Bidirectional Natural Language Parsing using Streams and Counterstreams

by

Keith A. Bonawitz

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

May 21, 2003

Author . . . .
Department of Electrical Engineering and Computer Science
May 21, 2003

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

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

Bidirectional Natural Language Parsing using Streams and Counterstreams

by

Keith A. Bonawitz

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis investigates the bidirectional exchange of information between linguistic and non-linguistic semantic inputs containing ambiguities. Such exchange is critical to Cognitively Complete Systems, in which collections of related representations and processes cooperate for their mutual problem-solving benefit. The exchange paradigm of *reconciliation* is defined, in which ambiguities and gaps in multiple input domains are simultaneously resolved. A complete architecture for implementing reconciliation between the linguistic and non-linguistic semantic domains is described, employing the Streams and Counterstreams bidirectional search technique, Combinatory Categorial Grammar, and Lexical Conceptual Semantics. This architecture has been implemented, resulting in a system that can act as an ambiguity resolving constraint between linguistic and non-linguistic inputs, with the side effect of also producing a language parsing and generating system. For example, if the system is presented with the phrase "The orange rolls," and a non-linguistic input (perhaps from a vision subsystem) describing the presence of bread (rolls) in the scene, the system will automatically select the interpretation of the the linguistic input describing pieces of bread of orange color. Thus, ambiguity is resolved in the linguistic domain, and the statement that the rolls have the property *orange* can be propagated to the non-linguistic semantic domain.

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

Acknowledgments

I thank Liz Baraff, Jake Beal, Jimmy Lin, and Greg Marton for exposing me to much of the research which serves as the foundation for this Thesis. Special thanks to my advisor, Patrick Winston, for giving me both the support and the opportunity to go in my own direction with this research.

This research is supported by the National Science Foundation, Award Number IIS-0218861. Portions of this document are published in the proceedings of the North American Chapter of the Association for Computational Linguistics 2003 Human Language Technology Conference (HLT-NAACL 2003, In Press) under the title “An Architecture for Word Learning using Bidirectional Multimodal Structural Alignment” [2].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis focuses on the cooperative exchange of information between the linguistic and non-linguistic semantic domains. The motivation behind this thesis stems in part from situations in which standard natural language parsers and generators seem to come up short: specifically, situations in which the correct interpretation of either the linguistic or non-linguistic semantic input is only achievable by considering both inputs simultaneously. This thesis is also supported by philosophical motivations calling for the creation of systems with broader and more robust cognitive abilities. Such systems rely on a high degree of cooperation and sharing between problem domains, and so the thesis embraces the philosophy by demonstrating a technique for cooperative information exchange in general, and by creating a system for cooperative exchange between the linguistic and non-linguistic semantic domains.

1.1 Motivation by Example

As an motivating example, let us consider a standard parser being presented with a few words: “The orange rolls.” Because of the ambiguity inherent in natural language, these words can be interpreted as having at least two meanings, including a piece of fruit moving by rotation, or a piece of bread with a red-yellow hue. With only the linguistic input to go on, a traditional parser could produce both interpretations. However, linguistic inputs are often so ambiguous that a parser cannot afford to produce all possible interpretations. The typical solution is to use some statistical heuristic to ignore unlikely interpretations. In this case, the parse might decide that the red-yellow-bread interpretation is statistically unlikely and return only the rotating-fruit interpretation.

Now let us also consider a similar scenario, but with the addition of visual input from a camera observing a few pieces of bread. For the sake of argument, we will assume that the visual recognition system is able to identify the bread in the input, but is unable to determine the color of the bread (perhaps there is low light, or the input is black and white). Now, if a human were serving as the parser, they would easily be able to identify the red-yellow bread interpretation of "the orange rolls" as the most likely, given the non-linguistic context. In fact, if that human had already familiarized themselves with the visual input before receiving the linguistic input, it is very possible that the rotating fruit interpretation would not even occur to them, because the red-yellow bread interpretation seems so obvious!

Returning to the artificial parsing system, we are likely to be disappointed. Standard parsers do not consider non-linguistic context when evaluating linguistic inputs, so the parser will behave exactly as if the non-linguistic input was not present. At best, it might be hoped that the parser would generate both the rotating-fruit and red-yellow-bread interpretations, and that some other system could be charged with ranking these interpretations using the non-linguistic context. But what happens when there are too many interpretations for the parser to enumerate? The parse may choose to prune a statistically unlikely interpretation before passing its results to the non-linguistic context ranking system, even though the non-linguistic context might overwhelmingly support that interpretation.

This thesis presents a solution to this example problem. By incorporating non-linguistic context directly into the natural language parsing process, and analogously by incorporating linguistic context into natural language generation, this system seeks to avoid these early-pruning problems. Furthermore, the system hopes to go one step further by passing on information about any ambiguities it resolves to the domains that gave rise to the ambiguity. In this example, each word in the linguistic domain can be assigned an unambiguous interpretation. Furthermore, the information that the bread is orange-colored can be passed on to the non-linguistic domain. Any other system that use the non-linguistic domain can then make use of the fact that the rolls are orange, just as if the vision system had actually been able to observe it.

1.2 Philosophical Motivation

1.2.1 The plight of AI research

Traditionally, researchers have applied a divide-and-conquer approach to the problem of Artificial Intelligence. For the past several decades, we have known that creating an artificial system capable of human-like intelligence is an exceptionally difficult problem that will require many subsystems specializing in the solution of a wide array of problems. The general approach, then, has been to take the overall problem (e.g. artificial intelligence) and break it down into smaller problem (e.g. vision, audio, natural language processing, kinematics, abstract reasoning, etc). These problem areas are recursively subdivided until thousands of micro-problems are being pursued in parallel. The standard view seems to be that once all of these sub-sub-sub-problems are perfectly solved, then they will be quickly lashed together and AI will be solved.

Unfortunately, these traditional approaches make Artificial Intelligence too hard. So far, I have simply depicted the standard technique of divide-and-conquer, and there is nothing inherently wrong with method of attack in many situations. The reason divide-and-conquer falls apart in this case is that subdividing the problem space does not necessarily result in simpler problems.

There is much reason to believe that the human brain shares information rampantly. Consider the example of the phoneme-restoration effect [17] where subjects hear a sentence, but with one of its phonemes completely removed. These subjects spontaneously hallucinate into the gap the phoneme that gives the sentence the most coherent meaning, and astonishingly are completely unaware that the sentence even contained a gap at all! Studies in which information sharing is abnormal give further clues into the nature of information sharing. Consider the effect of synesthesia, in which some people experience cross-over between various senses, literally seeing numbers in color, hearing notes in color, or tasting words. Experimental evidence [11] has recently been produced which demonstrates that synesthetes truly perceive differently, easily completing experimental tasks that are difficult or impossible for non-synesthetes. These experiments force us to consider how similar information sharing may be occurring all the time in everyone's brain, but is simply taken for granted.

In contrast to this paradigm of shared information, artificial intelligence researchers have attempted to solve each subproblem in isolation, without using any information from across the modular divide. Many systems continue to be created that ignore the context in which they are being used. In such a systems, understanding a scene containing a dog together with the label "boxer" would require either an object recognition system talented enough to identify different breeds of dog, or a language system somehow capable of guessing that "boxer" in this case did not refer to a professional fighter. Many intelligence problems, including visual scene reconstruction, object recognition, and speech recognition, suffer from under-constrained inputs. Creating systems which solve these problems becomes more tractable if they can work together with other systems, using context to reduce the range of possibilities.

1.2.2 Cognitively Complete Systems

In order to start focusing on cooperative systems, the notion of *Cognitively Complete Systems* has arisen. Research in this area investigates how modules and representations with different but related purposes can cooperate with one another for their collective benefit. In a Cognitively Complete System, it is no longer necessary for a particular representation or cognitive module to completely solve the problem assigned to it. Rather, it should make whatever progress can be made with the information available. The solution is later completed by cooperating with related cognitive facilities. The hope of Cognitively Complete Systems is that by relaxing the expectations from perfection in isolation to most-informed-guesses with help from neighbors, we will be able to make much progress in research areas that have traditionally been considered difficult.

This thesis has been executed as part of the MIT Artificial Intelligence Lab's Bridge Project. The Bridge Project has the goal of creating a Cognitively Complete System spanning the vision and language faculties. As its core structure, Bridge has a variety of representations, some closely tied to language, some closer to vision, and other more abstract representations in between. Tying these representations together is a web of bidirectional constraints. These constraints are charged with the task of maintaining coherence between the representations while providing avenues for information transfer. The complete Bridge system will be able to accomplish tasks such as imagining input sentences as "mental videos," then reflectively analyzing these videos to extend the understanding of the input. As an example, the sentence "The boy walked from the tree to the table in the house," could elicit a video, from which the question "Did the boy walk through a door?" might be answered. The complete Bridge system should also be able to visually observe events and express these in language. The symbolic structures required for language can then provide a more thorough understanding of what has been seen. Because the system supports bidirectional information exchange, it should be able to use a visual input together with a linguistic input as an opportunity for learning how the two inputs are related.

As a Cognitively Complete System, the Bridge project is interesting in and of itself. However, the interest should also extend to other fields of research related to artificial intelligence, such as vision, natural language processing, and cognitive science. In these fields, much multi-modal research has been proposed but remains virtually untested due to the lack of an adequate test bed. For example, natural language theories of word learning [10, 15] and language acquisition models [19, 7] could be testable in the context of Bridge. Thus, it is the further goal of the Bridge Project to support advancement in these domains by providing a Cognitively Complete System as a test bed for exploring new theories, especially those involving the notion of cooperative representations.

1.3 Thesis scope

As exploration in handling situations such as the "orange rolls" motivating example, and as one step in the creation of the Bridge System, I present here an investigation of information exchange between natural language inputs and a structured semantic representation of the worlds. This thesis seeks to answer two questions:

- What kind of exchange should we expect between natural language and non-linguistic semantics?
- How can this exchange be accomplished?

In answer to the first question, the notion of *reconciliation* is introduced. Under the reconciliation paradigm, both linguistic and non-linguistic input are presented to the system, both containing ambiguities and missing information. A reconciliatory exchange seeks to makes the best use of all the information available in both input domains to try to resolve ambiguities and fill gaps in both the linguistic and non-linguistic inputs.

With the concept of reconciliation laid out, I then describe a complete architecture for implementing reconciliation between the linguistic and non-linguistic semantic domains, employing the Streams and Counterstreams bidirectional search technique, Combinatory Categorial Grammar, and Lexical Conceptual Semantics. As part of the research leading to this thesis, I have also implemented the system described. Thus, this thesis stands not only as a proposed architecture for reconciliation, but also as documentation for a system I have successfully constructed.

Chapter 2

Solution Roadmap

In this chapter, I examine the problem domain of information exchange between linguistic and non-linguistic semantic domains. I then provide an outline of a system which will support bidirectional reconciliatory exchange between these domains. This outline will be realized in Chapter 4, with the support of techniques described in Chapter 3.

2.1 The Problem Domain

This section will briefly discuss characteristics of the linguistic input domain and the non-linguistic semantic domain, and will consider how information can be exchanged between them.

2.1.1 Characteristics of the linguistic input domain

Inputs in the linguistic domain are sentences and phrases consisting of unstructured sequences of words. These inputs can arise in several ways, including audio processing (e.g. speech recognition), visual processing (e.g. reading/text recognition), or in the case of an artificial system, directly from a keyboard or other text source.

When linguistic inputs enter the system, they are linear rather than structured (all that is known is that one word follows another). It is the responsibility of the system, with the assistance of a grammar, to extract a structured linguistic representation from the original unstructured representation. It is this structured linguistic representation that implies a semantic meaning.

2.1.2 Characteristics of the semantic input domain

The semantic domain is a structured representations of the world. Information can enter the semantic domain through a variety of non-linguistic channels: vision, kinesthetics, reasoning, imagination, and so on. Semantic information can also be extracted from linguistic inputs, for example by parsing.

The nature of language and grammar imposes restrictions on the structure of semantic representations that can be directly interchanged with language. The semantic principle of compositionality states that the meaning of any expression (such as a phrase) is a function of the meaning of its sub-expressions, where the particular function is determined by the method of composition. For example, the expression "The quick fox jumped over the log" can be considered a composition of the sub-expressions "The quick fox" and "over the log", where syntactic composition with "jumped" is the method of composition. In other words, the semantics of this sentence can be expressed: *jumped*(SEMANTICS("The quick fox"), SEMANTICS("over the log")). Recursive application of this principle reveals that the semantic value of an expression is a structured representation.

2.1.3 Parsing, Generation, and Reconciliation: Paradigms for connecting the linguistic and nonlinguistic domains

This thesis focuses on information exchange between related domains. Therefore, let us first examine in what manners information could be exchanged.

The first exchange paradigm is *parsing*. The parsing paradigm is a unidirectional exchange process in which the unstructured linguistic input is translated into semantic structures. Many linguistic inputs contain sufficient ambiguity that they can produce several semantic interpretations. This ambiguity can be a result of the acquisition

Parsing:

$$\text{“The red ball”} \qquad \overset{parsing}{\longrightarrow} \qquad \left[\begin{array}{l} \text{MODIFIER} \ \left[<\!\textit{Property}\ \text{RED}\!>\right. \\ <\!\textit{Thing}\ \text{BALL}\!> \end{array}\right.$$

Generation:

$$\left[\begin{array}{l} \text{MODIFIER} \ \left[<\!\textit{Property}\ \text{RED}\!>\right. \\ <\!\textit{Thing}\ \text{BALL}\!> \end{array}\right. \qquad \overset{generation}{\longrightarrow} \qquad \text{“The red ball”}$$

Reconciliation:

$$\left[\begin{array}{l} \left[<\!\textit{Thing}\ \text{ROLLS:BREAD}\!>\right. \\ \text{“The} \ \left\{\begin{array}{c} orange : color \\ orange : frult \end{array}\right\} \ \left\{\begin{array}{c} rolls : verb \\ rolls : bread \end{array}\right\} \text{”} \end{array}\right]$$

$$\overset{reconciliation}{\longrightarrow} \quad \left[\begin{array}{l} \left[\begin{array}{l} \text{MODIFIER} \ \left[<\!\textit{Property}\ \text{ORANGE}\!>\right. \\ <\!\textit{Thing}\ \text{ROLLS:BREAD}\!> \end{array}\right. \\ \qquad \text{“The orange:color rolls:bread”} \end{array}\right]$$

Figure 2-1: A schematic representation of the parsing, generation, and reconciliation paradigms for information exchange between the linguistic and non-linguistic domains. In the parsing paradigm, information flows from the linguistic domain to the non-linguistic domain. In the generation paradigm, information flows from the non-linguistic domain to the linguistic domain. In the reconciliation domain, information flows in both directions, resolving ambiguities and filling gaps in both domains.

of the linguistic input; for example, if the linguistic input is acquired from an audio recording of speech, there may be uncertainties as to what word was actually said. Ambiguity can also be implicit in the rules of grammar; for example, the phrase “the duck on the table with the flag” leaves the parser to guess whether “with the flag” implies that the duck has a flag, or that the table has a flag. In the parsing paradigm, the only recourse for dealing with this ambiguity is to produce all possible semantic interpretations. Because this is computationally expensive, most parsers actually use some statistical heuristic to guess which are the most likely interpretations, and ignore the rest. The parsing paradigm is the most commonly explored in artificial intelligence.

The second exchange paradigm is *generation.* Generation is essentially the reverse process from parsing. It is the process of taking a non-linguistic semantic input and translating it into a new linguistic representation, preserving as much meaning as possible.

The final exchange paradigm is *reconciliation.* Reconciliation is somewhat like parsing and generating simultaneously. In this paradigm, both linguistic and non-linguistic input are presented to the system, and it is assumed that there is a good chance the two inputs are related. Both of these inputs are expected to contain ambiguities and missing information; therefore it is the job of the reconciliation system to make the best use of all the information available at both inputs to try to resolve the ambiguities.

It is worth noting that these three paradigms can be generalized to settings that do not involve language. Cognitively Complete Systems which are rife with exchange channels connecting various representational domains, many of which are non-linguistics. In such situations, the paradigms can be given the generic names *forward translation*, *backward translation*, and *reconciliation*.

This thesis focuses on the reconciliation problem for several reasons. First, the reconciliation paradigm is critical for Cognitively Complete Systems to reach their full potential, because it enables representational cooperation. Furthermore, parsing and generation can be viewed as degenerate cases of reconciliation in which either the linguistic or non-linguistic side of the system is devoid of useful context. Thus, solutions to the reconciliation problem also provide for the parsing and generation problem. Finally, the reconciliation problem is the least explored of the paradigms, partly because it is the hardest, and partly because research environments often focus on a single input modality, where reconciliation cannot prove its worth.

2.2 Approaching the Reconciliation Problem

In the previous section, we observed that the linguistic input domain has an implicit semantic structure, while the non-linguistic semantic domain has an explicit structure. If we can make the semantic structure of the linguistic domain explicit, then intuitively we can approach the reconciliation task through structural alignment of semantic structures. The remainder of this chapter provides a very brief outline of a system architecture for reconciliation. As the rest of the thesis progresses, the outline presented here should serve the reader as a roadmap for how the various subsystems will fit together to achieve the larger goal of reconciliation.

2.2.1 Linguistic Processing: Semantics from Linguistic Input

Making explicit the semantics of a linguistic input requires two things: a grammatical framework and a search procedure.

It the function of a grammar to determine how words from the lexicon may combine together, as well as to determine and what the semantics of this combination will be. The grammar used in this thesis is Combinatory Categorial Grammar ([14], Section 3.2), a grammar based on functional representations of syntactic categories. This grammar uses Lambda Calculus expressions to handle semantics.

A search procedure is required to determine how the rules of grammar should be applied to a given linguistic input. Because this system focuses on bidirectional exchange of information, I have chosen a bidirectional search technique called Streams and Counterstreams ([16], Section 3.4). This technique is attractive because of its cognitive plausibility and its extensibility. However, it is difficult to find implementations of Streams and Counterstreams, therefore part of this research has been to formulate and implement a serial version of Streams and Counterstreams (Section 4.4). Finally, it is unclear whether this technique has been implemented outside of the visual processing domain, the domain in which it was suggested. Thus, another aspect of this research is the application of Streams and Counterstreams to linguistic processing using CCG (Section 4.5).

2.2.2 Structural Alignment and Bidirectional Exchange

Because we want information to flow bidirectionally, we cast the grammatical processing subsystem in terms of a constraint propagation network, with each constraint representing the application of a single grammatical rule (Section 4.3). In this way, linguistic and non-linguistic semantic inputs can be presented at opposite sides of the constraint network, and the network will minimize ambiguity. Gaps in the input domains are filled using structural alignment (Section 3.5), a process in which corresponding elements in two structured representations are identified by matching. Correspondence between non-matching elements is then implied by the structural constraints of the representations. For the present system, this structural alignment is actually implicit in the operation of the constraint network and the construction of the constraint network topology using bidirectional search.

2.2.3 Semantic Representation

In order to perform structural alignment, the representation for the semantic domain must have several key properties:

- The domain must be a structural representation and it must be symbolic, in order to allow alignment of symbols.
- For inferences made from structural alignment to be valid, the representation must obey the principle of compositionality.
- The representation should contain orthogonal elements (i.e. the same piece of semantics is not encoded into multiple symbols) so that there are canonical ways of expressing particular meanings.
- Finally, the semantic representation must be lexicalized, implying that the semantics of any linguistic phrase can be cleanly divided amongst the phrase's constituent words. Each word should get a single connected semantic structure that does not share semantic symbols with any other word.

With all of these properties in mind, we choose Lexical Conceptual Structures (LCS, [9], Section 3.1) as the basis for our semantics. Because our grammatical framework operates on Lambda-Calculus semantic functions, we will need to extend LCS with lambda calculus (Section 4.1). Due to reasons we will consider later, it will be preferable to using a variant of lambda calculus called Label-Selective Lambda Calculus (Section 3.3) for this extension. Supporting Label-Selective Lambda Calculus will also require an extension to CCG (Section 4.2).

Chapter 3

Foundational Techniques

In this chapter, I describe several techniques which serve as a foundation for the reconciliation system. Each topic is independent and presented in its own section. These foundational techniques will then be built into a complete system in Chapter 4. A complete explanation of any of these topics could easily fill this thesis. Therefore, only enough of each topic is presented here for the reader to understand the general technique an the role it plays in the present system. The techniques presented are:

- Section 3.1: *Lexical-Conceptual Semantics* (LCS), the semantic representation framework

- Section 3.2: *Combinatory Categorial Grammar* (CCG), the grammar framework for the parser

- Section 3.3: *Label-Selective Lambda Calculus* (LS λ-calculus), which will be used to extend CCG to support LCS.

- Section 3.4: *Streams and Counterstreams*, a priming-based bidirectional search technique

- Section 3.5: *Structural Alignment*, an method for transferring information across partially dissimilar structures.

3.1 Lexical Conceptual Structures

The system described in this thesis captures semantic information using Lexical-Conceptual Semantics (LCS), developed by Ray Jackendoff [9]. LCS is a formalized semantic representation focusing on spatial relations, and is tailored to represent how humans conceive of the world, particularly in the context of explaining the world with language. LCS is a recursive frame-based representation, where each frame has a primitive type and a set of parameters to be filled. To get a flavor for the LCS representation, consider the following structure representing "The boy threw the ball toward the lake:"[1]

$$\left[\begin{array}{l} \left[<\textit{Thing}\ \text{BOY}>\right. \\ \left[\begin{array}{l} \left[<\textit{Thing}\ \text{BALL}>\right. \\ \left[\begin{array}{l} \left[\begin{array}{l} \left[<\textit{Thing}\ \text{LAKE}>\right. \\ <\textit{Place}\ \text{AT}> \end{array}\right. \\ <\textit{Path}\ > \end{array}\right. \\ <\textit{Event}\ \text{GO}> \end{array}\right. \\ <\textit{Event}\ \text{CAUSE}> \end{array}\right.$$

LCS identifies several classes of frames. Most basic are THINGSs, which correspond to objects in the world. Building on this are PLACEs, which identify a location in space, typically using a THINGS as a reference. Next are PATHs, which describes connected progressions through space-time. Finally, there are STATEs and EVENTs, which describe how THINGSs interact with PATHs and PLACEs.

3.1.1 Virtues of LCS

The LCS framework is designed around the notion that the semantics of any word can be decomposed into a few orthogonal primitives. This is in contrast to classical semantic frameworks such as Logical Form (LF), in which nearly every word in a language has its own incomparable primitive function [1]. For example, in LF every manner of motion gets its own primitive, such as *fly(x,y,z)*, *walk(x,y,z)*, and

[1] This style of LCS notation has been adopted for clarity. The traditional style LCS notation for this sentence is: $[_{Event}\text{CAUSE}[_{Thing}\text{BOY}][_{Event}\text{GO}[_{Thing}\text{BALL}][_{Path}\text{TO}[_{Place}\text{AT}[_{Thing}\text{LAKE}]]]]]$

slither(x,y,z) [1]. Schank's Conceptual Dependency (CD) semantics is similar to LCS in its primitive-based decomposition, but still does not achieve the orthogonality of LCS. For example, CD primitives ATRANS, PTRANS, MOVE, and PROPEL can all be decomposed into LCS $[_{Event}\mathrm{GO}]$ and $[_{Cause}X[_{Event}\mathrm{GO}]]$ frames.

Furthermore, LCS has the most thorough treatment of spatial place and path semantics. First, LCS makes an explicit differentiation between places and paths, so that it is possible to represent the difference between being at a location and going towards a location. In addition, LCS identifies two independent properties of paths: path-types and path-roles. Path-types determine how a reference location relates to the path, and can be:

- *bounded paths*, which specify a source location (usually using "from") or a goal (usually using "to")
- *directions*, which specify a point not actually on the path, using words such as "away from" or "toward"
- *routes*, which specify a location in the interior of the path, for example with the verb "passes" or the preposition "via"

Path-roles determine the function of the path. Given a path, a thing may:

- *traverse*, such as "The mouse skittered toward the clock"
- *extend*, such as "The sidewalk goes around the tree"
- *orient*, such as "The sign points toward Philadelphia"

Taking these two factors together, LCS identifies 9 different classes of path, whereas previous schemes (by Schank and Johnson-Laird) omitted entire classes of paths [9].

Because LCS provides superior decomposition and coverage, it is an outstanding choice for capturing spatial semantics from vision.

3.1.2 Formal coverage of LCS

Here we present an abbreviated version of the LCS formalism [9, 1]. The complete formalism is presented in Table A in Appendix A.

Table 3.1: An abbreviation version of Lexical Conceptual Semantics [9, 1].

<Thing>	←	<*Thing* x>
<Place>	←	<Thing> <*Place* PlaceFunc> where PlaceFunc ∈ (AT, ON, IN, ABOVE, BELOW, BEHIND, . . .)
<Path>	←	<PathElement>* <*Path* > where <PathElement>* indicates any number of PathElements
<PathElement>	←	Thing <*PathElement* PathFunction> where PathFunction ∈ (TO, FROM, TOWARD, AWAY-FROM, VIA, ALONG, . . .)
<Event>	←	<Thing> <Path> <*Event* Go>
	←	<Thing> <Event> <*Event* Cause>
<State>	←	<Thing> <Place> <*State* Be>
	←	<Thing> <Path> <*State* Orient>
	←	<Thing> <Path> <*State* Extend>

3.1.3 Extensions to LCS

The variant of LCS used throughout this thesis differs from Jackendoff's original system in its treatment of Paths and in its inclusion of modifiers.

First, PathElements do not exist as frames in Jackendoff's analysis. Instead, Paths simply have a number of slot-pairs which are each filled with a PathFunction and a reference Thing or Place. The inclusion of PathElements simply clarifies this idea. Also, the present formalism requires that PathFunctions take only Places, not Things, as their arguments. In most cases, conversion from the original system to the present system merely requires wrapping the reference Thing in an $[_{Place}\text{AT}[_{Thing}]]$ clause.

Second, the framework here permits modifiers to be attached to Things and Events. Following the lead of Dorr [4], Propertys, such as BIG, RED, or SOFT, can be attached to Things. Similarly, Manners, such as QUICKLY, BRIEFLY, or FORCEFULLY, can be attached to Events. These modifiers permit the semantic recovery of adjectives and adverbs while parsing. As an example of the usage of modifiers, consider the following semantics for "the red ball:"

$$\left[\begin{array}{ll} \text{MODIFIER} & \left[<\textit{Property}\ \text{RED}> \right. \\ <\textit{Thing}\ \text{BALL}> & \end{array}\right.$$

3.2 Combinatory Categorial Grammar

Any system that interacts with language requires an appropriate grammar and lexicon. The lexicon contains all the words in the language, along with word-specific information required by the grammar, such as syntactic category and semantics (or in common terms, part of speech and meaning, respectively). It is then the function of the grammar to determine how words from the lexicon may combine together and what the resultant semantics is.

Clearly, choosing a grammar system is an important decision. There are many grammars systems to choose from, including Probabilistic Context Free Grammars (PCFG), Head-Driven Phrase Structure Grammar (HPSG), Tree Adjoining Grammar (TAG), and many more.

3.2.1 Mechanics of CCG

The grammar framework chosen for this system is Combinatory Categorial Grammar (CCG) [14]. CCG has many advantages in this system. First, there are only a handful of rules for combining constituents, and these rules are explicit and well defined. These qualities facilitate this system's usage of grammatical rules as the basis for constraints in a constraint propagation network. In addition, CCG is adept at parsing around missing information, because it was designed to handle linguistic phenomena such as parasitic-gapping[2]. The ability to gracefully handle incomplete phrases is crucial in our system, because it enables us to parse around ambiguities and gaps.

The defining feature of categorial grammars such as CCG is that syntactic categories are either:

- One of a small set of atomic elements. Usually this set is {S, N, NP} corresponding to Sentence, Noun, and Noun Phrase.
- A functor, taking other syntactic categories as arguments, and specifying whether to expect these arguments to the left or to the right. The functor also indicates what syntactic category is formed if all the arguments are satisfied.

For example, the syntactic category for "ball" is N, and the category for a determiner such as "the" is NP requiring N to the right. It follows that the phrase "the ball" should have the syntactic category NP, because "ball" satisfies the argument required by "the."

Syntactic category functors are expressed in argument-rightmost curried notation, using slashes to indicate on which side arguments are expected: / indicates an argument to the right, and \ indicates an argument to the left. Thus NP/N indicates a NP requiring a N to the right (and is therefore the syntactic category of a determiner),

[2] An example of a sentence with parasitic gapping is "John hates and Mary loves the movie," where both verbs share the same object. CCG handles this by treating "John hates" and "Mary loves" as constituents, which can then be conjoined by "and" into a single "John hates and Mary loves" constituent (traditional grammars are unable to recognize "John hates" as a constituent.)

while (S\NP)/NP indicates an S requiring one NP to the left and one to the right (this is the category of a mono-transitive verb).

These syntactic categories combine using a few simple productions, such as the following functional application and composition rules:
X/Y Y $\Rightarrow$ X (> *forward application*)
Y X\Y $\Rightarrow$ X (< *backward application*)
X/Y Y/Z $\Rightarrow$ X/Z (>B *forward composition*)
Thus, our intuitive derivation for “the ball” was formally an invocation of forward functional application (>) for NP/N N.

Semantics in CCG are lambda calculus expressions, and each syntactic rule also specifies how to combine the semantics. X:f is used to represent semantic category X with lambda calculus semantics f. The previously stated function application and composition rules, together with their semantic implications, are:
X/Y:f Y:a $\Rightarrow$ X:fa (> *forward application*)
Y:a X\Y:f $\Rightarrow$ X:fa (< *backward application*)
X/Y:f Y/Z:g $\Rightarrow$ X/Z:$\lambda x.f(gx)$ (>B *forward composition*)

For a full description and analysis of CCG, see [14].

3.2.2 CCG Derivations

CCG derivations are standardly written with a horizontal line indicating the application of a rule. The line is labeled to the right with a symbol representing the rule. The symbols are shown in the rule descriptions above. The resulting syntactic category and semantics are written below the line. For example, the derivation for the sentence “The boy kicks the ball” is:

$$\begin{array}{ccccc}
\text{the} & \text{boy} & \text{kicks} & \text{the} & \text{ball}\\
\cline{1-1}\cline{2-2}\cline{3-3}\cline{4-4}\cline{5-5}
\mathrm{NP/N} & \mathrm{N} & (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} & \mathrm{NP/N} & \mathrm{N}\\
\lambda x.x & boy' & \lambda y.\lambda x.(kicks'\ x\ y) & \lambda x.x & ball'\\
\cline{1-2}\cline{3-4}
\multicolumn{2}{r}{>} & \multicolumn{2}{r}{>B} & \\
\multicolumn{2}{c}{\mathrm{NP}} & \multicolumn{2}{c}{(\mathrm{S}\backslash\mathrm{NP})/\mathrm{N}} & \\
\multicolumn{2}{c}{boy'} & \multicolumn{2}{c}{\lambda y.\lambda x.(kicks'\ x\ y)} & \\
\cline{3-5}
 & & \multicolumn{3}{r}{>}\\
 & & \multicolumn{3}{c}{\mathrm{S}\backslash\mathrm{NP}}\\
 & & \multicolumn{3}{c}{\lambda x.(kicks'\ x\ ball')}\\
\cline{1-5}
\multicolumn{5}{r}{<}\\
\multicolumn{5}{c}{\mathrm{S}}\\
\multicolumn{5}{c}{(kicks'\ boy'\ ball')}
\end{array}$$

CCG allows a single phrase to have many different derivations all of which produce the same final results, both syntactically and semantically. For example, consider the following equivalent derivations of “the red ball.”

$$\begin{array}{ccc}
\text{the} & \text{red} & \text{ball}\\
\cline{1-1}\cline{2-2}\cline{3-3}
\mathrm{NP/N} & \mathrm{N/N} & \mathrm{N}\\
\lambda x.x & \lambda y.(red'\ y) & ball'\\
\cline{2-3}
 & \multicolumn{2}{r}{>}\\
 & \multicolumn{2}{c}{\mathrm{N}}\\
 & \multicolumn{2}{c}{(red'\ ball')}\\
\cline{1-3}
\multicolumn{3}{r}{>}\\
\multicolumn{3}{c}{\mathrm{NP}}\\
\multicolumn{3}{c}{(red'\ ball')}
\end{array}
\quad \text{or} \quad
\begin{array}{ccc}
\text{the} & \text{red} & \text{ball}\\
\cline{1-1}\cline{2-2}\cline{3-3}
\mathrm{NP/N} & \mathrm{N/N} & \mathrm{N}\\
\lambda x.x & \lambda y.(red'\ y) & ball'\\
\cline{1-2}
\multicolumn{2}{r}{>B} & \\
\multicolumn{2}{c}{\mathrm{NP/N}} & \\
\multicolumn{2}{c}{\lambda x.(red'\ x)} & \\
\cline{1-3}
\multicolumn{3}{r}{>}\\
\multicolumn{3}{c}{\mathrm{NP}}\\
\multicolumn{3}{c}{(red'\ ball')}
\end{array}$$

In the lefthand derivation, the functional application rule was applied twice, first to join “red” and “ball” into “red ball” (with category N), and then to join “the” with “red ball” producing a noun phrase. In contrast, the righthand derivation first uses the functional composition rule to create the non-classical constituent “the red”, with syntactic category NP/N. This is then joined with “ball” to complete the noun phrase.

3.3 Label-Selective Lambda Calculus

Label-selective lambda calculus (LS λ-calculus) is a variant of lambda calculus extended to include named parameters [5]. These named parameters provide greater flexibility over parameter ordering and omission.

3.3.1 Why LS λ-Calculus is Useful

To understand why one might want to use LS λ-calculus, let us first examine the limitations of traditional λ-calculus. The rules of λ-calculus are based on application ordering, which gives rise to asymmetries in its usage. For example, let us assume that we have a function $f : A \times B \to C$. If we let M be the lambda calculus body of f, then we can express the function as:

$$f \equiv \lambda x.\lambda y.M$$

. Now, if we want to apply an argument on the outermost parameter x, that is, we want to let x equal the constant A, while y remains a free parameter, then we can simply write:

$$(\lambda x.\lambda y.M)A \to \lambda y.[A/x]M$$

On the other hand, if we wish to apply an argument on the inner parameter y, that is, we want to let y equal the constant B while x remains a free parameter, then we must write the more awkward:

$$\lambda z.((\lambda x.\lambda y.Mz)B) \to \lambda z.[B/y][z/x]M$$

This same argument is presented in [5], using Curry's logical combinators S and K:

$$\lambda x.f(a, x) = fa$$

but

$$\lambda x.f(x, b) = Sf(Kb)$$

Either way, the conclusion is the same. Applying an argument to the outermost parameter is simple, applying to an inner parameter is more difficult. Moreover, to apply to an argument to particular lambda (eg, to get the argument substituted into a particular location in the body M,) one must know how deep in the lambda chain the correct parameter is. This in turn requires tracking which enclosing lambda abstractions have already been reduced away.

In our system, we we will be using lambda calculus to maintain the semantics of partial parses. In particular, these partial parses may have gaps in them, where some parameters have yet to be filled (they will be filled as the parse is completed). Furthermore, there may be parameters which never get filled. Both of these points will be explained in more detail in Section 4.1, where we describe the complete semantic system built from Lexical-Conceptual Semantics [9] and LS λ-calculus. The conclusion, though, is that classical λ-calculus complicates the job of out-of-order and optional argument application.

3.3.2 Explicit Function Application Notation

Traditional λ-calculus has implicit application operators, such as the implied operator in the middle of fa. In contrast, LS λ-calculus makes this operator explicit, using the ˆ symbol to indicate application. Thus, fa becomes $f\hat{}a$. With the application operator being made explicit, we can define the term *entity* to mean either a lambda abstraction (λ) or a function application (ˆ).

3.3.3 The LS λ-Calculus System

Label-selective λ calculus weakens the ordering requirements of λ-calculus through the use of labels associated with each λ term and application term. In the full label-selective system, the labels are pairs a, b where a is a string indicating a *symbolic label*, and b is an integer *index* indicating argument ordering ordering within that channel. Subsets of this system provide calculi in which:

- only the symbolic label is employed (*symbolic selective λ-calculus*)

- only the index is employed (*numerical selective λ-calculus*)

Because the fundamental concepts of LS λ-calculus can be understood by considering only symbolic selective λ-calculus, because this is the easiest to understand variant of LS λ-calculus, and because it is most important to this system, the balance of this section will deal only with the symbolic selective variant. The complete label-selective λ-calculus reduction system be found in Table A in Appendix A.

3.3.4 Symbolic Selective λ-Calculus

Symbolic selective λ-calculus is similar to normal λ-calculus, except that each lambda abstraction and each function application is assigned a symbolic label. A lambda abstraction with label a is denoted as λ_a, while a function application with label b is denoted as $\widehat{b}$. Labels are used to control when β-reduction is permitted to occur. Recall that β-reduction is the process of taking a lambda abstraction/function application pair and performing substitution based on them. In classical λ-calculus, this rule is denoted (using the explicit application operator):

Classical β-reduction: $(\lambda x.M)\hat{}N \rightarrow [N/x]M$

In symbolic selective λ-calculus, β-reduction is only permitted when the labels of the lambda abstraction and the function application are identical. If the labels are not identical, then a reordering rule (rule 3, below) instead applies to the pair, allowing the lambda abstraction to be moved outside the application. Finally, two reordering rules (rules 1 and 2, below) allow pairs of lambda abstractions or pairs of function applications to commute, so that their labels are in lexicographical order. Thus, the rules for symbolic selective λ-calculus, as presented in [5], are given in the table below.

Together, these rules effectively establish a different “channel” of application for each symbol, where the ordering of lambda abstractions and function applications within a particular channel is relevant, but the ordering between channels (ie, of abstractions or applications with different labels) is ignored. Each of these channels can be construed of as a named parameter for a function, and we can make an

Table 3.2: Rules of Symbolic Selective λ-Calculus.

	β-reduction			
(β)	$(\lambda_a x.M)\widehat{a}N$	$\rightarrow$	$[N/x]M$	
	Reordering			
(1)	$(\lambda_a x.\lambda_b y.M$	$\rightarrow$	$\lambda_b y.\lambda_a x.M$	$a > b$
(2)	$M\widehat{a}N_1\widehat{b}N_2$	$\rightarrow$	$M\widehat{b}N_2\widehat{a}N_1$	$a > b$
(3)	$(\lambda_a x.M)\widehat{b}N$	$\rightarrow$	$\lambda_a x.(M\widehat{b}N)$	$a \neq b, x \in FV(N)$

application to a particular function parameter without concerning ourselves with the relative position of the lambda abstraction in the lambda chain.

3.4 Streams and Counterstreams

A crucial part of any parsing or generation algorithm is a search procedure. Because we have context coming in from both sides of the search space (linguistic and non-linguistic semantic sides), a bidirectional search is most appropriate. In choosing a bidirectional search strategy, extra merit should be given for neural plausibility, for ability to support observed facets of cognition, and for ability to be extended to support other contextual clues. All of these factors support the choice of Shimon Ullman's Streams and Counterstreams [16] model for bidirectional search.

3.4.1 The search model

To understand the Streams and Counterstreams model, let us first start with a search space consisting of states and transitions, such as in Figure 3-1. In a typical unidirectional search model, a small number of states are marked as *source states*. One or more *goal states* are also designated, either explicitly or indirectly by specifying a class of goal states which satisfy a particular success criterion. A search process starts at the source state, and uses some strategy to choose sequences of state-transitions, in hopes of finding a path to a goal state.

In contrast, Bidirectional search models treat the source and goal symmetrically; the search-space is traversed both forward from the source states and backward from the goal states. The search processes operating in each direction interact with each other whenever their paths intersect in the search-space. This interaction provides hints for quickly completing the remainder of the search. For example, in the simplest bidirectional search models, the forward and backward search processes operate independently, until such time as the forward searcher reaches a state already encountered by the backward searcher. At this point, the forward searcher quickly reaches the goal by tracing the backward-path. (See Figure 3-2).

The specific style of bidirectional search employed in this system is based on Streams and Counterstreams [16], in which forward and backward search processes interact with each other by means of primed pathways. For each transition, two priming values are maintained: a forward priming and backward priming[3]. Primings are used when a decision must be made between several possible transitions that could extend a search path; those transitions that have a higher priming are preferred for expansion, using the forward priming for forward searches, backward priming for backward searches. Transition primings in a particular direction (either forward or backward) are increased whenever a search path traverses the transition in the *opposite* direction. The net influence of the primings is that transitions previously traversed in one direction are more likely to be explored in the opposite direction, if the opportunity arises. By extension, primings provide clues for finding a path from any state to the target state. In the example shown in Figure 3-2, Streams and Counterstreams would have primed the center frame's darkened transitions in the direction opposite of the arrow. When the upward search reaches the F node in the final frame, the search would be more likely, though not guaranteed, to make F→B the transition explored in the upward direction.

[3] Ullman [16] actually presents Streams and Counterstreams in terms of primed state-nodes, rather than primed transitions. The underlying concept is the same; because this system actually uses the transition-priming variant, it is presented here to avoid later confusion.

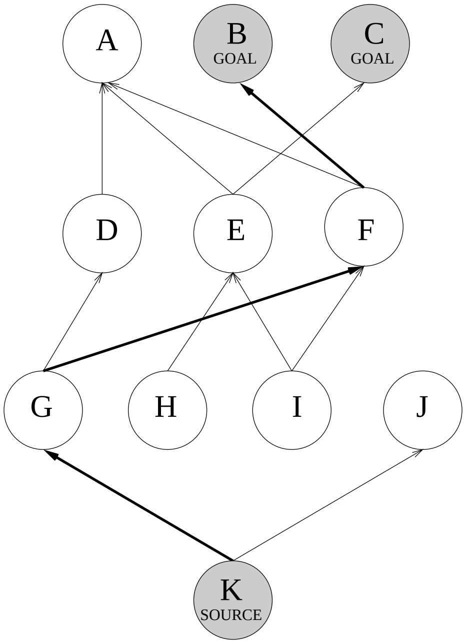


Figure 3-1: This figure depicts a typical unidirectional search space, with circles indicate states and arrows indicating transitions. A unidirectional searcher seeks to find a path from the source state to one of the goal states. The depicted search space contains only one such path, highlighted with bold transition arrows.

3.4.2 Advantages for Streams and Counterstreams

The streams and counterstreams model is attractive in terms of its neural plausibility, its support for observed phenomena, and its extensibility.

In his presentation of Streams and Counterstreams, Ullman [16] makes it a point to back the model up with supporting neurological evidence. A model is laid out showing how Streams and Counterstreams could be achieved by neural groups. Furthermore, much neurological evidence is presented in support of the model's viability, including the extremely reciprocal nature of bottom-up and top-down pathways in the cortex, and the observation that the cortex is formed of several layers, with regular patterns of forward, backward and lateral connections. These data help to convince us that Streams and Counterstreams is not an outlandish proposal.

Furthermore, the model supports many observed facets of cognition. Cognitive science literature is rife with observations of primed concepts and associations. Priming is often implicated in situations where people are presented with a situation, and

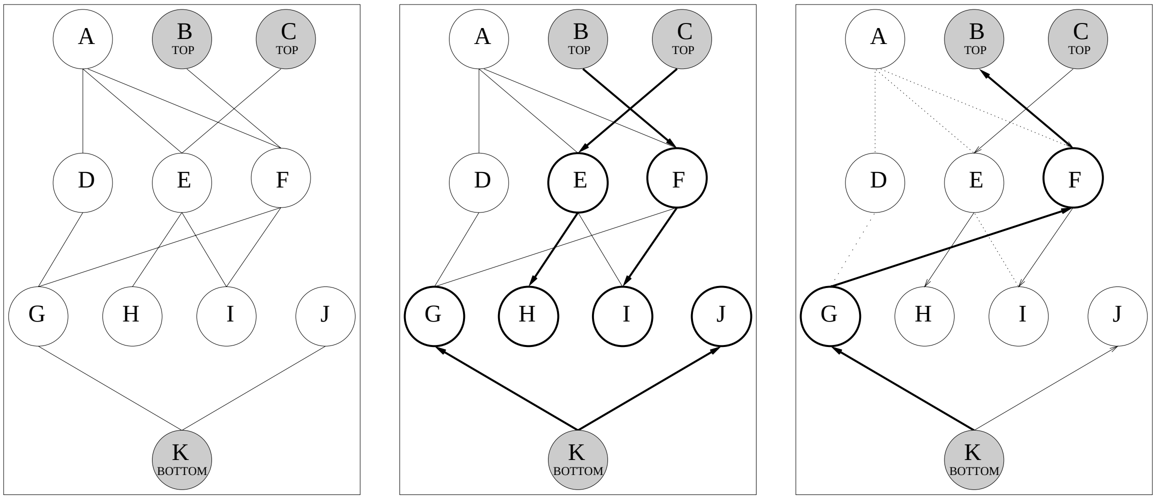


Figure 3-2: This figure shows a typical search space for bidirectional search. The "source" and "goal" states of the unidirectional search space have been replaced with "top" and "bottom" states. The bidirectional searcher seeks to find a pathway between the "top" and "bottom" states. The left frame shows that space at the beginning of the search. The searcher works from both the top and the bottom states simultaneously. The middle frame shows the search in progress, before the bottom-up and top-down processes meet. Nodes that have been discovered have bold edges, the transitions used to discover these nodes are also marked in bold. In this example, breadth-first search is being conducted from each of the source states. The G and J have been discovered by upward search from K. The F and I nodes have been discovered by downward search from B. The E and H nodes have been discovered by downward search from C. Let us assume that upward search from K is the next to expand, and that it expands to F next. Because F has already been found by downward search from B, the upward search from B will quickly follow the trail used by the downward searcher. The completed search is shown in the last frame.

they either make different choices or are faster to reach a correct answer, based on what types of experience they have had in the very recent past. Priming can be considered a cognitive advantage because it is often the case that a new problem situation is similar to, or at least related to, other recently considered situations. Therefore, recently activated concepts and associations are likely to continue to be useful; the priming mechanism keeps these highly available. One convincing example of just how deeply priming can affect thought is the work done by Lera Boroditsky [3] in which visual priming of spatial reasoning concepts leads to different interpretations of time metaphors such as "move the deadline forward two days." In the experiments, subjects were led to think about spatial scenarios before interpreting time metaphors. The results showed that if subjects were led to think of themselves moving through a stationary environment, then the above time metaphor is much more likely to interpreted as meaning there are two additional days before the deadline. In contrast, if the subjects are lead of themselves as stationary while the environment moves past them, then they are significantly more likely to interpret the metaphor as implying that there are two fewer days before the deadline. Thus, priming in the spatial domain is found to bias the interpretation of language involving the time domain! Because the Streams and Counterstreams model has priming at its core, it is ideally positioned to investigate cognitive priming in an artificial intelligence environment.

Finally, the model is extendible. Many factors may come to bear on the final solution of difficult cognitive problems such as parsing. The choice between technically correct and incorrect options is often clear. However, there are regularly many technically correct options, and the best of these (in relation to the current situation) must be chosen. Many hints as to the best choice are available, not only directly through priming, but also through other heuristics, often acquired through experience. For instance, much research has gone into statistical heuristics for pruning the parsing search space. Deniz Yuret's work on Lexical Attraction [20], for example, shows amazing ability to infer phrase structures of novel linguistic inputs purely through statistical analysis of the co-occurrence of words it inputs that the system has previously parsed. A parser based on Streams and Counterstreams can incorporate Lexical Attraction by arranging for transitions which bring together lexically attracted words to have primings that are naturally higher than they would be otherwise. We conclude that Streams and Counterstreams is in a good position to benefit from such statistical hints, because the biases can easily be integrated into the system as primed pathways.

3.5 Structural Alignment

Structural alignment is a process presented by Gentner and Markman [6] for transferring information between partially dissimilar representational structures. During structural alignment, corresponding elements in two structures are identified by matching. Correspondence between non-matching elements is then implied by the structural constraints of the representations. For example, in Figure 3-3, structural alignment first matches A, E, and F between the two representations. Then, based on structural constraints, 1 is inferred to correspond with C, and 2 with B(D).

In its original presentation, structural alignment was used as an engine to conduct analogy. Commonalities between the two elements related under the analogy were aligned, bringing dissimilar elements into alignment. These aligned differences were then used as candidate locations for inference between the analogy elements. In the present architecture, structural alignment of the semantics of partial parse structures with semantic structures in the target domain (the non-linguistic semantics during parsing, and the semantics of words in the linguistic domain during generation) will be used to reduce ambiguities by transferring clarifying information across alignment points.

3.5.1 Steps for Structural Alignment

Gentner and Markman [6] identify *structural consistency* as the criterion for successful structural alignment. Structural consistency requires two main features:

```
                    X
                   /
  A               A
 / \             / \
2   C           B   1
   / \          |  / \
  E   F         D E   F
```

Figure 3-3: Structural alignment between these two structures infers the correspondences C↔1 and 2↔B(D). Structural alignment between semantic representations will bring unknown words into correspondence with their probable semantics.

- *Parallel Connectivity*: Matched entities related by a structural relation are required to be related by the *same* structural relation ship. For example, in figure 3-3, if we assume that children in the tree are ordered (that is, being the left-child of node P is semanticly different from being the right-child of node P), then the alignment A↔A and 2↔1 is illegal, because in the first structure, 2 is the left-child of A, while in the second structure 1 is the right-child of A.

- *One-to-One Correspondence*: When an element appears multiple times in a representation, it is limited to always match the same element in the other representation.

In order to achieve these structural consistency requirements, Gentner and Markman propose conducting structural alignment in three stages. First, individual elements are matched locally between representation, without regard to structural properties. Second, structurally consistent sets of local matches are calculated. Finally, these structurally consistent matches are used to align large, partially dissimilar sections of the representation.

Chapter 4

System Construction

This chapter describes how the techniques presented in Chapter 3 are combined and extended to realize a bidirectional reconciliatory exchange between linguistic and non-linguistic semantic inputs. This chapter has one section devoted to each major feature of the system.

- Section 4.1: *λ-LCS*, a unification of Lexical Conceptual Semantics with Label Selective λ-Calculus.

- Section 4.2: *Label-Selective CCG* (LS-CCG), an extension of CCG to employ Label Selective λ-Calculus as its semantic system. Together with λ-LCS, this provides the linguistic framework (grammar and semantics) for the system.

- Section 4.3: *CCG-based Constraint Propagation*, parsing and generation using CCG rules as constraints in a constraint propagation network

- Section 4.4: *Serialized SCS*, an implementation of Streams and Counterstreams on a serial processor.

- Section 4.5: *SCS Parsing*, using Serialized SCS to bidirectionally construct the appropriate LCS/CCG constraint network topology for specific linguistic and non-linguistic inputs.

4.1 Lexical Conceptual Structures and Lambda Calculus

This section describes a unification of Lexical Conceptual Semantics (LCS) with Label-Selective Lambda Calculus (LS λ-Calculus). This will serve the basis semantics in the system, for words in the lexicon, for non-linguistic semantics, and for partial parses during reconciliation.

4.1.1 Why parameterize LCS with LS-λ calculus?

The semantic manipulations of CCG are founded on λ-calculus, so it is natural to stick with some variant of λ-calculus for our parameterization of LCS semantics. However, understanding the motivation for choosing the label-selective variant of λ calculus requires us to investigate the relation we need to establish between CCG and LCS.

CCG associates a syntactic category with each linguistic constituent during parsing. When parsing begins, each word has its own syntactic category. As the parse continues, syntactic categories are assigned to phrases, even traditionally incomplete phrases such as "Anna married." As described in Section 3.2, the syntactic categories assigned are functional descriptions based on the number, type, and directionality of expected arguments. This is all straightforward for grammatical constructions which take a fixed number of parameters. For example, a mono-transitive verb such as "married" always takes a subject noun phrase to the left, and an object phrase to the right. Therefore, its syntactic category is simply (S\NP)/NP.

The story gets a little more complicated when we consider grammatical constructions which take a variable number of parameters, such as a noun taking adjectives or a verb taking prepositional phrases. Because the number of optional adjuncts cannot be known ahead of time, the question is: what are the appropriate syntactic categories for nouns and adjectives? Clearly we cannot have a construct such as N\Adj, because we don't know how many adjective parameters to include. Therefore, the standard CCG approach is to allow the syntactic category for nouns to remain N, while making the category for adjectives be N/N. That is, an adjective takes a noun on the right and produces a noun when it combines. In this way, any number of adjectives can be chained onto a noun.

This choice has implications for the semantics of expressions. Specifically, it forces us to write the semantics of an adjective as a function that takes a noun-semantics as its argument. For example, in standard CCG semantics the phrase “the bouncy red ball” would produces the semantic structure $bouncy(red(ball))$. While successful logic systems have been built using semantic structures such as these, some important questions are raised. For example, what is the relation between $bouncy(red(ball))$ and $red(bouncy(ball))$? Does $red(ball)$ return the same type of semantics that *ball* alone does, because they can both be parameters to $bouncy()$?

Furthermore, LCS does not support having the adjectival construction surround the noun construction. In all other LCS frames, the outer frame by itself provides a semantic pattern, whose details are filled in by sub-frames. For example, a GO frame by itself already conveys the gist of any substructure that could be rooted at that GO frame. The parameters of the GO frame merely specify details of the GO. In contrast, an Adjective frame which takes a Thing frame as a parameter would not follow this paradigm. Furthermore, other LCS frames which require Things, such as the GO frame:

$$\left[\begin{array}{l} <\textsc{Thing}> \\ <\textsc{Path}> \\ <\textit{Event}\ \textsc{Go}> \end{array}\right.$$

would be incompatible with the adjective-enhanced thing frame: $\left[\begin{array}{l} \quad \left[<\textit{Thing}\ \text{BALL}>\right. \\ <\textit{Adjective}\ \text{RED}> \end{array}\right.$

Instead, the LCS-style treatment of adjectives [4, 9] is to embed them under the Thing frame, so “red ball” becomes:

$$\left[\begin{array}{l} \text{MODIFIER}\ \left[<\textit{Property}\ \text{RED}>\right. \\ <\textit{Thing}\ \text{BALL}> \end{array}\right. .$$

Unfortunately, the structural relationship of the entities in the LCS structure are opposite of those of the typical CCG *red*(*ball*) construction. One possible solution would be to keep the *red*(*ball*) surface appearance, letting *ball* get replaced with $\left[< \textit{Thing}\ \text{BALL} > \right.$, and writing a non λ-calculus implementation of *red* as a mutator which modifies $\left[< \textit{Thing}\ \text{BALL} > \right.$ by inserting the $\left[< \textit{Property}\ \text{RED} > \right.$ modifier. This would introduce an additional layer of complication, because the actual semantics of *red* could not be expressed in λ-calculus. This means that the semantics of *red* in the lexicon would be incomparable with the semantics of *red* after it has been applied to a noun, without a matcher that non only supports LCS matches and λ calculus matches, but also matches on whatever format the *red* mutator is actually written. Since matching will be a critical part of constraint propagation, this solution is avoided in favor of the more unified approach of label-selective lambda calculus.

Using LS λ-calculus, we can remain completely in the domain of λ-calculus throughout the semantic manipulation and matching process. In this paradigm, we can write the semantics for a noun as a λ expression which takes an unrestricted number of optional parameters for property modifiers. That is, the semantic representation for "ball" is:

$$\lambda_{property,*} p. \left[\begin{array}{l} \text{MODIFIER p*} \\ < \textit{Thing}\ \text{BALL} > \end{array} \right.$$

In this semantics, the * indicates the collapsed inclusion of an infinite number of indexed lambda-abstractions on the label *property.* This makes full use of the indexing features of LS λ-calculus, in addition to the symbolic labeling features. The details of these indexing operations are not critical, so long as it is understood that LS-λ calculus provides the support for managing and reducing these indexed constructions. For simplicity, the remainder of this thesis will not explicitly use this infinite index set notation.

With this semantic representation for "ball", we can then define the semantics of "red" to be:

$$\lambda_{noun} n. \left(n_{\widehat{property}} \left[< \textit{Property}\ \text{RED} > \right. \right)$$

4.1.2 Unified representation of LCS and LS-λ calculus

In order to keep a consistent and coherent representation and notation, the current system treats λ-calculus entities for lambda abstraction and functional application as frames similar to the LCS frames. Thus, the following equivalences are introduced:

$$\lambda_a x.M \quad \equiv \quad \left[\begin{array}{l} M \\ <\lambda_a \ x> \end{array}\right.$$

$$M\widehat{a}N_1 \quad \equiv \quad \left[\begin{array}{l} M \\ N_1 \\ <\widehat{a}\ > \end{array}\right.$$

Using the unified representation, the semantics for "red" and "ball" are given by:

$$\textrm{red} \quad := \quad \left[\begin{array}{l} \left[\begin{array}{l} \textrm{n} \\ \quad \left[<\textit{Property}\ \textrm{RED}> \right. \\ <\widehat{\textit{property}}\ > \end{array}\right. \\ <\lambda_{noun}\ \textrm{n}> \end{array}\right.$$

$$\textrm{ball} \quad := \quad \left[\begin{array}{l} \left[\begin{array}{l} \textrm{MODIFIER p} \\ <\textit{Thing}\ \textrm{BALL}> \end{array}\right. \\ <\lambda_{property}\ \textrm{p}> \end{array}\right.$$

We then derive the semantics of "red ball" as follows:

$$red\ \widehat{noun}\ ball \quad \rightarrow \quad \left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \textrm{n} \\ \quad \left[<\textit{Property}\ \textrm{RED}> \right. \\ <\widehat{\textit{property}}\ > \end{array}\right. \\ <\lambda_{noun}\ \textrm{n}> \end{array}\right. \\ \left[\begin{array}{l} \left[\begin{array}{l} \textrm{MODIFIER p} \\ <\textit{Thing}\ \textrm{BALL}> \end{array}\right. \\ <\lambda_{property}\ \textrm{p}> \end{array}\right. \\ <\widehat{noun}\ > \end{array}\right.$$

$$\rightarrow \left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \text{MODIFIER p} \\ <\mathit{Thing}\ \text{BALL}> \end{array}\right. \\ <\lambda_{\mathit{property}}\ \text{p}> \end{array}\right. \\ \left[<\mathit{Property}\ \text{RED}> \right. \\ <\widehat{\mathit{property}}\ > \end{array}\right.$$

$$\rightarrow \left[\begin{array}{l} \text{MODIFIER} \quad \left[<\mathit{Property}\ \text{RED}> \right. \\ <\mathit{Thing}\ \text{BALL}> \end{array}\right.$$

4.2 Extending CCG with Label-Selective Lambda Calculus

In order to complete our integration of label selective λ-calculus into the grammatical and semantic framework, we must extend the CCG framework to support the management of LS λ-calculus labels.

4.2.1 Labeled Syntactic Categories

The first step in adapting CCG to Label Selective λ-calculus is to note that syntactic categories are responsible for tracking the function arguments, and therefore they should also track the label with which those arguments should be applied. Let us continue with the "red ball" example from the previous section. In traditional CCG, "red" has the syntactic category N/N, while "ball" has the category N. However, inspecting the LS-λ calculus, we find that we also need to know that we should apply the semantics of "ball" to the semantics of "red" using the label *noun*. We can record this information if we extend the syntactic category definition of "red" to be N/N.noun, where the ".noun" clause indicates that the syntactic category function argument immediately to its left should be applied on the *noun* channel.

4.2.2 Labeled Syntactic Rules

The second step in adapting CCG to Label Selective λ-calculus is to adapt the syntactic rules to make use of the labeled syntactic categories when generating semantics. This is essentially a matter of analyzing which semantic parameter is being filled, finding its corresponding syntactic parameter, and using the same label. For example, in plain CCG, the simple functional application and composition rules were:

$$\begin{array}{llll} \text{X/Y:}f & \text{Y:}a & \Rightarrow \text{X:}fa & (>\ \textit{forward application}) \\ \text{Y:}a & \text{X}\backslash\text{Y:}f & \Rightarrow \text{X:}fa & (<\ \textit{backward application}) \\ \text{X/Y:}f & \text{Y/Z:}g & \Rightarrow \text{X/Z:}fg & (>\text{B}\ \textit{forward composition}) \end{array}$$

In Label-Selective CCG, these rules become:

$$\begin{array}{llll} \text{X/Y.label:}f & \text{Y:}a & \Rightarrow \text{X:}f\widehat{\mathit{label}}a & (>\ \textit{forward application}) \\ \text{Y:}a & \text{X}\backslash\text{Y.label:}f & \Rightarrow \text{X:}f\widehat{\mathit{label}}a & (<\ \textit{backward application}) \\ \text{X/Y.fX:}f & \text{Y/Z.gX:}g & \Rightarrow \text{X/Z.gX:}f\widehat{fX}g & (>\text{B}\ \textit{forward composition}) \end{array}$$

4.3 CCG-based Constraint Propagation

This section describes how λ-CCG with LCS semantics can be used as the foundation for a constraint propagation network.

4.3.1 The Reconciliation System as a Constraint Propagation Network

The reconciliation system for linguistic and non-linguistic semantic inputs may be viewed as a single large constraint, as in Figure 4-1. This constraint has two inputs: on one side, it takes a set of semantic representations with non-linguistic origin. On the other side, it takes a linguistic input string, together with possible meanings for each word in the string, as determined by a lexicon (treating unknown words as having any possible meaning). As output, the constraint eliminates, in each input set, all meanings which do not lead to a successful structurally aligned parse.

In order to achieve such a complicated constraint, it is useful to decompose the constraint into a network of simpler constraints, each working over a local domain

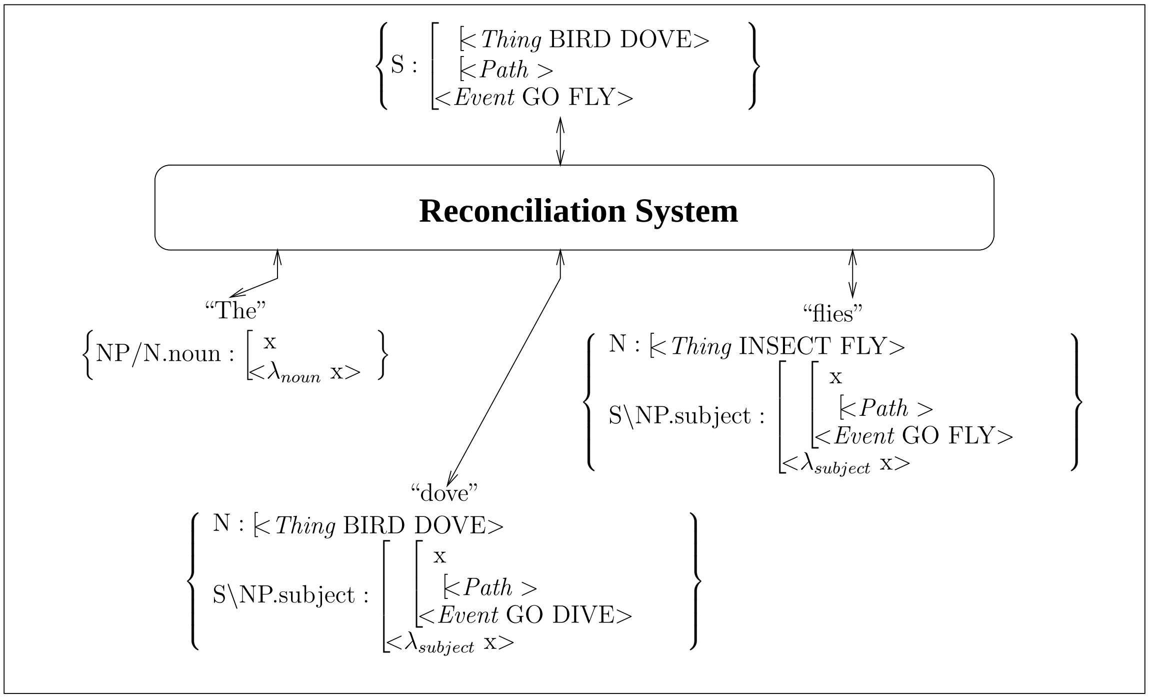


Figure 4-1: The reconciliation system functions as a constraint on linguistic and non-linguistic interpretations, requiring that expressions follow grammatical rules and that they produce alignable semantics. This example shows the system presented with the sentence "The dove flies," and with a corresponding conceptual structure (coming from the non-linguistic domain). In this situation, the system will eliminate the interpretations of "dove" as a GO and flies as THINGs. Furthermore, if the word "dove" had not been known, the system would still select the verb form of flies (by alignment), which brings "dove" into alignment with the appropriate fragment of semantic structure (the THING frame).

of only a few constituents rather than over the domain of an entire sentence, as in Figure 4-2. We can then base these subconstraints on grammatical rules over a fixed number of constituents, and trust the composed network to handle the complete sentence.

4.3.2 Definitions of constraint terms

The remainder of this thesis will use the following terminology to describe constraints in the constraint propagation network. Each location in the network containing a value set will be referred to as a *port*. The actual constraint, in this case implementing

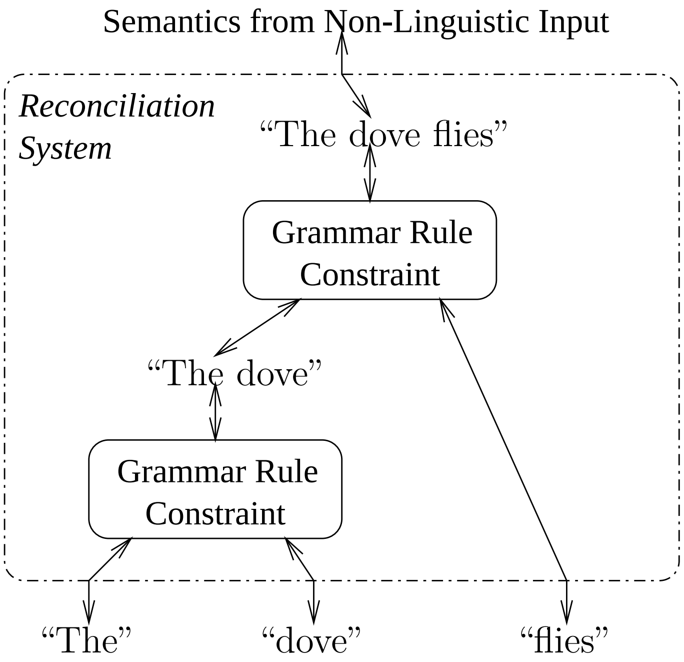


Figure 4-2: The reconciliation constraint in Figure 4-1 is actually implemented as a network of simpler constraints, as shown here. Each constraint implements a grammatical rule, as shown in Figure 4-3.

one of the CCG rules, will be referred to as a *rulebox*. Thus, figure 4-3 shows one rulebox, labeled CCG Constraint, and three ports connected to this rulebox. The process of applying the rule embodied by a rulebox, in order to restrict the value sets in the ports connected to that rulebox, is called *firing* the constraint.

4.3.3 Implementing a Constraint from a CCG rule

Implementing a CCG constraint, such as the one in Figure 4-3, is straightforward when viewed from the highest level. In the figure, the topmost port corresponds to the result of reducing according to one of the CCG rules. The bottom ports each correspond to one of the terms in the rule. We can fire the rule box using the following algorithm:

```
Fire($rulebox$)
$results \leftarrow \{\}$
$C \leftarrow$ the Cartesian product of the rulebox ports' valuesets
```

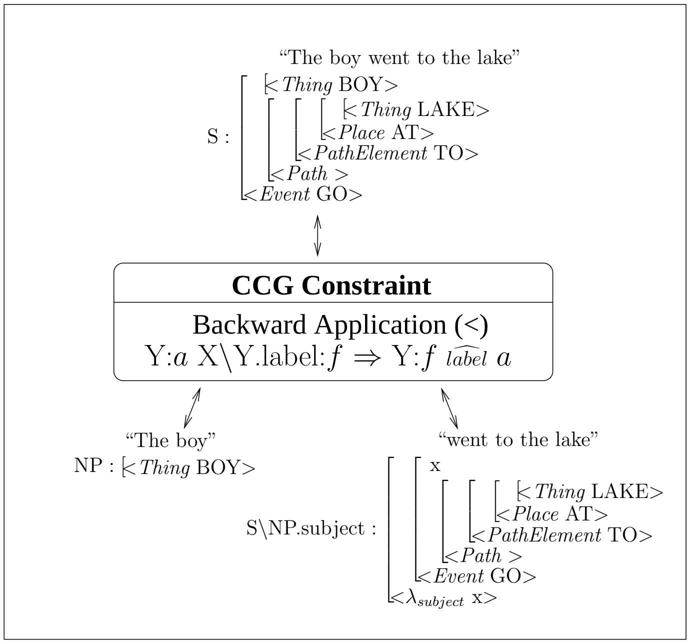


Figure 4-3: This figure shows one of the CCG Rules, Forward Functional Application, being treated as a constraint in a constraint propagation network. Any one of the three inputs can be left unspecified, and the constraint can completely determine the value based on the other two inputs.

```
foreach $x \in C$:
    match each value in x against appropriate CCG rule term
    if match is conflict-free:
        $y \leftarrow$ instantiate each term of CCG rule, using match bindings
        $results \leftarrow results \bigcup \{y\}$
update rulebox ports' valuesets from $results$
```

To summarize, this algorithm matches values in ports connecting to a rule box against the terms of CCG rule embodied by that rule box. If there is a value in each of the ports such that all the terms of the CCG rule can be matched with consistent match-bindings, then those values are allowed to maintain in the ports' valuesets, though they may be modified to remove any ambiguities that could be resolved through matching.

4.3.4 Challenges for matching λ-LCS expressions

The difficulty in the constraint firing algorithm given above is creating an appropriate matching algorithm. For example, let us once more consider the situation in figure 4-3. For this constraint to fire successfully, it must be able to match the CCG rule production term against the value in the production port:

$$\mathrm{Y} : f \ \widehat{\mathit{label}}\ a \ \simeq \ \mathrm{S} : \left[\begin{array}{l} \left[<\!\mathit{Thing}\ \text{BOY}\!> \right. \\ \left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \left[<\!\mathit{Thing}\ \text{LAKE}\!> \right. \\ <\!\mathit{Place}\ \text{AT}\!> \end{array}\right. \\ <\!\mathit{PathElement}\ \text{TO}\!> \end{array}\right. \\ <\!\mathit{Path}\ \!> \end{array}\right. \\ <\!\mathit{Event}\ \text{GO}\!> \end{array}\right.$$

The challenge here is that the elements that correspond to f, a, and the application $\widehat{\mathit{label}}$ in the rule pattern are not explicitly present in the port value datum. These elements have already been removed by β-reduction, or alternately they were never explicitly in the datum structure because the datum structure came from a non-linguistic source. In order to perform matching, we need methods that can undo the reductions dictated by LS-λ calculus, returning the data structure to its pre-reduced state.

4.3.5 Reversing LS λ-calculus reductions: fronting and β-expansion

There are four reduction rules in LS λ-calculus, and each one of them must be reversible in order to allow us to massage the datum structure into a format that matches the pattern.

The reordering rules are simple to reverse. The rules tell us how to exchange two entities (λ abstractions or function applications), in addition to providing a condition on when such an exchange should be considered a reduction (eg, when swapping the entities brings their labels into lexicographical order). All that we must modify in the reordering rules is the condition on which swapping the entities is preferred to leaving them unswapped[1]. To determine this preference, we define a new operation called *fronting*. The fronting operation is similar to the operation which reduces λ-calculus expressions, except that the fronting operation also takes an entity (a specific lambda abstraction or function application frame) as a parameter. The fronting operation then applies the reordering rules whenever they bring the specified entity closer to the outermost position in the entity chain. The operation succeeds if the specified entity can be brought all the way to the outermost position; otherwise, if there are no more applicable reordering rules and the entity could not be brought to the outermost position, the operation fails.

The β-reduction rule is significantly more complicated to reverse, because information, such as labels and variables, is lost during β-reduction. To further complicate matters, for any reduced λ-calculus expression, there exist an infinite number of λ-calculus structures which will reduce to the given expression. We therefore seek to write a *β-expansion* operation, which hallucinates the lost information and which uses heuristics to limit exploration of the infinite number of possible β-expansions.

We start by observing that β-reduction operation of LS-λ-calculus can only eliminate pairs of entities with a λ abstraction embedded inside a function application of the same label. It follows that the β-expansion operation will only be permitted to hallucinate pairs of entities with a λ abstraction embedded inside a function application of the same label. Therefore, no λ abstractions hallucinated by β-expansion will be frontable. As we shall see, fronting of an entity will be a requirement for matching. This will mean that β-expansion is only useful for matching against function application frames in the pattern.

The first step of the β-expansion of a λ-calculus structure M is to compute all possible locations for extracting a variable from the structure. That is, we find all locations where a substructure can be replaced with a variable. We use a temporary variable, let us say x, and compute all these variable extractions. For each variable

[1] A second case of rule 3 must also be added. As written in section 3.3, rule 3 only permits an application to be moved from inside to outside a λ abstraction. The new case of rule 3 also permits an application to be moved in the opposite direction, from outside to inside the λ abstraction.

extraction, let M_x be the structure M with a substructure replaced by x, and let N be the substructure that x replaced. Because only complete substructures are replaced, the number of variable extractions grows in proportion to the number of substructures in M. Since M must have a finite number of substructures, we know that there are also a fixed number of variable extractions.

The second step in β-expansion is to use the variable expansions to create λ-calculus expressions. This is the point at which the set of infinite possibilities is introduced. However, working with the system has revealed that only two possibilities are fruitful. These are:

$$\text{Standard expansion:}\quad \left[\begin{array}{l} \left[\begin{array}{l} \mathrm{M}_x \\ <\lambda_{label}\ x> \end{array}\right. \\ \mathrm{N} \\ <\widehat{label}\ > \end{array}\right.$$

$$\text{Inverted expansion:}\quad \left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} y \\ \mathrm{N} \\ <\widehat{label2}\ > \end{array}\right. \\ <\lambda_{label}\ y> \end{array}\right. \\ \left[\begin{array}{l} \mathrm{M}_x \\ <\lambda_{label2}\ x> \end{array}\right. \\ <\widehat{label}\ > \end{array}\right.$$

The standard expansion handles most of the semantic derivations. The inverted expansion makes possible the structure-reversing derivations such as the adjective-noun derivations discussed in section 4.1. The variables x and y, and the labels *label* and *label2* are placeholders for actual variable and label names. When the β-expansions are later matched against a pattern, they will take on the value in that pattern.

4.3.6 Matching λ-LCS Expressions

Matching of a λ-calculus datum against a pattern then proceeds as follows. The pattern structure is walked, starting with the outermost frame. If the pattern frame is not an entity, that is, if the pattern frame is a standard Lexical-Conceptual Semantics frame, then standard structural matching is pursued, binding meta-variables[2] in the pattern appropriately.

If the pattern frame is an entity, then the matcher tries to front an entity of matching type (λ-abstraction or function application) and matching label in the datum structure. If fronting is successful, then the pattern frame is matched with fronted datum frame and the matcher recurses into the substructures of these frames.

If the pattern frame is an entity, and furthermore it is a function application frame, then the β-expansion operation is applied to the datum. For each resulting expansion, the β-expanded entity frame (the function application frame on the label *label* in the description above) is fronted. If the fronting operation is successful, then matching continues as above, with the pattern frame being matched to the fronted datum frame and the matcher recursing into the substructures of these frames.

4.4 Serialized Streams and Counterstreams

The streams and counterstreams model of bidirectional search, as presented in section 3.4 is a parallel search model. It relies on parallel exploration of the search space in opposing directions. In order to provide a more complete description of how Streams and Counterstreams could be implemented, while avoiding the difficulties inherent in parallelism, we develop here a serial version of Streams and Counterstreams.

4.4.1 Operating in Serial

It is relatively straight-forward to run Streams and Counterstreams serially. Rather than execute a whole set of parallel search processes (starting from each source state on both the top and bottom of the search space), we execute one search process at a time, each starting from one of the source states and proceeding in the appropriate direction. Each search process executes a beam search, with a beam width of 1. At each node in the search path, primings are used to stochastically choose which of

[2] The variables to be bound in the pattern are referred to as meta-variables to differentiate them from the standard variables used as part of the λ-calculus expressions

several path-extension transitions the beam should follow, giving preference to highly primed transitions. Once every source state has been explored in the appropriate direction, the cycle repeats until an external signal cues that the search should be terminated.

Because the search cycles repeatedly until interrupted, we may assume that over time a large number of these beam-search processes originate from each search node. In the absence of primings, this search should stochastically approximate an exhaustive search. In the presence of primings, the search should function exactly like the original parallel version of Streams and Counterstreams.

4.4.2 Time-limited answers

As the search continues to cycle, the best answers (according to a heuristic evaluation function) accumulated so far from one of the probe-searches are always available. Thus, the search procedure always uses whatever time it has been given to provide the best answer it can, with the most bidirectional influence. The search can be terminated if the heuristic determines that the best possible solution has been found. Alternately, in an interactive system, the search can be terminated or modified in response to external stimuli.

4.4.3 Implementing Primings

The final aspect of Streams and Counterstreams remaining to be specified is the method for storing primings. To achieve this, we require that every search transition can produce a set of *priming keys*. A priming key can be any constant identifier, for example a string or a number. There are two critical requirements for the priming keys:

- All semantically equivalent transitions should produce identical priming keys

- Reciprocal transitions (that is, equivalent transitions in the forward and backward search directions) should produce identical priming keys

We then implement primings by maintaining tables mapping priming keys to stored priming values. We keep one table for each of the forward and backward search directions. The priming for a transition is calculated by getting the transition's priming key and looking it up in the the priming table for the current search direction. Primings are updated whenever the beam-search process finishes a search path. Each transition on that search path has its priming key generated. The entry in the priming table for that direction *opposite* of the current search direction is then increased.

4.5 Constructing the Constraint Topology with Streams and Counterstreams

As described in section 4.3, we seek to achieve our reconciliation goals by using a constraint propagation network to bridge the linguistic and non-linguistic inputs. The constraints in this network are parse rules from Label-Selective CCG using λ-LCS as its semantics. Because each constraint embodies the application of a parse rule, the topology of our constraint network necessarily embodies a parse tree[3] for any input it can handle. Because the inputs to our system do not include this parse tree, we must consider how to generate an appropriate constraint network topology.

The system in this thesis employs the serialized Streams and Counterstreams bidirectional search technique to construct the constraint network topology required for reconciliation. In 4.5.1, I discuss possible alternatives to bidirectional search, and why bidirectional search is preferred. The remainder of this section then describes how Streams and Counterstreams searching is realized, including the definition of the search space, the means of generating priming keys, and finally the methods for closing the priming loop.

[3] or generation tree, if viewed from the non-linguistic side

4.5.1 Bidirectional Search is Best for Topology Construction

Several options are available for providing the required constraint network topology. One such option is to use the same network topology to handle all sentences of the same length. For example, we can create a constraint network version of the Chart parsing technique (a dynamic programming parsing algorithm, described in the context of CCG by Steedman [14]). Such a network would have to contain every possible parse tree, and thus performing constraint propagation on this network would essentially result in an exhaustive search of the parse space. While such an approach is guaranteed to generate all appropriate parse trees, it does so at great cost, because it still must consider every incorrect parse tree.

A better solution would be to avoid the exhaustive search by constructing a custom constraint topology for each sentence, using standard heuristic parse techniques. The drawback to this approach is that we are not interested in finding just any potential parse of a phrase/sentence, nor even the most statistically probable parse. Because our intent is to perform structural alignment with input from the non-linguistic domain, our goal in parsing is to find the semantic parse structure which aligns best with the semantic structure input from the non-linguistic domain. It follows that we should use the non-linguistic input to guide our search.

4.5.2 The Search Space

Here I define the search space for constructing the constraint network topology using Streams and Counterstreams. This includes defining the states in the search space, as well as the transitions that lead between them.

First, let us consider what information is included in the states that will serve as sources for the forward and backward search processes. The "forward" and "backward" directions are symmetric under Streams and Counterstreams, but for clarity we will call the parsing pass the forward pass, and the generation pass the backward pass.

The forward, or parsing, process starts with a linguistic input: a list of words, each with an associated set of entries defining the syntactic category and semantics of the various interpretations of the word. Each word corresponds to a port in the constraint propagation network. As yet, there are no rule-boxes in the network. It will be the function of the searcher to fill in the rule-boxes which will bring these words' ports together into constraint structure that produces a single coherent semantics. Thus, the forward search process originates at a state defined by a constraint propagation network with a list of ports and no rule-boxes. Each transition will add a rulebox to the network. Adding this rulebox will also have the side-effect of adding a new port to the network containing the result of the rulebox's CCG rule reduction.

The backward, or generation, process starts with the non-linguistic input: a semantic structure to be expressed[4]. This semantic structure occupies a port in an otherwise empty constraint propagation network. It will be the function of the searcher to fill in the rule-boxes which will divide up the semantics into units that can be associated with words. Thus, the backward search process originates at a state defined by a constraint propagation network with a single port and no rule-boxes. Each transition will add a rulebox to the network. Adding this rulebox will also have the side-effect of adding new ports to the network, one for each source term in the CCG rule.

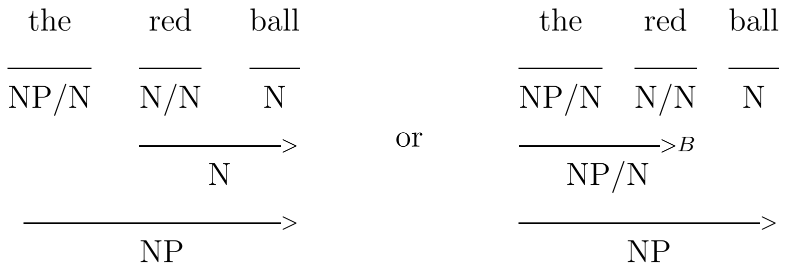
Thus we have that states in the search space are constraint network topologies, while transitions record the addition of a rulebox to the the constraint network.

4.5.3 Priming Keys

Now that we have defined the transitions for the search space, we proceed to define the priming keys for these transitions. Recall that identical priming keys should be produced by semantically equivalent transitions and by reciprocal transitions. This still leaves us to define what makes two transitions semantically equivalent in our search space.

[4] Actually, the non-linguistic input may also include a set of acceptable syntactic categories for the phrase to be generated. For example, if the syntactic category S is specified, then the generated expression will be a sentence.

Because each transition simply adds to the constraint propagation network a rule-box embodying a CCG rule, we find that determining the semantic equivalence of two transitions reduces to determining the semantic equivalence of the two CCG rule applications. This is not very straightforward, because CCG allows the same linguistic string to be parsed in multiple equivalent ways. For example, “The red ball” supports both of the following derivations:



These derivations are equivalent, despite the fact that the constituents are combined in a different order, that the derivation tree is structurally different, and that the second version uses the functional composition rule while the first version uses only functional application. Ideally, we would like to choose a method of generating priming keys that results in the same keys being generated for both of these derivations. We can achieve this ideal by noticing what is common between these two derivations. Both syntactically and semantically, CCG rules describe how arguments get applied to functions. Therefore, appropriate priming keys should be produced by monitoring the source of a syntactic or semantic fragment and its destination once it gets used as as the argument of a function.

4.5.4 Annotating Syntactic Categories

First we look at generating priming keys from syntactic category function application. In order to track the origins of syntactic categories and their function arguments, we annotate syntactic categories with identifiers indicating their origins. The exact nature of these identifiers is not important; rather, it is expected that the system components that gave to the reconciliation inputs will provide identifiers that are most appropriate to the input modality. For example, if linguistic input was extracted from an audio stream, the identifier for a word might specify a segment of the audio stream and word it was interpreted as, such as "AudioStream.(sample43234-sample72342, the)". For the remainder of this explanation, such an identifier will be abbreviated as "AS.the".

With these identifiers in hand, we can proceed to annotate the syntactic category. We declare two types of annotations:

- *originArgument annotation*: Every syntactic category argument has an originArgument annotation which holds the identifier of the word in which the argument was first specified.

- *originHead annotation*: Each element of the syntactic category (either a primitive such as "N" or a curried functional frame such as (S\NP)) which is not the argument of another syntactic category is annotated with an originHead annotation holding the identifier of the word in which that category was first specified.

Thus, the annotated syntactic category of words in our example phrase could be:

$$\begin{array}{rcl}
\text{the} & := & \text{NP}\{\text{originHead=AS.the}\} \ / \ \text{N}\{\text{originArgument=AS.the}\} \\
\text{red} & := & \text{N}\{\text{originHead=AS.red}\} \ / \ \text{N}\{\text{originArgument=AS.red}\} \\
\text{ball} & := & \text{N}\{\text{originHead=AS.ball}\}
\end{array}$$

Now that syntactic categories are annotated, we need to modify the constraints in the constraint propagation network to maintain these annotations. The main change is that whenever a metavariable from a CCG pattern binds to different instances of a syntactic category, the annotations for those instances get merged in the bindings. This has the effect of propagating the annotations through the network. For example, consider the two derivations we considered before. Abbreviating "originArgument" as *arg* and "originHead" as *head*, the annotated versions of these derivations are:

$$\begin{array}{ccc}
\text{the} & \text{red} & \text{ball} \\
\hline
\text{NP\{head=AS.the\}} & \text{N\{head=AS.red\}} & \text{N\{head=AS.ball\}} \\
\text{/ N\{arg=AS.the\}} & \text{/ N\{arg=AS.red\}} & \\
\cline{2-3}
 & \multicolumn{2}{r}{>} \\
 & \multicolumn{2}{c}{\text{[}\textbf{join arg=AS.red head=AS.ball}\text{]}} \\
 & \multicolumn{2}{c}{\text{N\{head=AS.red\}}} \\
\hline
\multicolumn{3}{r}{>} \\
\multicolumn{3}{c}{\text{[}\textbf{join arg=AS.the head=AS.red}\text{]}} \\
\multicolumn{3}{c}{\text{NP\{head=AS.the\}}}
\end{array}$$

and

$$\begin{array}{ccc}
\text{the} & \text{red} & \text{ball} \\
\hline
\text{NP\{head=AS.the\}} & \text{N\{head=AS.red\}} & \text{N\{head=AS.ball\}} \\
\text{/ N\{arg=AS.the\}} & \text{/ N\{arg=AS.red\}} & \\
\cline{1-2}
\multicolumn{2}{r}{>B} & \\
\multicolumn{2}{c}{\text{[}\textbf{join arg=AS.the head=AS.red}\text{]}} & \\
\multicolumn{2}{c}{\text{NP\{head=AS.the\}}} & \\
\multicolumn{2}{c}{\text{/ N\{arg=AS.red\}}} & \\
\hline
\multicolumn{3}{r}{>} \\
\multicolumn{3}{c}{\text{[}\textbf{join arg=AS.red head=AS.ball}\text{]}} \\
\multicolumn{3}{c}{\text{NP\{head=AS.the\}}}
\end{array}$$

Finally, we generate priming records whenever a CCG rule application fills a syntactic category parameter. The priming record indicates the originArgument of the syntactic category argument, as well as the originHead of the syntactic category head. In the above derivation, the priming records have been filled in, enclosed in square braces. Notice that in both derivations, the same set of priming records is generated.

4.5.5 Annotating Semantic Structures

Generating priming records from the application syntactic functions is only half the story; we can also generate priming records from semantic structures. We will once again appeal to annotations in order to track the origin of each semantic frame. Furthermore, we will continue to use identifiers generated by the system supplying the non-linguistic semantics as the basis for our identifiers. Consider for example if the visual system had observed a red ball, and had assigned the object a unique identifier such as ”Vision.Object209”. Using “originSemantic” as our annotation key, the vision system might annotate the LCS structure representing the red ball as follows:

$$\left[\begin{array}{ll} \text{MODIFIER} & \left[\begin{array}{l} <\textit{Property}\ \text{RED}> \\ \{\text{originSemantic=Vision.Object209.property.red}\} \end{array}\right. \\ \multicolumn{2}{l}{<\textit{Thing}\ \text{BALL}>} \\ \multicolumn{2}{l}{\{\text{originSemantic=Vision.Object209}\}} \end{array}\right.$$

We can then proceed to compute priming keys based on semantic annotations by observing when applying CCG rule causes one annotated LCS frame to become the child of another annotated frame. Let us look at the sample derivation above, this time focusing on semantic annotations.

$$\begin{array}{ccc}
\text{the} & \text{red} & \text{ball} \\
\hline
\text{NP/N.noun} & \text{N/N.noun} & \text{N} \\
\left[\begin{array}{l} \text{x} \\ <\lambda_{noun}\ \text{x}> \end{array}\right. &
\left[\begin{array}{l} \left[\begin{array}{l} \text{n} \\ \left[\begin{array}{l} <\textit{Property}\ \text{RED}> \\ \{\text{sem=V.O209.property.red}\} \end{array}\right. \\ <\widehat{property}\ > \end{array}\right. \\ <\lambda_{noun}\ \text{n}> \end{array}\right. &
\left[\begin{array}{l} \left[\begin{array}{l} \text{MODIFIER p} \\ <\textit{Thing}\ \text{BALL}> \\ \{\text{sem=V.O209}\} \end{array}\right. \\ <\lambda_{property}\ \text{p}> \end{array}\right.
\end{array}$$

$$\xrightarrow{\hspace{20em}}>$$

[modifier V.O209 V.O209.property.red]

$$\begin{array}{c}
\text{N} \\
\left[\begin{array}{l} \text{MODIFIER} \quad \left[\begin{array}{l} <\textit{Property}\ \text{RED}> \\ \{\text{sem=V.O209.property.red}\} \end{array}\right. \\ <\textit{Thing}\ \text{BALL}> \\ \{\text{sem=V.O209}\} \end{array}\right.
\end{array}$$

$$\xrightarrow{\hspace{25em}}>$$

$$\begin{array}{c}
\text{NP} \\
\left[\begin{array}{l} \text{MODIFIER} \quad \left[\begin{array}{l} <\textit{Property}\ \text{RED}> \\ \{\text{sem=V.O209.property.red}\} \end{array}\right. \\ <\textit{Thing}\ \text{BALL}> \\ \{\text{sem=V.O209}\} \end{array}\right.
\end{array}$$

and

$$
\begin{array}{ccc}
\text{the} & \text{red} & \text{ball} \\
\hline
\mathrm{NP/N.noun} & \mathrm{N/N.noun} & \mathrm{N} \\
\left[\begin{array}{l} \mathrm{x} \\ <\lambda_{noun}\ \mathrm{x}> \end{array}\right. &
\left[\begin{array}{l} \left[\begin{array}{l} \mathrm{n} \\ \quad \left[\begin{array}{l} <\textit{Property}\ \mathrm{RED}> \\ \{\mathrm{sem{=}V.O209.property.red}\} \end{array}\right. \\ <\widehat{property}\ > \end{array}\right. \\ <\lambda_{noun}\ \mathrm{n}> \end{array}\right. &
\left[\begin{array}{l} \left[\begin{array}{l} \mathrm{MODIFIER\ p} \\ <\textit{Thing}\ \mathrm{BALL}> \\ \{\mathrm{sem{=}V.O209}\} \end{array}\right. \\ <\lambda_{property}\ \mathrm{p}> \end{array}\right.
\end{array}
$$

$$
\xrightarrow{\hspace{12em}}{}_{>B}
$$

$$
\begin{array}{c}
\mathrm{NP/N.noun} \\
\left[\begin{array}{l} \left[\begin{array}{l} \mathrm{n} \\ \quad \left[\begin{array}{l} <\textit{Property}\ \mathrm{sem{=}V.O209.property.red}> \\ \{\mathrm{RED}\} \end{array}\right. \\ <\widehat{property}\ > \end{array}\right. \\ <\lambda_{noun}\ \mathrm{n}> \end{array}\right.
\end{array}
$$

$$
\xrightarrow{\hspace{20em}}{}_{>}
$$

$$
\begin{array}{c}
[\textbf{modifier V.O209 V.O209.property.red}] \\
\mathrm{NP} \\
\left[\begin{array}{l} \mathrm{MODIFIER} \ \left[\begin{array}{l} <\textit{Property}\ \mathrm{RED}> \\ \{\mathrm{sem{=}V.O209.property.red}\} \end{array}\right. \\ <\textit{Thing}\ \mathrm{BALL}> \\ \{\mathrm{sem{=}V.O209}\} \end{array}\right.
\end{array}
$$

Once again, we observe that the same set of priming keys, namely the single key [modifier Vision.Object209 Vision.Object209.property.red], is generated for both derivations.

4.5.6 Closing the Loop

At this point, we have methods of generating both syntactic and semantic based priming keys. Furthermore, the priming keys we generate satisfy the requirement that semantically equivalent transitions generate identical priming keys.

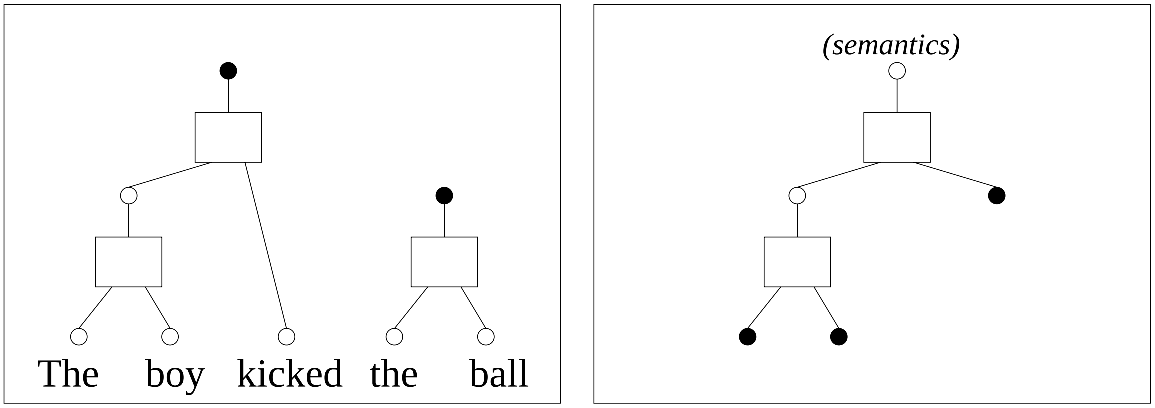


Figure 4-4: This figure highlights head ports and foot ports of a constraint propagation network, as used in the annotation transfer stage of priming key generation. The blocks represent constraints, and the circles represent ports. In the left frame, the darkened ports are the head ports, the sites of annotation transfer for forward search processes. In the right frame, the darkened ports are the foot ports, the sites of annotation transfer for backward search processes.

However, the astute reader will have noticed a shortcoming in the plan: the syntactic annotations originate at the linguistic source, while the semantic annotations originate at the non-linguistic source. As a result, only the syntactic priming keys will be available during forward (parsing) search, and only the semantic priming keys will be available during backward (generation) search. This is at opposition with the requirement that reciprocal transitions should generate identical priming keys. With no overlap between the forward and backward priming keys, Streams and Counterstreams completely falls apart. The solution is to make both syntactic and semantic priming keys available during both the forward and backward search phases.

To make the semantic priming keys available during forward search, we add an *annotation transfer* stage to the priming key generation process. During annotation transfer stage, the constraint network is scanned for *head ports*: ports which are not children of any rule port – that is, ports which are as close as possible to representing the complete semantics generated by the system, as shown in the left frame of Figure 4-4. Next, the annotation transfer process tries to align each semantic structure in each head port with some portion of the semantics from the non-linguistic source. When an alignment is found, the semantic annotations are copied from the non-linguistic semantic structure to the aligned semantic structure from the head port. Finally, these annotations are propagated back through the constraint network, and are thus available for priming key generation.

To make the syntactic priming keys available during backward search, we similarly add an annotation transfer stage during the backward search. This time, we transfer annotations at *foot ports*: ports which are not the parent of any rule port – that is, ports which are as close as possible to representing individual words, as shown in the right frame of Figure 4-4. For each foot port, we try to match its syntactic category/semantic pair against those of each the words in the linguistic input. When a match is made, annotations are copied over, propagated through the constraint network, and once again are available for priming key generation.

Chapter 5

Contributions

5.1 An Implemented Reconciliation System

The previous chapters have outlined an architecture for performing bidirectional reconciliation between the linguistic and non-linguistic semantic domains, in the context of a Cognitively Complete System such as the Bridge System. As part of this research, I have also implemented the system described throughout this thesis. Thus, this thesis stands not only as a proposed architecture for reconciliation, but also as documentation for a system I have successfully constructed. By implementing the architecture described here, I both validate the tenability of the architecture and establish a platform for future experimentation.

The architecture in this thesis brings together many subsystems and techniques to achieve the goal of reconciliation. The result is a system that can act as an ambiguity resolving constraint between linguistic and non-linguistic inputs. For example, if the system is presented with the phrase "The orange rolls," and a non-linguistic input (perhaps from a vision subsystem) describing the presence of bread (rolls) in the scene, the system will automatically select the interpretation of the the linguistic input describing pieces of bread of orange color. Thus, ambiguity is resolved in the linguistic domain, and the statement that the rolls have the property *orange* can be propagated to the non-linguistic semantic domain.

5.2 Contributions to AI and Related Fields

In pursuing the research in this thesis, several contributions have been made to artificial intelligence and related fields of research.

A general contribution to Bridge Project is made by creating the system's *first true reconciliation pathway.* One of the primary goals of the Bridge project is to investigate how information may be shared between different modules and representations in a manner beneficial to the understanding of all the participating modules. Thus, the current research advances the goals of the bridge project by providing a model for how other reconciliatory connectors could be created for mutually beneficial information exchange.

A specific contribution to Bridge Project is made by *implementing a connection between the linguistic domain and non-linguistic semantic domain.* The Bridge System requires continuous pathways between many different domains and processes in order to reach its full functionality as a Cognitively Complete System. To this end, the reconciliatory parser implemented as part of this research makes an important contribution to Bridge by completing the pathway between two subsystems.

A contribution directly to the field of artificial intelligence is made by *implementing Streams and Counterstreams.* The Streams and Counterstreams model of bidirectional search has the potential to be a very powerful tool for artificial intelligence researchers. However, it is difficult to find implementations of the model outside of the simple test system in Ullman's original paper. Therefore, this research makes a contribution to the field by providing an implementation of the Streams and Counterstreams model which can be useful either directly as a modular drop-in solution, or indirectly as a sample for how the model could be implemented.

A contribution directly to the field of artificial intelligence is made by *applying Streams and Counterstreams outside the domain of visual processing.* As implementations of Streams and Counterstreams have been difficult or impossible to find, the search model has not been applied to problem domains outside of Ullman's own vision research, and even there its application seems to have been limited. By applying the

Streams and Counterstreams model to the novel problem domain of bidirectional natural language parsing, the flexibility and usefulness of the model is further examined, and a case study for applying to model to complex problem domains is generated.

A contribution to the field of natural language processing is made by *extending Combinatory Categorial Grammar with Label-Selective Lambda Calculus* so that it may more naturally support semantic representations such as Lexical Conceptual Structures, in which adjunct semantics may be embedded under the head semantics, such as embedding a property's semantics inside the thing it modifies, rather than vice versa.

A contribution to the fields of computational linguistics and cognitive science is made by providing a critical component in creating *a test bed for further research involving language in cognitively complete situations.* Currently, experimental investigations into word learning models and language acquisition models are hindered by the lack of a system to present the learner with matching linguistic and world knowledge representations. By creating a cognitively complete system, the Bridge System should help enable such research. Furthermore, by providing a strong and flexible natural language parsing and generation module, this research lays the foundation for inserting word learning models [2, 10, 15] and language acquisition models [19, 7] into the Bridge system.

Appendix A

Tables

Table A.1: The complete space of Lexical Conceptual Semantics Expressions [9, 1]. See notes in Section 3.1.3 for discussion of variations from Jackendoff's original specification.

<Thing>	←	[<*Thing* x>
<Place>	←	[<*Place* Place> where Place ∈ {HERE, THERE, . . . }
	←	[<Thing> <*Place* PlaceFunc> where PlaceFunc ∈ {AT, ON, IN, ABOVE, BELOW, BEHIND, . . . }
	←	[<Path> <*Place* ON>
<Path>	←	[<PathElement>* <*Path* > where <PathElement>* indicates any number of PathElements
<PathElement>	←	[<*PathElement* PathElement> where PathElement ∈ {UPWARD, DOWNWARD, EASTWARD, WESTWARD, . . . }
	←	[<Thing> <*PathElement* PathFunction> where PathFunction ∈ (TO, FROM, TOWARD, AWAY-FROM, VIA, ALONG, . . .)
<Event>	←	[<Thing> <Path> <*Event* Go>
	←	[<Thing> <Place> <*Event* Stay>
	←	[<Thing> <Event> <*Event* Cause>
	←	[<Event> <Event> <*Event* Cause>
	←	[<Thing> <Event> <*Event* Let>

(continued on next page)

Table A.2: Lexical Conceptual Semantics Expressions, continued

$$\begin{array}{lcl}
\langle\textsc{State}\rangle & \leftarrow & \left[\begin{array}{l} \ \langle\textsc{Thing}\rangle \\ \ \langle\textsc{Place}\rangle \\ \langle\textit{State}\ \text{Be}\rangle \end{array}\right. \\
 & \leftarrow & \left[\begin{array}{l} \ \langle\textsc{Thing}\rangle \\ \ \langle\textsc{Path}\rangle \\ \langle\textit{State}\ \text{Orient}\rangle \end{array}\right. \\
 & \leftarrow & \left[\begin{array}{l} \ \langle\textsc{Thing}\rangle \\ \ \langle\textsc{Path}\rangle \\ \langle\textit{State}\ \text{Extend}\rangle \end{array}\right.
\end{array}$$

Table A.3: The complete label-selective lambda calculus, as presented in [5].

Definitions

$FV(M)$ is the set of free variables in M.

$[N/x]M$ is the result of replacing all free occurrences of x with N in M (or an appropriate α-renaming thereof).

Substitutions

$$\begin{array}{lcl}
[N/x]x & = & N \\
[N/x]y & = & y \quad \text{if } y \text{ is a variable and } y \neq x \\
[N/x](M_1{}_{\widehat{p}}M_2) & = & ([N/x]M_1){}_{\widehat{p}}([N/x]M_2) \\
[N/x](\lambda_p x.M) & = & \lambda_p x.M \\
[N/x](\lambda_p y.M) & = & \lambda_p y.[N/x]M \\
 & & \qquad \text{if } y \neq x \text{ and } y \notin FV(N) \\
[N/x](\lambda_p y.M) & = & \lambda_p z.[N/x][z/y]M \\
 & & \qquad \text{if } y \neq x \text{ and } y \in FV(N) \\
 & & \qquad \text{and } z \notin FV(N) \cup FV(M)
\end{array}$$

Reductions

$$\begin{array}{llcll}
 & \beta\text{-reduction} & & & \\
(\beta) & (\lambda_p x.M){}_{\widehat{p}}N & \rightarrow & [N/x]M & \\
 & & & & \\
 & \text{Symbolic reordering} & & & \\
(1) & (\lambda_{am}x.\lambda_{bn}y.M & \rightarrow & \lambda_{bn}y.\lambda_{am}x.M & a > b \\
(2) & M{}_{\widehat{am}}N_1{}_{\widehat{bn}}N_2 & \rightarrow & M{}_{\widehat{bn}}N_2{}_{\widehat{am}}N_1 & a > b \\
(3) & (\lambda_{am}x.M){}_{\widehat{bn}}N & \rightarrow & \lambda_{am}x.(M{}_{\widehat{bn}}N) & a \neq b, x \in FV(N) \\
 & & & & \\
 & \text{Numeric reordering} & & & \\
(4) & (\lambda_{am}x.\lambda_{an}y.M & \rightarrow & \lambda_{an}y.\lambda_{am-1}x.M & m > n \\
(5) & M{}_{\widehat{am}}N_1{}_{\widehat{an}}N_2 & \rightarrow & M{}_{\widehat{an}}N_2{}_{\widehat{am-1}}N_1 & m > n \\
(6) & (\lambda_{am}x.M){}_{\widehat{an}}N & \rightarrow & \lambda_{am-1}x.(M{}_{\widehat{an}}N) & m > n, x \notin FV(N) \\
(7) & (\lambda_{am}x.M){}_{\widehat{an}}N & \rightarrow & \lambda_{am}x.(M{}_{\widehat{an-1}}N) & m < n, x \notin FV(N)
\end{array}$$

Bibliography

[1] John R. Bender. Connecting language and vision using a conceptual semantics. Master's thesis, Massachusetts Institute of Technology, 2001.

[2] Keith Bonawitz, Anthony Kim, and Seth Tardiff. An architecture for word learning using bidirectional multimodal structural alignment. In *Proceedings of the North American Chapter of the Association for Computational Linguistics 2003 Human Language Technology Conference (HLT-NAACL 2003, In Press)*, 2003.

[3] Lera Boroditsky. Metamorphic structuring: understanding time through spatial metaphors. *Cognition*, 75(1):1–28, 2000.

[4] Bonnie Dorr. The use of lexical semantics in interlingual machine translation. *Machine Translation*, 7(3):135–193, 1992.

[5] Jacques Garrigue. *Label-Selective Lambda-Calculi and Tranformation Calculi.* PhD thesis, University of Tokyo, 1994.

[6] Dedre Gentner and Arthur B. Markman. Structure mapping in analogy and similarity. *American Psychologist*, 52(1):45–56, 1997.

[7] Edward Gibson and Kenneth Wexler. Triggers. *Linguistic Inquiry*, 25(3):407–454, 1994.

[8] Berthold Klaus Paul Horn. *Robot Vision.* McGraw-Hill, New York, New York, 1986.

[9] Ray Jackendoff. *Semantics and Cognition*, volume 8 of *Current Studies in Linguistics Series*. MIT Press, Cambridge, Massachusetts, 1983.

[10] Sourabh Niyogi. Bayesian learning at the syntax-semantics interface. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society (CogSci2002)*, 2002.

[11] Vilayanur S. Ramachandran and Edward M. Hubbard. Synaesthesia – a window into perception, thought and language. *Journal of Consciousness Studies*, 8(12):3–34, 2001.

[12] Jeffrey Mark Siskind. Acquiring core meanings of words, representated as jackendoff-style conceptual structures, from correlated streams of linguistic and non-linguistic input. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL-1990)*, 1990.

[13] Mark Steedman. A very short introduction to ccg (draft). 1998.

[14] Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, Massachusetts, 2000.

[15] J.B. Tenenbaum and F. Xu. Word learning as bayesian inference. In *Proceedings of the 22th Annual Conference of the Cognitive Science Society (CogSci2000)*, 2000.

[16] Shimon Ullman. *High Level Vision*, chapter 10, pages 317–358. Sequence Seeking and Counter Streams: A Model for Information Flow in the Visual Cortex. MIT Press, Cambridge, Massachusetts, 1996.

[17] R. M. Warren. Restoration of missing speech sounds. *Science*, 167:392–393, 1970.

[18] Patrick Winston. Bridge project memo. AI Memo 0, MIT Artificial Intelligence Laboratory, 2003.

[19] Charles D. Yang. *Knowledge and Learning in Natural Language*. PhD thesis, Massachusetts Institute of Technology, 2000.

[20] Deniz Yuret. Lexical attraction models of language. Submitted to *The Sixteenth National Conference on Artificial Intelligence*, 1999.