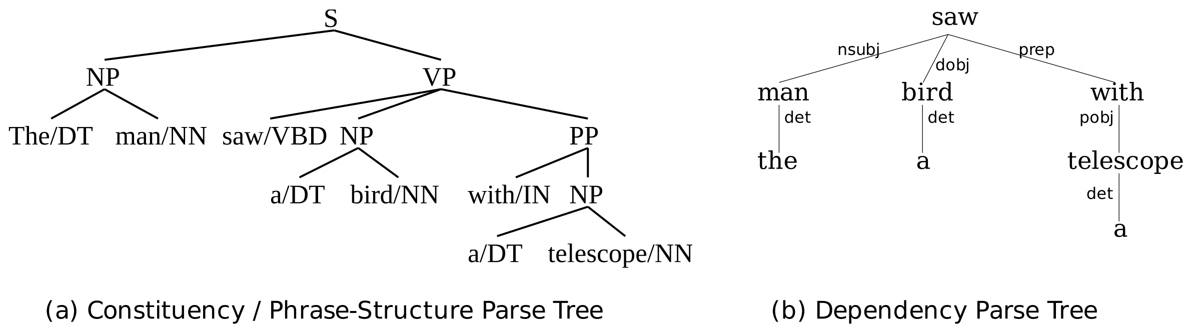


Figure 4.1 Parse Structures of "The man saw a bird with a telescope"

Figure 4.1 (a) is a phrase-structure description of the sentence. Phrase structure identifies the grammatical constituency of a sentence according to the rules of a Context-Free Grammar (CFG).

The hierarchies that CFGs sanction specify which phrases are subsumed by which other phrases, as governed by recursively applicable rules that ground ultimately in part-of-speech (POS) categories. Figure 4.1 (b) depicts a dependency parse tree, in which the structure is governed by a notion of word-based attraction and selection. The links in dependency parse point toward the modifiers of words. These modifiers, such as the word "the" in the example, are called the *dependents*, and the words they modify are the *heads*.

The difficulty in parsing arises from real or apparent ambiguity in language. In the example of Figure 4.1, it is clear that the sentence describes an event where a man used a telescope to view a bird.   Such clairvoyance comes only by virtue of that birds do not usually have telescopes, however, and that telescopes may be used to view objects. This type of commonsense knowledge that we often take for granted, along with other non-linguistic judgments that contribute to parsing, seem to make *correct* parsing of a sentence an AI-complete problem.

4.1.1 Statistical vs. Principle-Based Parsing

My implementation depends on an existing parser to create phrase-structure parse trees from sentences and fragments. Because accurate parsing is still very much an open problem, practicality factored most heavily into the decision to use a parser based on a Probabilistic Context-Free Grammar (PCFG), rather than several more elegant solutions currently under development.

PCFG-based parsers are statistically trained on large corpora of hand-parsed sentences. This training allows the parsers to acquire an underlying grammar that governs the productions used to generate the parse trees in the training corpora. To summarize the functionality of PCFG-based parsers very tersely, the goal of a PCFG is to maximize the conditional probability of a produced parse tree, given the observed frequencies of the productions in the training corpus. A PCFG may incorporate statistical information pertaining to lexical features in the corpus in order to improve performance, but any apparent understanding of the constraints of the lexicon is illusory. The preferred alternative to statistical parsing is a principle-based approach that is informed by linguistic principles rather than by statistical data extracted from a large corpus. Niyogi [2005] provided a critique of the present trend to apply traditional machine-learning techniques to problems of computational linguistics.

Of the principle-based parsers currently available, none has coverage comparable to its statistical counterparts. An expedient solution, therefore, is to rely on a statistical parser until a better alternative becomes available. The Stanford Parser is a feature-rich statistical parser that supports phrase-structure and dependency parsing, and often produces correct fine structure even in light of frequent errors in the overall parse structure, especially pertaining to difficult problems such as prepositional-phrase attachment. Correctness on a small scale is important for my implementation to function properly. In Sections 2 and 3 I introduce some ways to address the problem of erroneous parses by way of prepositional-phrase rearrangement.

4.2 Semantic Representations

The stated purpose of Lance is to extract descriptions—that is, instances of representational frameworks—from sentences and phrases. In this section I characterize the representational frameworks. Because the Gauntlet System aims, in part, to explore the space of representational frameworks in order to achieve the best coverage, it would be infeasible to describe completely the frameworks that Lance must operate on. One way to address this inherently underspecified domain is to impose a structure on the frameworks to achieve a kind of meta-representation that constrains the structure of the representations themselves. This is the approach taken in the design of Lance. The mechanics of the chosen meta-representation are presented in Section 2. Presented here are three representations that the meta-representation must accommodate, because the virtues of each suggest that each will be an indispensable member of the Gauntlet System's repertoire. The frameworks described are Jackendoff's Lexical Conceptual Structures (LCS), Borchardt's Transition Spaces, and Vaina's and Greenblatt's Thread Memory. The choice to focus on these three representations to motivate the design is based on their known wide descriptive coverage: Mark Seifter has demonstrated that in a corpus of roughly 50,000 sentences from the Wall Street Journal, roughly one description cast in a subset of LCS is found per every four sentences processed.[Seifter 2007] The descriptions here are by no means comprehensive, but they motivate the considerations involved in specifying Lance's design parameters by highlighting the aspects of each framework that make it powerful.

4.2.1 Lexical Conceptual Structures

Developed by Ray Jackendoff[Jackendoff 1983], LCS is a framework that embodies the philosophy that languages are shrunk-wrapped around the ways in which observable phenomena project in the mind. This notion that language naturally conveys the types of information that the human mind is predisposed to perceive allows LCS to benefit from cognitive psychology, whereas traditional semantics is confined to the realm of classical logic. That Jackendoff devoted special attention to the ways in which visual processing accounts for linguistic forms makes LCS an especially strong framework for the Gauntlet System, which is focused on the confluence of language and vision. The following example illustrates how the meaning of the sentence "The squirrel climbed the flagpole" is represented by LCS:

$$\left[_{\text{Event}}\ \text{GO}\left(\left[_{\text{Thing}}\ \text{SQUIRREL}\right],\left[_{\text{Path}}\ \text{UP}\left[_{\text{Thing}}\ \text{FLAGPOLE}\right]\right]\right)\right]$$

The example demonstrates two strengths of LCS. The first is that the description evokes a trajectory, that is, motion of an object along a path. In the example, the path, the moving object, and its motion all refer to physical attributes, but this need not be the case in general. There is strong psychological evidence that humans are adept at reasoning in terms of spacial metaphors[Jackendoff 1983]. The other strength illustrated by the example is that LCS captures the semantics of motion using a conveniently small set of primitives: the meaning of *climbed* is cast in terms of GO and UP[7]. By comparison, the traditional Logical Form representation requires a separate semantics for each motion verb[Jackendoff 1983]. Schank's Conceptual Dependency Theory[Schank 1984] contains many primitives, such as MOVE (in the sense of moving a body part), INGEST, ATRANS (meaning abstract transfer), PTRANS (meaning physical transfer), PROPEL, EXPEL, and GRASP, which can all be reduced to compositions of LCS primitives, even

7 The UP primitive is used here and in [Jackendoff 1983] for clarity; it may be decomposed further into primitives that literally mean "toward the top of."

if some of the specificity of Schank's primitives might be lost in the translation. The compromise of specificity in favor of concision in LCS will be worthwhile when designing the Gauntlet System's vision components, because the complexity of extracting descriptions from the visual stream excludes any representations with motion semantics that cannot be decomposed into sequences of easily-recognizable actions.

All descriptions in LCS comprise nested frames. At the highest hierarchical level are EVENTs and STATEs. EVENTs typically describe the relations expressed by action verbs such *as climb, force*, and *allow,* whereas STATEs typically describe relations expressed by verbs that enumerate an object's position, orientation, or state of being, such as *point,* and *be*. At the next level are PATHS, which express logical routes along which either orientation or action may happen. PATHS are composed of PLACES, which identify a position relative to a reference object. The simplest items of LCS are THINGS, which identify objects. The following is an abbreviated list of the productions sanctioned by LCS[Jackendoff 1983]:

$$[\text{EVENT}] \rightarrow \begin{cases} \left[_{\text{Event}}\ \text{GO}\left(\left[_{\text{Thing}} x\right], \left[_{\text{Path}} y\right]\right)\right] \\ \left[_{\text{Event}}\ \text{STAY}\left(\left[_{\text{Thing}} x\right], \left[_{\text{Place}} y\right]\right)\right] \\ \left[_{\text{Event}}\ \text{CAUSE}\left(\left[_{\begin{Bmatrix}\text{Thing}\\\text{Event}\end{Bmatrix}} x\right], \left[_{\text{Event}} y\right]\right)\right] \\ \left[_{\text{Event}}\ \text{LET}\left(\left[_{\begin{Bmatrix}\text{Thing}\\\text{Event}\end{Bmatrix}} x\right], \left[_{\text{Event}} y\right]\right)\right] \\ \quad\ldots \end{cases}$$

Here, GO accounts for action-verb constructions, STAY accounts for constructions such as “The books remained on the shelf,” CAUSE for those such as “John made us laugh” and LET for those such as “Mary allowed John to leave.”

$$[\text{STATE}] \rightarrow \begin{cases} \left[_{\text{State}}\ \text{BE}\left(\left[_{\text{Thing}} x\right], \left[_{\text{Place}} y\right]\right)\right] \\ \left[_{\text{State}}\ \text{ORIENT}\left(\left[_{\text{Thing}} x\right], \left[_{\text{Path}} y\right]\right)\right] \\ \left[_{\text{State}}\ \text{GO}_{\text{Ext}}\left(\left[_{\text{Thing}} x\right], \left[_{\text{Path}} y\right]\right)\right] \end{cases}$$

BE accounts for “The book was on the table.” ORIENT refers to pointing objects, for example “the sign pointed toward the center of town.” GO_{Ext} refers specifically to trajectory-like constructions in which the verb describes extent, instead of motion: “The road went from Boston to New York.”

$$[\mathrm{PATH}] \rightarrow \left[\left\{ \begin{array}{l} \mathrm{TO} \\ \mathrm{FROM} \\ \mathrm{TOWARD} \\ \mathrm{AWAY\text{-}FROM} \\ \mathrm{VIA} \end{array} \right\} \left(\left\{ \begin{array}{l} [_{\mathrm{Thing}}\, x] \\ [_{\mathrm{Place}}\, x] \end{array} \right\} \right) \right]$$

Paths are classified as *bounded paths, directions*, or *routes.* Bounded paths include those for which the source, goal, or both is specified, such as in the phrase “to the house.” The reference object in directions serves only to orient the path, and does not lie on the path directly, as in the phrase “toward the house.” Routes contain the specified reference object, but that object is not the source or goal of the path, such as in “via the house.”

$$[\mathrm{PLACE}] \rightarrow \left[_{\mathrm{Place}}\ \mathrm{PLACE\_FUNCTION}\left([_{\mathrm{Thing}}\ x]\right)\right]$$

$$[\mathrm{PLACE}] \rightarrow \left[_{\mathrm{Place}}\ \mathrm{ON}\left([_{\mathrm{Path}}\ x]\right)\right]$$

$$[\mathrm{THING}] \rightarrow [_{\mathrm{Thing}}\ x]$$

The PLACE-FUNCTIONs used in the Gauntlet System thus far are BEHIND, ON, ABOVE, UNDER, AT, NEAR, LEFTOF, RIGHTOF, and IN.

4.2.2 Transition Space

In his work on Causal Reconstruction[Borchardt 1993], Gary Borchardt observed that changes in objects and attributes, rather than the objects or attributes themselves, are the key to discovering the causal relations that answer how-and-why questions about events. To see why state transitions are so important, consider the following sequence of events: The hammer began to move toward the glass, then the hammer made contact with the glass. The glass was broken.

In the context of the example, the objects are the hammer and the glass. The attributes are variables such as the state of the glass (broken or unbroken), the motion of the hammer, the distance between the hammer and the glass, and the contact of the hammer with the glass. When faced with the question of why the glass broke, though, it is difficult to answer succinctly in terms of these attributes alone. The glass did not break simply because the hammer moved toward it, because had the hammer never touched the glass during the course of its motion, there would have been no collision. The glass did not break solely because contact occurred either, because had the hammer simply come to rest on the glass there might have been no damage. The deduction that answers the question is, roughly stated, that because the hammer's motion existed at the time contact occurred, the resulting impact caused the glass to break. In order to make explicit the constraints that enable such deductions, Borchardt introduced the Transition Space representation of events.

Transition Space is best visualized with a table in which each row corresponds to a relation between particular objects and attributes, and each column represents a particular time interval. Each cell of the table may have one of ten values: APPEAR, DISAPPEAR, CHANGE, INCREASE, DECREASE, and the corresponding opposites, NOT_APPEAR, NOT-DISAPPEAR, etc. Figure 4.2 is a simple Transition-Space description of the example.

$\cancel{\Delta}$	$-$	D	*distance*(glass,hammer)
A	$\cancel{D}$	$\cancel{D}$	*speed-of*(hammer)
A	$\cancel{D}$	$\cancel{D}$	*heading-of*(hammer)
$\cancel{A}$	$\cancel{A}$	A	*state-is*(glass,broken)

t_0 t_1 t_2 t_3

Key

A	APPEAR	$\cancel{A}$	NOT-APPEAR
D	DISAPPEAR	$\cancel{D}$	NOT-DISAPPEAR
Δ	CHANGE	$\cancel{\Delta}$	NOT-CHANGE
$+$	INCREASE	$\cancel{+}$	NOT-INCREASE
$-$	DECREASE	$\cancel{-}$	NOT-DECREASE

Figure 4.2: A simplified Transition-Space rendering of an event.

In order to frame a useful description in terms of Transition Space, there must be a way to ascertain the relevant state changes and temporal relationships. The notion of relevance in this context would seem circuitous were it not for that Borchardt required all input to his system to adhere to Grice's maxims of quantity, quality, relation and manner[Borchardt 1993]. In effect, the requirement ensured that all state changes that were present were relevant, and that no relevant state change was absent from the description. Enforcing such a requirement does compromise the flexibility of Borchardt's system, but arguably it does not detract from the utility of Transition Space as a representation.

Although the process of determining relevance becomes much more complicated when Grice's maxims take the form of suggestions rather than guarantees, it may be that we learn to identify relevant features by experience, or that we take cues about relevance from our sub-symbolic reasoning abilities. In the context of the Gauntlet System, this would mean that either the expert-specific memory, the cross-representational memory, or both would need to participate in the process of determining relevance. Until these aspects of the Gauntlet System's design are explored further, it is acceptable to fall back on Grice's maxims as a way to evaluate the clarity of language input to the Gauntlet System.

4.2.3 Thread Memory

As described by Lucia Vaina and Richard Greenblatt[Vaina, Greenblatt 1993], the Thread Memory representation organizes class hierarchies in a convenient and flexible way. Lance benefits from Thread Memories not only to capture categorical information expressed in sentences, but also to facilitate internal processes that enable Lance to learn to recognize representational frameworks.

A thread is a loop-free chain linking semantic nodes, accessible via a key. Once a thread memory forms, its key evokes the memory, by linking to the first element of the thread. An encounter with a particular mallard duck, for example, generates the following:

mallard → living-thing → animal → bird → duck → mallard

The key, *mallard,* though not an element of the thread, triggers the arrangement of semantic nodes ending in the mallard category. The links point from general nodes toward more specific nodes, starting with the first node following the thread's key.

Traditionally, categorical information is represented by a tree. Trees capture categorical information efficiently in simple cases, and thus may seem superior at first glance to threads, which may appear unnecessarily redundant. Barring closer consideration, threads seem wasteful—why, after all, should an encounter with a new mallard duck not be described by precisely the same memory as that for the mallard in the example, if the new bird looks, walks, and quacks the same way?

The power of Thread Memories rests in their flexibility and robustness. As the following points explain, the benefits outweigh the costs of using threads instead of trees. The actual cost incurred by thread representations as opposed to trees in complex systems is minimal.

- Retrieval of classes from threads occurs in an order conducive to robust performance in the presence of errors: even if the key a for a different type of duck were mistakenly associated with the thread for *mallard*, retrieval of semantic nodes would continue to provide useful information until the last node of the thread. Trees constrain information retrieval to proceed from specific to general categories, and thus do not have this advantage.
- Anomalies are handled efficiently in Thread Memories, whereas they are cumbersome to represent with trees. In adding a very peculiar duck to the memory that exhibits properties of both plants and animals, for instance, the Thread Memory representation does not have to evaluate the effects of adding the plant category on other memories of ducks. Any such modification of a tree would affect all subordinate categories.
- Threads easily permit objects to participate in more than one classification hierarchy at once. In a certain context, a dog is a canine, which is a mammal, which is an animal, etc. In a different context, however, the same memory of a dog should evoke the notion of a pet, and its associated classification hierarchy. Thread Memory makes this type of flexibility simple to achieve, by linking multiple threads to a single key.

That threads possess this added flexibility and robustness as compared to trees distinguishes them as a superior representation for the Gauntlet System, in which efficiency is not of highest priority. The redundancy of Thread Memory does not preclude their use in efficient systems, though, in particular because short, bushy trees offer little space savings over separate threads for each leaf.

In the meta-framework employed to implement representations in Lance, threads may augment the ontology of any representation. For example, LCS has the basic symbol GO rather that a complicated description of motion. Suppose a certain expert in the Gauntlet System were adept at detecting whether a person in a visual scene was running or walking—then it might seem reasonable to expand Jackendoff's LCS to include these new categories of motion. As noted in Section 4.2.1, however, the simplicity of LCS greatly benefits the Gauntlet System, and adding more basic motion types such as RUN would have complications, because the changes in the ontology would propagate their effects throughout the system. Thread Memory neatly addresses the problem, however, by allowing the GO primitive to expand into threads such as GO→WALK and GO→RUN. In this way, no generality is lost and designers of Gauntlet's experts may decide on a per-expert basis whether to depend on the augmented ontology.

5. Discussion

The methods that I implemented for learning how to model correspondences between language and semantic descriptions have some important limitations. Some of these limitations suggest straightforward improvements to the design, while others pose exciting challenges that motivate future inquiry.

Efficiency presents a problem for Lance. I am not aware that there is any aspect of my approach that is inherently costly from a computational perspective, but there are several aspects of my implementation in which I favored simplicity rather than computational efficiency. In particular, Lance incurs combinatorial explosion when processing training examples that have many lexical bundles that each bear one thread found in a complex semantic form. An example of this phenomenon is compound noun phrases. Consider an example that is plausible in the Wall Street Journal corpus, "the stock volume analysis." WordNet contains at least 16 distinct threads pertaining to *stock,* at least six pertaining to *analysis* and at least five pertaining to *volume*[Miller, G. 2006]. Lance must therefore try up to 480 combinations of threads, even though most of the combinations are not sensible. Several AI techniques apply to this problem; most notably constraint propagation. In the noun-phrase example, it would be unlikely that the interpretation of *stock* as meaning *honor* and *volume* as *quantity* even merits consideration. Constraint propagation is an apt choice for addressing this problem because it permits elimination of unlikely or impossible combinations by virtue of constraints that could be shared by pairs of lexical items in phrases such as "the stock volume analysis."

Another problem area that might benefit from application of AI techniques is approximation of high-order constraints. In order to approximate high-order constraints via heuristic methods, my implementation stores a collection of examples along with each first-order model. Together, the first-order model, which is built using the Arch-Learning methodology, along with the collection of examples, form an augmented model that can be thought of as a two-stage filter. The first stage eliminates all descriptions that are not admitted by the first-order model, and the second stage ranks descriptions according to their degree of familiarity so that the less likely descriptions can be eliminated. Clearly, as the number of training examples increases, a simple average-distance computation may become both ineffective and inefficient, because the number of stored training examples will grow without bound and there may be significant variation within a single augmented model's domain.

Addressing the limitations of my rudimentary second-stage filtering could be fertile ground for application of machine-learning techniques. I have not tried to bring advanced clustering techniques or other classification machinery to bear on this problem but I believe such an attempt would be informative. A technique that seems especially promising in its capacity to draw out the underlying classes from often-inscrutable distance scores is the use of Self-Organizing Maps (SOMs)[Kohonen 2001], which not only provide a way to separate prohibitively high-dimensional data into categories in an unsupervised manner, but also lend themselves to convenient means of visualizing the categories in a low-dimensional view. Such visualization opportunities align well with my goal of creating a transparent training interface, that elucidates the constraints so that a human trainer can revise assumptions about the problem domain. SOMs have the additional feature that they may limit the number of stored examples, so that the space required for learned models can have an enforced bound without substantive loss of generality.

As noted, my methods for obtaining a distance-like score from a pair of descriptions are ad-hoc. Additionally, because it would violate abstractions to include any particulars of the representational frameworks in the distance calculation, the methods are also woefully ill-suited to any particular representation. I believe that it would be an interesting challenge to design a scoring method or other classification scheme that evolves along with the model to which it pertains.

Undoubtedly there are many schemes, of varying complexity, that could be applied to the task of filtering the output of the first-order models. Although trying one or several of such schemes might lead to new insights, the holy grail is a new kind of model that can acquire all of the constraints of the model explicitly. One approach might be to incorporate information from the dependency parse into the learning method. As discussed in Section 3, it is difficult to learn constraints that span disparate parts of the constituency parse tree because the system would be forced to postulate interdependence among seemingly disconnected variables. General methods for discovering such interdependence are difficult to derive, and likely infeasible.

In a dependency parse, many of the nodes that are separated by several links in constituency trees are brought closer together. The subject-verb-object relationship, for example is made explicit. By augmenting the constituency tree with dependency links, it may become possible to learn many new rules governing semantic interpretation as first-order constraints, via similar methods to those employed in this implementation. If, for example the subject-verb dependency link were augmented by one or more if-then constraints, in which the antecedent and consequent were constrained bundles of the type already learned by my system, it would be possible in principle to learn constraints such as *if the THING represented by noun phrase is a physical object, then the verb must be a CHANGE verb and not an INCREASE or DECREASE verb in order for the model to admit a TRANSITION-SPACE description.*

Yet another way capture constraints that the current models can only approximate is to augment the semantic descriptions themselves so that each representation's dependencies on lexical and structural information can be expressed entirely in terms of first-order constraints. This approach has the advantage that it does not require augmentation of the present constituency-parse model. For example, TRANSITION-SPACES could subdivide into two categories, PHYSICAL and ABSTRACT. The model of TRANSITION-SPACE→ABSTRACT would admit only abstract THINGS as the subject of the transition and would admit any type of transition verb. The model of TRANSITION-SPACE→PHYSICAL would admit only physical objects, and would place restrictions on the transition verb such that it cannot mean *increase* or *decrease.* Neither of the two models suffers from the overgeneralized constraints that plague the present model of TRANSITION-SPACE. Factoring models with high-order constraints into two or more models that the present implementation can learn would improve coverage.

In addition to the issue of improving the coverage of the constraints, I noted that context-sensitivity, in the form of some kind of feedback directed from Gauntlet's expert components to Lance, will be essential for successful concept formation. Currently, Lance has minimal support for feedback in the form of a priming mechanism that allows anticipated partial descriptions to be supplied so that the next sentence interpreted, or any of the phrases it comprises, will be assigned these supplied forms preferentially. A simple improvement would allow the external components to prime descriptions that are anticipated to be *similar* to upcoming matches, rather than strictly equivalent. Another improvement involves prepositional-phrase attachments. The present implementation tries attachment permutations until one happens to have a representation at the highest level, or until all possibilities are exhausted. The implementation should be improved to choose the prepositional-phrase attachment scheme that permits one of the primed descriptions to be assigned. Finally, it may be possible to allow re-tagging and re-parsing of sentence fragments to accommodate primed descriptions. If the recognition process were be modified so that it could be executed in the inverse direction, such that parse trees could be generated from partial semantic descriptions, then primed descriptions could be rendered as sentence fragments and the tags from these sentence fragments could be applied to the input fragment, which could then be re-parsed. Such behavior could serve to bootstrap the imagination-stimulating loops sought in the Gauntlet System, because a remembered semantic description could cause an incoming sentence to be re-parsed so that it is recognized as having a similar description.

6. Contributions

I applied the powerful techniques of the Arch-Learning paradigm[Winston 1970] to the domain of semantic interpretation of language. Specifically, I implemented a program, Lance, that learns how to generate semantic descriptions of sentences and fragments by processing a training sequence consisting of sentences and fragments paired with their semantic descriptions. The training sequence must emphasize, through its carefully chosen examples, the important structural and lexical features of phrases that determine how the phrases project in the semantic domain. Lance benefits from the observation that examples of one semantic category often constitute counterexamples of other categories corresponding to similar linguistic forms. Accordingly, Lance can generate counterexamples from examples in the training set, thus taking full advantage of both the generalization and specialization modes of the Arch-Learning paradigm.

Lance learns by taking a single description-phrase pair as the initial model of the correspondence between a particular syntax form and its corresponding semantic description. Lance then generalizes or specializes this model according to observed regularities and constraints in its training sequence that pertain to the model. Generalization occurs in response to examples, and takes the form of shortening Thread Memories [Viana and Greenblatt 1979] pertaining to lexical or semantic items. Specialization occurs in response to near misses and takes the form of adding symbols or explicit constraints to the Thread Memories. The explicit constraints employed by Lance are MUST-BE-A, MUST-NOT-BE-A, MUST-BE-IN-SET, MUST-HAVE-FEATURE, MUST-NOT-HAVE-FEATURE, and MUST-HAVE-FEATURE-IN-SET. These explicit constraints decorate the Thread Memories of the initial description in order to transform that description into a model.

Thus, in this thesis and its supporting work, I have developed an approach to learning how to translate phrases and sentences to meanings. The Arch-Learning paradigm provides the foundation for my approach, in which presentation of parse trees paired with instantiated representations enables learning by examples and near misses. From the perspective of the numbers, I have:

- Implemented Lance, a 12,000 line Java program.
- Demonstrated that Lance can learn models for recognizing THINGS PARTS, PLACES, PATHELEMENTS, TRAJECTORY-SPACES, TRANSITION-SPACES, CAUSES, and IS-A relations. From a training sequence comprising 95 examples, Lance has learned 27 models that specify the mapping between phrases and different manifestation of each of these semantic categories.

References

Bonawitz, Keith. 2003. Bidirectional Natural Language Parsing Using Streams and Counterstreams. M. Eng., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Borchardt, Gary. 1994. *Thinking between the Lines*, *Artificial Intelligence series*. Cambridge MA: MIT Press.

Fellbaum, Christiane. 1998. *WordNet an Electronic Lexical Database*. 19 vols. Vol. 17, *Language Speech and Communication series*. Cambridge, MA: MIT Press.

Finlayson, Mark. 2007. *Java Wordnet Interface*. MIT 2007 [cited 2007]. Available from http://www.mit.edu/~markaf/projects/wordnet/.

Jackendoff, Ray. 1983. *Semantics and Cognition*. Vol. 8, *Current Studies in Linguistics*. Cambridge, Massachusetts: MIT Press.

Kohonen, Teuvo. 2001. *Self-Organizing Maps*. Third ed. Vol. 30, *Springer Series in Information Science*: Springer.

Larson, Stephen. 2003. Intrinsic Representation: Bootstrapping Symbols from Experience. M. Eng., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz. 1994. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics* 19 (22):313-330.

Marr, David. 1976. Artificial Intelligence -- a personal view. Cambridge, MA: MIT A. I. Lab. Memo 355.

Miller, Catherine. 2006. Modeling Trust in Human Conversation. M. Eng., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Miller, George A. 2007. *WordNet 3.0* [Website]. Princeton University 2006 [cited 10/1/2007 2007]. Available from http://wordnet.princeton.edu/perl/webwn.

Niyogi, Sourabh. 2005. Steps Toward Deep Lexical Acquisition. Paper read at Workshop on Psychocomputational Models of Human Language Acquisition, at Ann Arbor, Michigan.

Seifter, Mark. 2007. Building Representations from Natural Language, Dept. of Electrical

Engineering and Computer Science, MIT, Cambridge, MA.

Vaina, Lucia and Greenblatt, Richard. 1979. The Use of Thread Memory in Amnesic Aphasia and Concept Learning. Cambridge, MA: MIT A. I. Lab. Working Paper 195.

Winston, Patrick. 1970. Learning Structural Descriptions from Examples. Ph.D., Electrical Engineering Dept., MIT, Cambridge, MA.

———. 2007. *System Building Using the Wire Paradigm* [Website] 2003 [cited 2007]. Available from http://groups.csail.mit.edu/genesis/wire.html.

———. 2007. Understanding Concepts: An Essential Aspect of Robust Intelligence. Cambridge, MA.

———. 2007. Biologically Inspired Artificial Intelligence. Cambridge, MA.

Winston, Patrick et al. 2007. *Four Powerful Pieces* [Website] 2003 [cited 2007]. Available from http://groups.csail.mit.edu/genesis/frames.html.