Incremental Natural Language Understanding with Combinatory Categorial Grammar

Mark McConville

Master of Science
School of Artificial Intelligence
Division of Informatics
University of Edinburgh
2001

Abstract

This thesis presents a model of incremental natural language understanding based on the grammatical formalism known as Combinatory Categorial Grammar. The model constitutes an integrated system involving a cyclical process of parsing, semantic adjudication and filtering. The motivating data for the model are the well-known observations about garden path effects in human sentence processing, and particularly the fact that the presence and strength of the garden path effect is influenced by the referential context in which the sentence is uttered, as well as the actual lexical items selected. It is argued that the model successfully explains certain garden path phenomena in English. The model has been implemented in the Java programming language.

Acknowledgements

I would like to thank the following people who have helped me at various points in this project: my supervisor Mark Steedman, Jason Baldridge, Julia Hockenmaier, Steve Clark, Johan Bos, and Beata Klebanov. Special mention is also due to all my colleagues on the MSc course this year. We've worked hard, learned a lot, and had a lot of fun together. This research was supported by an EPSRC Advanced Course Studentship.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mark McConville*)

Table of Contents

Chapter 1

Introduction

1.1 Garden Path Effects and Natural Language Understanding

Steedman (2000) claims that certain well-known experimental observations derived from the study of human sentence processing provide support for a model of natural language understanding with the following features:

- Natural language understanding is an *incremental* process, utterances being evaluated against, and integrated with, the listener's current knowledge base more or less one word at a time.
- Ambiguity can be resolved *mid-utterance*, swiftly and irrevocably, and not all possible analyses of a string are maintained until the end of the sentence.
- Ambiguity resolution is based, not on syntactic criteria, but solely on the semantic *sensibleness* of the competing analyses of the utterance so far.
- An utterance analysis is sensible to the extent that: (a) its referring expressions can be linked to appropriate referents in the current knowledge base; and (b) the events and states that it posits are consistent with the knowledge base.

The first claim, that natural language understanding is an incremental process, is uncontroversial. The literature contains a large amount of evidence that listeners have no problem understanding sentence fragments[1]. The remaining three claims are supported by observations about the relative unprocessability of so-called 'garden path' constructions, together with the

[1] For example, Marslen-Wilson and Tyler (1980), Tanenhaus et al. (1990)

fact that the degree of unprocessability of any specific utterance containing a potential garden path can be influenced by the context in which it is uttered and the plausibility of the states and events that it describes.

Garden path constructions were first brought to attention by Bever (1970). The following example is taken from Steedman (2000:238).

(1.1) The doctor sent for the patient arrived.

This sentence involves an ambiguity in the word *sent*, which can be, among others, a past tense finite intransitive verb taking a prepositional phrase complement, or the past participle of a transitive verb used as the head of a nominal postmodifier. When read in isolation, this sentence is difficult to process; most readers misresolve the ambiguity, choosing the finite verb analysis and thus being 'led down the garden path'. The conclusion here is that this kind of ambiguity is resolved mid-sentence, extremely swiftly and irrevocably, and hence that not all analyses of an utterance are maintained until the end of a sentence. By the time the reader reaches the actual finite verb *arrived*, the past participle reading of *sent* has already been discarded and cannot be recovered.

Ambiguities such as that in (1.1) are termed *attachment ambiguities* by linguists working within the constituent structure tradition, since they are manifested in the syntactic tree structure with the relevant subtree being attached to different nodes. Initially, such examples led researchers to claim that attachment ambiguities are resolved by structural criteria, such as the *Minimal Attachment Principle* in Frazier (1978), where the 'simplest' syntactic tree structure is preferred. In (1.1) the simplest structure for the fragment *the doctor sent* is claimed to be one where the verb *sent* is immediately dominated by the S node rather than the subject NP node, and thus the garden path effect is correctly predicted[2].

However, such a view leads to two erroneous conclusions: (a) sentence (1.1) will always give rise to a garden path effect, no matter what context it is uttered in; (b) sentences of an identical syntactic structure to (1.1), but containing different lexical items, will always give rise to a garden path effect. Experiments reported in Crain and Steedman (1985) and Altmann and Steedman (1988) provide evidence against these conclusions, showing that the presence and strength of the garden path effect can be influenced by referential context and semantic plausibility.

[2]Other syntactic approaches to structural ambiguity resolution are Fodor et al. (1974), Kimball (1973), and Marcus (1980).

For example, in a referential context where there is more than one doctor, and where the fact is known that one of the doctors was summoned to treat a patient, the sentence in (1.1) does not give rise to a strong garden path effect, as the following discourse makes clear.

(1.2) Two doctors were on duty in the hospital. A patient in the trauma ward started to exhibit symptoms of internal bleeding, so one of the doctors was summoned. The other doctor went to the cafeteria to get some coffee. The doctor sent for the patient arrived, but it was already too late.

But in a referential context containing only one doctor and where the fact that he or she was summoned for a patient is not known, the garden path effect is much stronger.

(1.3) A doctor and a nurse were on duty in the hospital. A patient in the trauma ward started to exhibit symptoms of internal bleeding, so one of them was summoned. The other one went to the cafeteria to get some coffee. The doctor sent for the patient arrived, but it was already too late.

The conclusion reached by Crain and Steedman is that (1.1) cannot be said to be intrinsically a garden path construction, since the strength of the garden path effect depends on the context in which it is uttered. The most that can be said is that there is a *potential* garden path effect.

The fact that the strength of the garden path effect in sentences like (1.1) depends on the actual lexical items chosen, rather than simply on the syntactic structure, is illustrated by contrasting (1.1) with the following sentence, again taken from Steedman (2000:238)[3].

(1.4) The flowers sent for the patient arrived.

This sentence is identical to that in (1.1) apart from the second word, the inanimate common noun *flowers* being substituted for the animate *doctor*. However, read in isolation, (1.1) gives rise to a strong garden path effect in the reader, whereas (1.4) does not. This is somewhat surprising — Crain and Steedman's *Principle of Parsimony*, which claims that the analysis which is most expensive to accommodate in the current referential context is discarded, would appear to predict that (1.4) should give rise to a strong garden path effect too, since it is cheaper to accommodate one bunch of flowers than two. Why then does the reader not misresolve the ambiguity in (1.4)?

The reader appears to make use of a source of knowledge distinct from the referential context in processing (1.4), which encodes information about which events and states of affairs are

[3]Bever (1970) used the examples *The horse raced/pushed past the barn fell* to make the same point.

plausible and which are not. This knowledge source will contain general principles such as the fact that only animate entities can send for things and that flowers are not animate entities. The human sentence processor can use this knowledge to perform a simple inference to conclude that flowers cannot send for things, and therefore that the finite verb analysis of *sent* is the wrong one.

In conclusion, Crain, Steedman and Altmann claim that the relative unprocessability of garden path sentences such as (1.1), and the fact that the degree of unprocessability is influenced by both referential context and knowledge-based inferences about plausibility, support a model of natural language understanding where: (a) Utterances are evaluated against the listener's current knowledge base more or less one word at a time; (b) So-called attachment ambiguities are resolved mid-sentence without maintaining all possible analyses until the end of the sentence is reached; and (c) Decisions about which analysis is correct are based, not on the syntactic structure, but rather on the cost of accommodating the analysis into the referential context, and whether or not the states and events described are consistent with the reader's knowledge base.

1.2 The Strict Competence Hypothesis and Combinatory Categorial Grammar

Steedman (2000) goes on to make the following claims:

- The evidence about garden path effects presented in section 1.1 suggests that a model of human natural language understanding must make available semantic representations for sentence fragments such as *the flowers sent for*, which can then be evaluated for referential felicity and semantic plausibility.
- By taking a classical, bidirectional categorial grammar and adding operations of type-raising and functional composition, a grammatical formalism can be developed which provides enough derivational flexibility to license these fragments as constituents, which can then be assigned semantic representations by the rules of compositional semantics.
- This kind of generalised categorial grammar is compatible with a very simple incremental parsing model, and is thus superior to other grammatical formalisms which require the parser to be equipped with extra features designed to deliver partial semantic representations.

The discussion of garden path effects in section 1.1 included the observation that sentence (1.4), repeated as (1.5), does not give rise to a garden path effect in the reader.

(1.5) The flowers sent for the patient arrived.

Steedman suggests that this is due to the finite verb analysis of *sent* being discarded early in the processing of the utterance, definitely before the actual finite verb *arrived* is encountered, and probably immediately after the preposition *for*, under the hypothesis that disambiguation occurs immediately the disambiguating information becomes available. It is assumed that the process of natural language understanding is a modular one, involving at least two devices: (a) a *parser*, which constructs semantic representations for the input strings; and (b) an *interpreter*, which evaluates the representations constructed by the parser, against the processor's current knowledge base. Since the disambiguation of *sent* in (1.5) is assumed to take place immediately after the word *for* has been encountered, this means that the interpreter will have at least the following two semantic representations for the substring *the flowers sent for* to evaluate.

(1.6) $\lambda y.summon'y(def'(\lambda x.flowers'x))$

(1.7) $\lambda z\lambda P.P(def'(\lambda y.flowers'y \wedge send'zy\ sb'))$

The semantic representations in these examples are assumed to be expressions of some typed higher-order logic. Definite reference is expressed by the function def' of logical type $<<e,\ t>,\ e>$[4], where $def'P$ denotes the contextually unique member of set P, rather than assuming a generalised quantifier analysis of the definite article. Referring expressions with an indefinite denotation are represented by means of the individual constant sb', standing mnemonically for *somebody*. When the interpreter comes to evaluate these two semantic representations, the one in (1.6) will be judged to be implausible, since an event where a bunch of flowers summons something is inconsistent with our knowledge of what kinds of events are possible in the real world, and hence discarded.

In order for the semantic representations of a sentence fragment such as *the flowers sent for* to be evaluated in this way, the parser must somehow make these representations available to the interpreter. Unfortunately however, most theories of English grammar assume that sentences have a right-branching constituent structure. Thus, such sentence fragments as *the flowers sent for* are not seen as being constituents of the sentence and cannot be assigned a

[4] A function from sets to individuals

semantic representation in the usual way, by means of the rules of compositional semantics contained within the grammar.

One proposed solution to this problem is to enrich the parser so that, in addition to constructing semantic representations for the syntactic constituents licensed by the grammar, there is some additional means of generating partial representations for incomplete constituents[5]. Steedman proposes an alternative — a grammatical formalism that provides enough derivational flexibility to allow such fragments to be grammatically licensed syntactic constituents in their own right, complete with a compositionally derived semantic representation.

There are two reasons to support this latter approach. Firstly, a grammatical formalism which allows for non-standard syntactic constituents is compatible with the *Strict Competence Hypothesis* (Steedman (2000:228)) — the requirement that the only semantic representations constructed by the parser are those associated with constituents licensed by the competence grammar. The Strict Competence Hypothesis is in effect an extra dimension of adequacy which can be imposed upon grammatical theories in addition to the familiar dimensions of descriptive and explanatory adequacy, and which can be summarised as follows:

(1.8) If two grammars, A and B, are equivalent in terms of descriptive and explanatory adequacy, and A is compatible with a simpler parsing model than B, then grammar A is better than grammar B.

In other words, since it is suggested that the human sentence processor must be able to evaluate the semantic representations of sentence fragments such as *the flowers sent for*, a grammar which licenses these fragments as syntactic constituents will be better than one that does not. This is because the former will be compatible with a simpler parsing model, where no extra-grammatical apparatus is necessary in order to build semantic representations for incomplete constituents[6].

A second reason supporting a theory of grammar which permits a much freer notion of surface constituency is that such constituents are necessary even for a merely descriptively adequate account of English grammar. Some examples of this involve coordination of traditional non-constituents, as in the following examples (Steedman (1985), Dowty (1988)):

(1.9) John loves and Peter admires Mary.

(1.10) John gave Mary a book and Peter a bible.

[5] For example Stabler (1991), Pulman (1986).

[6] But see Schieber and Johnson (1993) for objections to this argument.

The grammatical formalism proposed by Steedman is *Combinatory Categorial Grammar* (CCG), a mildly context-sensitive generalisation of the categorial grammars of Ajdukiewicz (1935) and Bar-Hillel (1953). CCG supplements a standard bidirectional, application-only categorial grammar with three operations derived from the combinatory logic of Curry and Feys (1958). The first of these is *functional composition*, which allows two functional categories to combine partially. Functional composition can be seen as a recursive generalisation of functional application, defined as follows[7]:

(1.11) **Forward Functional Composition**

$X/Y \ Y/Z \Longrightarrow X/Z$ iff $X/Y \ Y \Longrightarrow X$

The second extra operation used in CCG is *type raising*, which has the effect of converting atomic or otherwise simpler categories into more complex functional categories[8]:

(1.12) **Forward Type Raising**

$X \Longrightarrow T/(T\backslash X)$

There is much more that can be said about these operations, as well as the third operation which is used in CCG — *functional substitution*. For example, there are interesting restrictions which must be placed on both category variables in the type raising rule in English, so that the grammar does not overgenerate. Also, it is generally assumed that the number of recursions allowed in functional composition must be restricted to some finite number, possible determined by the maximum valence in the lexicon. The interested reader should turn to Steedman (2000:ch.3) for more information. For the purpose of this project, the interesting thing about CCG is that it provides the possibility of deriving a grammatical sentence of English in many different ways, and thus licenses a much wider range of syntactic constituents than is apparent from the directionalities specified in the lexicon.

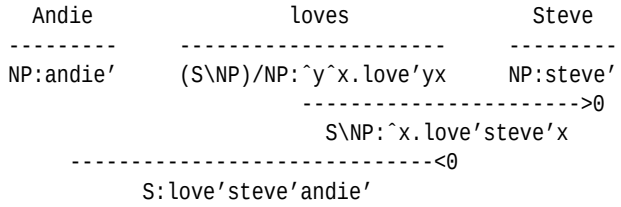
As an example of this, take the simple transitive sentence *Andie loves Steve*. A standard categorial grammar will be able to derive this sentence in only one way, assuming a lexicon where transitive verbs are assigned the category $(S\backslash NP)/NP$ and proper names are NPs[9]:

[7] This particularly elegant formulation of functional composition is adapted from Pareschi (1987). There is also a related backslash version of this rule.

[8] Again, there is also a related backslash version of this rule.

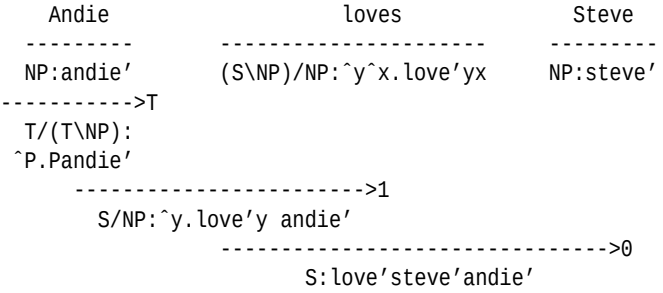
[9] An instance of forward and backward functional application in a derivation is symbolised by a line underneath the two constituent strings, annotated with > 0 or < 0 respectively. Categories include semantic representations, separated from syntactic categories by the colon operator. The carat symbol ^ is used to represent the λ operator.

(1.13)



Using type-raising and functional composition, on the other hand, the following derivation is also possible[10]:

(1.14)



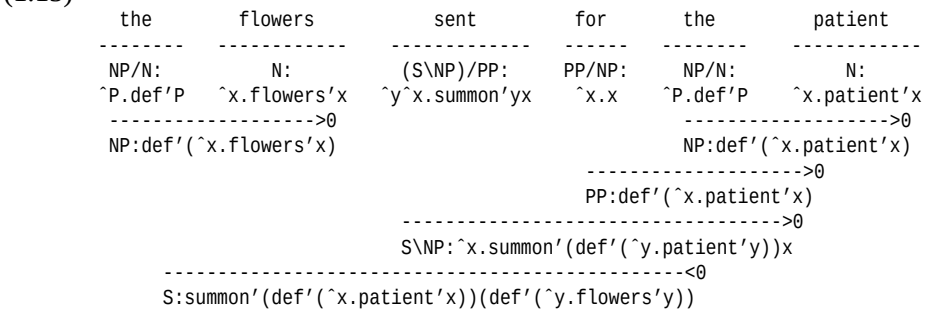
The derivation in (1.14) has the same end result as that in (1.13) — they both accept the sentence *Andie loves Steve* and assign it the semantic representation $love'steve'andie'$. However, the order of combination is different. Whereas in (1.13) the verb and direct object combine first, producing a right-branching derivation, in (1.14) it is the subject and verb which combine first, yielding a left-branching derivation.

Thus, in the left-branching derivation in (1.14) the string formed from the subject of the sentence and the finite verb is a constituent with a well-formed semantic representation. The capacity of CCGs to form such non-standard constituents, a side effect of the inclusion of type-raising and functional composition in the grammar, allows problematic syntactic phenomena such as right-node raising and object-extracting relative clauses to be captured in a monostratal grammar, without the need for non-monotonic operations of movement or deletion, traces or slash/gap features on verbal categories (citetsteedman85).

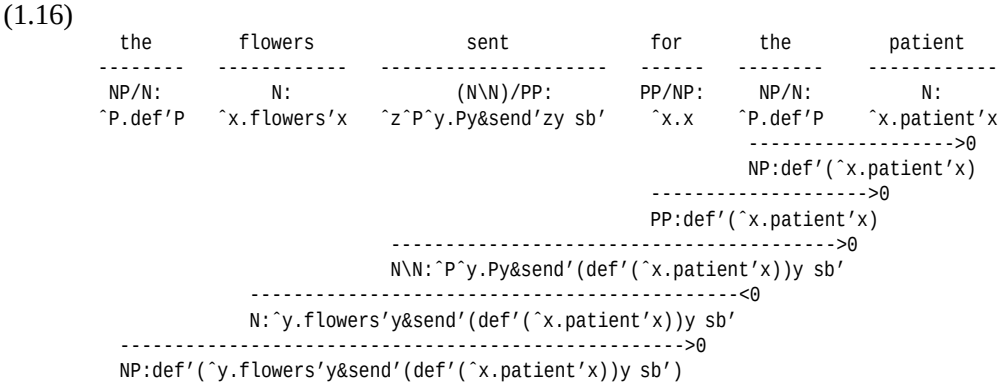
How, then, can CCG provide derivations for a string like *the flowers sent for the patient*, as both a sentence and a noun phrase, such that the substring *the flowers sent for* is a constituent? Standard right-branching derivations for this string, of the kind produced by an application-only categorial grammar with standard assumptions about the English lexicon, are as follows:

[10] An instance of forward functional composition is symbolised by the annotation $> n$, where n is the number of recursions applied.

(1.15)

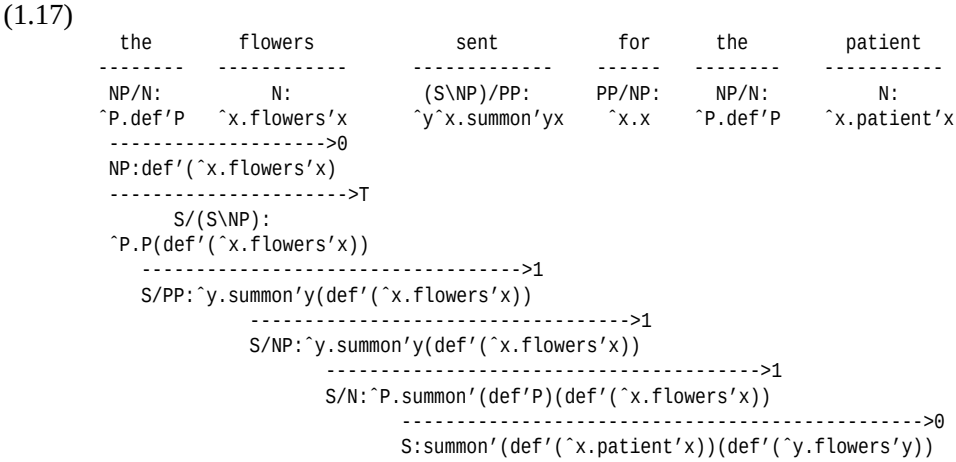


(1.16)



Note that the finite verb and past participle analyses of *sent* are assigned the categories $(S\backslash NP)/PP$ and $(N\backslash N)/PP$ respectively. By applying the rules of type-raising and functional composition permitted in CCG, both of the above derivations can be given a left-branching equivalent:

(1.17)



(1.18)

```
  the         flowers               sent                for        the         patient
--------    ------------     ---------------------     ------    --------    ------------
 NP/N:           N:               (N\N)/PP:           PP/NP:      NP/N:           N:
^P.def'P    ^x.flowers'x     ^z^P^y.Py&send'zy sb'     ^x.x     ^P.def'P    ^x.patient'x
          ---------------->T
              N/(N\N):
           ^Q.^x.Q(flowers')x
  -------------------->1
         NP/(N\N):
  ^Q.def'(^x.Qflowers'x)
     ------------------------------------->1
     NP/PP:^z.def'(^y.flowers'y&send'zy sb')
               ---------------------------------------->1
               NP/NP:^z.def'(^y.flowers'y&send'zy sb')
                    ------------------------------------------->1
                    NP/N:^P.def'(^y.flowers'y&send'(def'P)y sb')
                              ---------------------------------------------------->0
                              NP:def'(^y.flowers'y&send'(def'(^x.patient'x))y sb')
```

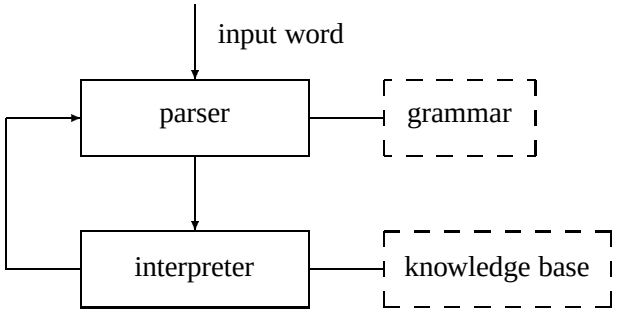
In both of these CCG derivations, the substring *the flowers sent for* is a constituent with a compositionally derived semantic representation. In (1.17) it has the syntactic category S/NP and the representation $\lambda y.summon'y(def'(\lambda x.flowers'x))$; in (1.18) the syntactic category is NP/NP and the semantic representation is $\lambda z\lambda P.P(def'(\lambda y.flowers'y \wedge send'zy\ sb'))$.

In conclusion, CCG is a grammatical formalism based on categorial grammar, which allows enough flexibility of derivation to license such non-standard surface structure constituents as *the flowers sent for*. This kind of constituent is claimed to be necessary to ensure a grammatical theory which can both explain garden path effects, by constructing and evaluating semantic representations for such sentence fragments, and which is compatible with the Strict Competence Hypothesis, which prefers grammars which require simpler parsing models.

1.3 Aims of the Project and Plan of the Thesis

The model of natural language understanding motivated by the observations in sections 1.1 and 1.2 can be summarised by the following diagram:

(1.19)



This model depicts natural language understanding as a *cyclical* process, where each new cycle is triggered by a single word entering the system. The natural language understanding model, or *processor*, consists of two devices operating in series: the parser, and the interpreter. The *parser* is a device which accepts as input a word from the assumed speaker. The grammar is assumed to consist of a lexicon and a set of combinatory rules. The parser consults the lexicon to find out the possible lexical categories and semantic representations for the input word, and then uses the combinatory rules to integrate these with the possible analyses of the string which has been processed so far. At the end of the parse phase, the parser will have constructed a set of semantic representations for the utterance up to and including the most recent input word.

The competence grammar consulted by the parser is assumed to be a CCG, consisting of a lexicon and a set of combinatory rules. The combinatory rules are general schemata for the combination of categories, such as functional application, functional composition and type raising. Using a CCG as the competence grammar has the advantages discussed in section 1.2 — since a majority of left-adjacent substrings of a sentence are grammatically licensed constituents with a compositionally derived semantic representation, a simple parsing model can be used to construct these incrementally, without requiring any extra-grammatical apparatus designed solely for the purpose of constructing partial representations.

After the parser has constructed the semantic representations for the utterance up to and including the current input word, the interpreter then must evaluate these representations against the current extra-linguistic knowledge base of the processor. I assume that this contains information of two kinds: (a) a *token database* containing representations of all the entities which are known to exist in the world; and (b) a *token database* containing all the facts which are known about the entities in the token database, as well as general facts about what kinds of entities can have what kinds of properties, or enter into what kinds of relations. Using the

information contained within the token and fact databases, the interpreter is able to identify those semantic representations which are referentially infelicitous or inconsistent with the way the processor believes the world to be. The interpreter can then instruct the processor to eliminate these analyses from consideration. In addition, the interpreter can add reference-semantic information to the semantic representations it processes, so that, for example, definite descriptions are annotated with the set of possible referents from the token database. As a side effect of the operation of the interpreter, the knowledge base may be updated with new information contained within the utterance — either new tokens can be added to the token database, or new facts to the fact database.

The model of incremental natural language understanding presented in (1.19) is a particularly simple one, with a view to a straightforward transparent implementation in an object-oriented programming language like Java. One further condition must be placed on the design of the interpreter module:

(1.20) The interpreter may not inspect or manipulate the syntactic information within an analysis.

This will ensure that there is no possibility of syntactic information being introduced by the back door into the process of ambiguity resolution.

The aim of this project is to elaborate and implement the model of natural language understanding presented in diagram (1.19), so as to develop a system which exhibits human-like garden-path behaviour as discussed in section 1.1. In particular, the system should model the way in which garden path effects in human readers are sensitive to changes in referential context (i.e. the token which exist and the properties they have) and the semantic plausibility of the states and events they describe. The model will be implemented as a collection of Java classes, and the implementation will aim for theoretical transparency rather than computational efficiency.

The system will reflect the modular nature of natural language understanding in a direct fashion, and to this end the parser and interpreter will be implemented independently. Chapters 2 and 3 of this thesis will deal with the implementation of the *parser*. Categorial grammars in general, and CCG in particular, are compatible with one of the simplest families of parser — the bottom-up shift-reduce parsers. Chapter 2 will introduce the basic non-deterministic shift-reduce model, and demonstrate why it is an inappropriate basis for a practical system. Chapter 3 will examine ways of eliminating the non-determinism inherent in the shift-reduce parser, with a view to constructing an efficient parser for practical use.

Chapter 4 will deal with the design and implementation of the interpreter. An algorithm will be developed which finds possible referents for referring expressions contained within semantic representations, and it will be shown how this algorithm interacts with a consistency checker so as to filter out implausible or infelicitous representations, and hence predict garden path effects. Finally, an integrated model of semantic plausibility checking (by means of a theorem prover) and reference filtering will be developed which successfully accounts for the garden path data dicussed in section 1.1.

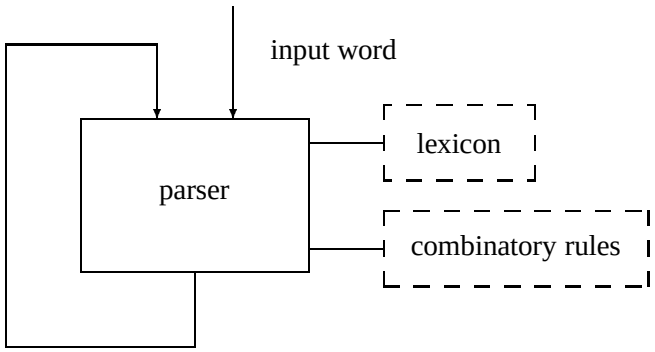
Chapter 5 will summarise and draw conclusions.

Chapter 2

Shift-Reduce Parsing with Combinatory Categorial Grammar

According to the model of incremental natural language understanding described and motivated in chapter 1, the parsing module can be represented by the following diagram, extracted and adapted from diagram (1.19).

(2.1)



The parser is defined as a cyclical device where each cycle is triggered by a new input word entering the system. The parser takes the input word, looks up its lexical categories and semantic representations in the lexicon, and then applies the combinatory rules so as to integrate these with the analyses of the previous part of the utterance.

This chapter will present a simple model of incremental parsing based on the CCG grammatical formalism presented in section 1.2. CCG is particularly well-suited for use with a

family of parsers known as the *bottom-up shift-reduce parsers*. This chapter will introduce the simplest member of this family, the *non-deterministic* shift-reduce parser, and illustrate the use of naïve breadth-first techniques for simulating the non-determinism, an approach which ensures completeness at the expense of efficiency.

2.1 Non-Deterministic Shift-Reduce Parsing with CCGs

Section 1.2 introduced the grammatical formalism CCG, which was argued to be a useful tool for developing incremental language processing systems which exhibit human-like behaviour when processing utterances containing a garden path construction. CCG allows a much greater flexibility of derivation than most other grammatical formalisms, and thus can provide left-branching derivations from a right-branching lexicon. This has the result that most left-adjacent substrings of an utterance can be regarded as constituents, and can thus receive suitable semantic representations. All that is needed to construct these representations from an input string is an extremely simple parser which can combine constituents one by one.

One of the simplest parsing models, and one which is extremely well-suited for use with categorial grammars, is the bottom-up *shift-reduce* parser. This parser pushes the input words one by one onto a stack, optionally reducing the top two elements of a stack into one element if there is a combinatory rule which licenses the reduction. The most characteristic member of the shift-reduce parser family, indeed its lowest common denominator, is the *non-deterministic* model. Recall that the parser has access to two sources of linguistic data: a set of combinatory rules, and a lexicon. I assume that the combinatory rules include forward and backward versions of both functional application and functional composition, that type raising is a lexical rule producing extra lexical categories for selected lexical items, and that the lexicon contains at least the following lexical entries.

(2.2) Andie :- $\{NP : andie',\ S/(S\backslash NP) : \lambda P.Pandie'\}$
saw :- $\{N : \lambda x.saw'x,\ (S\backslash NP)/NP : \lambda y\lambda x.see'yx\}$
Steve :- $\{NP : steve',\ S/(S\backslash NP) : \lambda P.Psteve'\}$

Note that the proper names *Andie* and *Steve* have one basic and one type raised lexical category, and that *saw* is ambiguous between a transitive verb (past tense of *see*) and a common noun (a tool for cutting wood). The parser also has access to a *workspace* on which to perform its computations. The workspace is conceptualised as a pushdown stack of elements where each element is a pair $\alpha : \beta$ where α is a category and β a semantic representation.

Imagine that the non-deterministic shift-reduce parser is processing the utterance *Andie saw Steve*, one word at a time. The parser reads the first word *Andie* and looks it up in the lexicon, where it finds a set of two potential entries. Being a non-deterministic device, it is guaranteed to always guess correctly at any decision point, so it chooses to add the subject type raised category $S/(S\backslash NP) : \lambda P.Pandie'$ to the workspace. Then, the parser reads the second word *saw*, looks it up in the lexicon, guesses which is the appropriate lexical entry (i.e. the verbal analysis), and pushes this onto the stack on the workspace. At this stage in the parse process, the workspace will look as follows.

(2.3)
```
--------------------
(S\NP)/NP:^y^x.see'yx
--------------------
S/(S\NP):^P.Pandie'
--------------------
```

The phase of the parse process where a new word is looked up in the lexicon and its chosen entry is pushed onto the stack already existing on the workspace is known as the *shift* phase. After every shift, the parser has another choice to make: either read the next input word and shift it, or attempt to *reduce* the top two elements on the current stack. Imagine that the parser decided to reduce the above stack at this point. It will apply the rule of forward functional composition, and the stack will be reduced to the following.

(2.4)
```
--------------------
S/NP:^y.see'y andie'
--------------------
```

After each reduce step, the parser must again choose whether to shift or to reduce again. Since the current stack in (2.4) cannot be further reduced, the only option is to shift the next word *Steve*. Assuming that it chooses the non-type-raised category, the stack will look like the following.

(2.5)
```
--------------------
      NP:steve'
--------------------
S/NP:^y.see'y andie'
--------------------
```

Next, the parser chooses to reduce the stack. The appropriate rule will be forward functional application, and the resulting stack is as follows.

(2.6)
```
-----------------
S:see'steve'andie'
-----------------
```

Now there are no more words to shift and the stack cannot be further reduced, so the parse has come to an end. The parser concludes that the input string *Andie loves Steve* is a sentence with the semantic representation $see'steve'andie'$. To sum up, the non-deterministic shift-reduce parser can be represented by the following algorithm[1].

(2.7) **A Non-Deterministic Shift-Reduce Parser**

```
Repeat until there are no more words to process:

  Either:  SHIFT:
           Look up the next word in the lexicon.
           Choose one of the word's lexical entries
             and push it onto the stack.

  Or:      REDUCE:
           Choose a combinatory rule
             and apply it to the top two elements on the stack.
```

This shift-reduce parsing model contains three distinct sources of non-determinism:

lexical ambiguity During the shift phase, when an input word has more than one lexical entry, the device must decide which to add to the stack.

shift-reduce conflicts At every cycle in the parse process, the device must choose whether to shift the next input word, or to attempt to reduce the top two elements on the stack.

rule choice During the reduce phase, when more than one combinatory rule may be used to reduce the top two elements on a stack, the device must decide which to apply.

In order to create a practical parsing system based on this non-deterministic model, two strategies may be followed. We can choose to simulate non-determinism by pursuing all possibilities in parallel in a breadth-first search through the parse states. Or we can attempt to eliminate the non-determinism altogether, by developing ways of making accurate guesses. Section 2.2 will discuss a parser which uses the former approach to simulate all of the three sources of non-determinism in algorithm (2.7). Chapter 3 will look at models which attempt to eliminate one or more of these sources of non-determinism, for example by consistently reducing a stack if at all possible.

[1] This algorithm is essentially the same as that in Steedman(2000:230).

2.2 Simulating Non-Determinism: A Naïve Breadth-First Shift-Reduce Parser

A naïve breadth-first shift-reduce parser maintains all possible parse alternatives running in parallel, so as to allow a simulation of backtracking should any particular sequence of decisions made by the non-deterministic parser fail. This involves three main alterations to the non-deterministic device summarised in algorithm (2.7) above:

- The workspace must be conceptualised as a *collection* of parse stacks, one for every distinct choice made in the parse process.
- Whenever a word is shifted onto the workspace, there must be one distinct parse stack for every possible combination of one of the word's lexical entries pushed onto one of the stacks from the previous cycle of the parse — if the shifted word has n entries and there are m stacks already on the workspace, then as a result of the shift phase there will be nm stacks on the workspace.
- After each shift phase there must be a reduce phase, involving every stack on the workspace being reduced as far as possible. Whenever two elements on top of a stack are about to be reduced according to a combinatory rule, a copy of the stack is first made and preserved so that there is always a stack on the workspace representing every distinct step of the parse process.

This parsing strategy can be summarised by the following algorithm.

(2.8) **A Naïve Breadth-First Shift-Reduce Parser**

```
Repeat until there are no more words to process:

  SHIFT:
  Look up the next word in the lexicon.
  Make sure there is a copy of the workspace
    for each of the word's lexical entries.
  Repeat once for each lexical entry:
    Push the entry onto each stack in its copy of the workspace.
  Merge all the copies of the workspace into one big workspace.

  REDUCE:
  Repeat for each stack on the workspace in turn:
    Repeat until the top two elements in the stack cannot reduce:
      Copy the stack.
      Reduce the top two elements of the copy
        by applying the relevant combinatory rule.
      If more than one combinatory rule is possible,
        then apply them all.
```

How would this parser go about parsing the same sentence *Andie saw Steve* from section 2.1? After the first two words *Andie saw* have been shifted onto the workspace, there will be four stacks under consideration, since each word has two lexical categories[2]:

(2.9)

```
                                                 -----------
                 ----------      -----------      (S\NP)/NP:
----------       N:^x.saw'x      (S\NP)/NP:       ^y^x.see'yx
N:^x.saw'x       ----------      ^y^x.see'yx      -----------
----------        S/(S\NP):      -----------       S/(S\NP):
   NP:a'           ^P.Pa'          NP:a'           ^P.Pa'
----------       ----------      -----------      -----------
```

The fourth stack on the workspace can now be copied and reduced, by forward functional composition, adding a new stack:

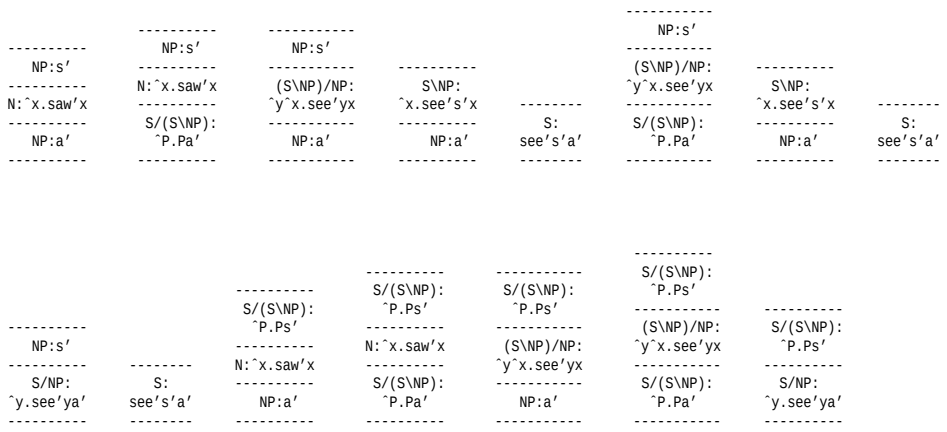
(2.10)

```
                                                  -----------
                 ----------      -----------      (S\NP)/NP:
----------       N:^x.saw'x      (S\NP)/NP:       ^y^x.see'yx
N:^x.saw'x       ----------      ^y^x.see'yx      -----------     ----------
----------        S/(S\NP):      -----------       S/(S\NP):        S/NP:
   NP:a'           ^P.Pa'          NP:a'           ^P.Pa'         ^y.see'ya'
----------       ----------      -----------      -----------     ----------
```

After the third word *Steve* has been shifted, the workspace will contain ten stacks. After reduction of each stack, the workspace will contain the following fifteen stacks:

[2]The logical constants *andie'* and *steve'* will henceforth be represented in semantic representations as *a'* and *s'* to save space.

(2.11)



No more words are available as input to the parser, therefore the parse is complete.

This naïve breadth-first shift reduce parser is *complete* in that it will always find a semantic representation for every grammatical string in the language, and if a string has more than one semantic representation then the parser is guaranteed to find all of them. However, when used in isolation as a simple parser, without any means of eliminating unviable parse stacks, the naïve breadth-first parser is far too inefficient to be practical. This is due to a feature of CCG often known as *spurious ambiguity* — the fact that most grammatical strings will have a plurality of potential derivations, many of them with identical semantic representations. The parser defined by the algorithm in (2.8) will find one successful parse for each potential derivation and since in CCG the number of distinct derivations of a string increases almost exponentially with the length of the string, this soon becomes unmanageable. This inefficiency is also evident in the number of parse stacks under consideration on the workspace after each word has been processed. When parsing the sentence *Andie saw Steve* there are two stacks after the first word, five stacks after the second word, and fifteen stacks after the third word has been parsed.

However, it should be noted that, in the context of the model of incremental natural language understanding sketched out in the introduction, the inefficiency of this parsing model is not necessarily fatal. Remember that the output of the parser after every word has been processed is then fed into an interpreter whose job it is to eliminate non-sensible analyses. If the output of the parser is assumed to be a collection of stacks of varying degrees of reduction, then it is possible to envisage an extended model of the interpreter which not only eliminates non-sensible *semantic representations* (i.e. those which posit implausible states or events or are referentially incompatible with the current token database), but also has some method for

eliminating non-sensible parse stacks. This kind of proposal is made by Niv (1993), who posits a module intermediate between the parser and interpreter, known as the *unviable state filter*, which discards stacks which the processor has learned from experience have little chance of being developed into a successful parse. The lesson here is that a naïve breadth-first shift-reduce parser could in principle be used in an efficient incremental language processing system, as long as the post-parser modules are powerful enough to eliminate all but a bounded number of alternative parse states.

The conclusions of chapter 2 are as follows:

- CCGs are particularly compatible with one of the simplest types of parsing algorithm — the non-deterministic shift-reduce parser, defined in algorithm (2.7).
- The non-deterministic shift-reduce parser contains three distinct sources of non-determinism: lexical ambiguity; shift-reduce conflicts; and rule choice.
- A naïve breadth-first shift-reduce parser, defined in algorithm (2.8), is a version of the shift-reduce parser which uses exhaustive breadth-first search techniques to simulate non-determinsism.
- The naïve breadth-first shift-reduce parser is complete, but highly inefficient, as the number of parse stacks on the workspace rises exponentially as each new input word is processed.

Chapter 3

Eliminating Non-Determinism: Reduce-First Parsing

Chapter 2 presented the model of non-deterministic shift-reduce parsing for use with CCGs, and showed how naïve breadth-first search techniques can be used to simulate non-determinism. The result is a complete but extremely inefficient parser, where the number of parse stacks being considered at any one moment increases apparently exponentially with the number of words processed since the start of the utterance.

This chapter will illustrate an alternative approach. Instead of merely *simulating* the non-determinism of the basic model, we can design a parser which systematically *eliminates* some or all of it. Recall the three sources of non-determinism present in the basic shift-reduce parser in algorithm (2.7):

- Lexical ambiguity.
- Shift-Reduce conflicts.
- Rule choice.

It is hard to see how a parser can be designed so as to eliminate the first of these. To do this, it would have to make a consistently accurate choice of which lexical category the input word should be associated with in its particular context of utterance, prior to pushing it onto the parse stack. This choice would have to be based solely on the elements already existing on the stack. Presumably some kind of statistical device trained on data (much like Niv's (1993) unviable state filter, except located as part of the parser itself) could be designed for this task,

but it is doubtful whether it would be accurate enough to always guess correctly. Indeed, there is no psycholinguistic evidence I am aware of that shows humans making garden path type mistakes that could be put down to a consistent wrong guess of lexical category.

The third source of non-determinism, rule choice conflict, is possibly more potential than actual for CCGs. In order for this to be a real problem, there would have to be some parse stack fulfilling two criteria: (a) the top two elements on the stack can be reduced by two different combinatory rules; and (b) this would have to give rise to a different semantic representation at the end of the utterance. I have not been able to find evidence of this kind of situation, so assume that non-determinism caused by rule choice conflicts is not a real problem for the efficiency of shift-reduce parsers. Thus, the non-determinism can safely be simulated, or the rules themselves arranged in some kind of order of precedence, assuming that only one rule can apply at a time[1].

This chapter will deal with the second source of non-determinism in the basic shift-reduce parsing model — the issue of shift-reduce conflicts. A model of shift-reduce parsing is introduced, which systematically eliminates this type of non-determinism by operating a strict reduce-first policy. This parser proves to be more efficient than the naïve breadth-first parser in algorithm (2.8), but is incomplete. Two different methods of restoring completeness will be contrasted. Finally, an even more ruthless version of the reduce-first parser will be discussed, which seems to offer something like constant efficiency.

3.1 A Simple Reduce-First Parser

The second source of non-determinism in the shift-reduce parser, the shift-reduce conflicts, differs from the others, in that: (a) it *is* a big problem for efficient parsing; and (b) it can be eliminated by a simple alteration to the parsing algorithm. A intuitively appealing way to reduce this type of non-determinism would be to consistently prioritise one of the two operations — if it is possible at some stage of the parse process to either shift a new word onto the stack, or to reduce the stack, then either always shift or always reduce. Of these two possibilities, the former is immediately unappealing, since this would mean the processor would have to wait until the entire sentence has been shifted before any of it could be reduced, and this would mean it wouldn't have access to the left-branching derivations required for truly incremental

[1]Indeed, the recursive definition of functional composition given in (1.11) would appear to support the ordering of rules, where progressively higher recursive versions of functional composition are attempted if functional application fails.

natural language understanding. The second approach, known as the reduce-first strategy is more promising.

The reduce-first parser can be summarised by the following algorithm[2].

(3.1) **A Reduce-First Parser**

```
Repeat until there are no more words to process:

  SHIFT:
  Look up the next word in the lexicon.
  Make sure there is a copy of the workspace
    for each of the word's lexical entries.
  Repeat once for each lexical entry:
    Push the entry onto each stack in its copy of the workspace.
  Merge all the copies of the workspace into one big workspace.

  REDUCE:
  Repeat for each stack on the workspace in turn:
    Repeat until the top two elements in the stack cannot reduce:
      Reduce the top two elements of the stack
        by applying the relevant combinatory rule.
```

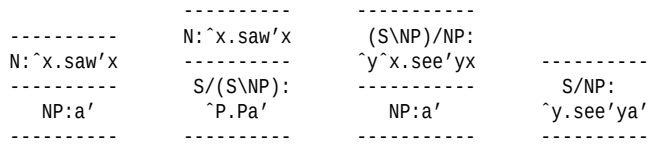
This parsing model is in certain respects similar to the naïve breadth-first model described in algorithm (2.8) in section 2.2. The shift phase ensures that all the alternatives based on distinct lexical categories of the most recent input word are, at least initially, pursued in parallel. The key difference, however, lies in the reduce phase. Whereas the naïve breadth-first parser always makes a copy of the relevant parse stack before reducing its top two elements, the reduce-first model never keeps copies. Once a reduction has taken place, there is no way for the parser to recover the two constituents that were combined during the reduction.

How would this parser go about parsing the utterance *Andie saw Steve*, again assuming the lexicon in (2.2)? After the two words *Andie saw* are shifted, the workspace will contain the same four stacks as in the naïve breadth-first parser trace in (2.9).

The reduce phase will then take each stack, one at a time, and attempt to reduce it as much as possible, without keeping any copies of the intervening stages. Only the fourth stack in (2.9) can reduce at this point, by means of forward functional composition, and the workspace at the end of the reduce phase will be the following.

[2]This algorithm is essentially that described in Niv (1994).

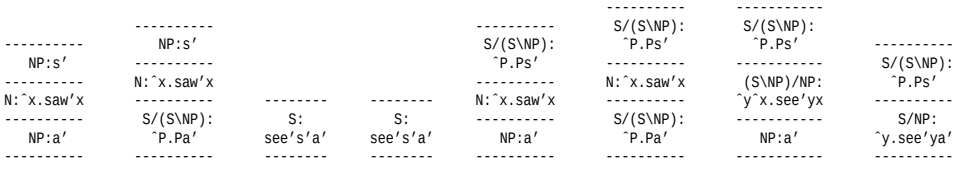
(3.2)



Note that while the naïve breadth-first parser had five stacks on the workspace in (2.10) after the two words *Andie saw* had been parsed, the reduce-first parser has only four.

The next step is to shift the third word *Steve* onto the workspace. It has two distinct lexical categories, and there are four stacks already on the workspace, so the new workspace will have eight separate parse stacks. After reduction, the workspace will be as follows:

(3.3)



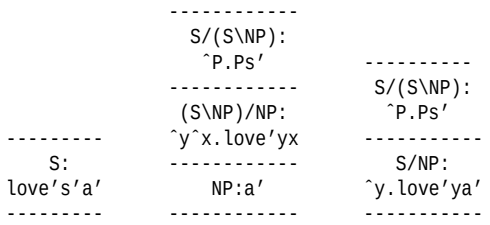
Note again that while the naïve breadth-first parser had fifteen stacks on the workspace in (2.11) after the three words *Andie saw Steve* had been parsed, the reduce-first parser has only eight.

How *efficient* is this reduce-first parser? On the one hand, it is more efficient than the naive breadth-first model from section 2.2, since it never stores copies of a stack before it is reduced, and is thus able to keep the number of alternative parse stacks on the workspace down to a more manageable level. In fact, after processing each word of the sentence *Andie saw Steve*, the reduce-first parser would have a workspace with two, four and eight stacks respectively, as against two, five and fifteen stacks for the naïve breadth-first shift-reduce parser. On the other hand, the trace of the parse process above shows that the reduce-first parser cannot be said to be particularly efficient, since many of the stacks on the workspace are never going to lead to a successful outcome. The problem is that, although the reduce-first parser as presented here is successful in eliminating non-determinism caused from shift-reduce conflicts (by prioritising the reduce operation) and from rule choice (by using the first successful rule which comes to hand), it makes no attempt to reduce non-determinism deriving from lexical ambiguity. In order to deal with this problem it would be necessary to use a purpose built device such as Niv's unviable state filter so as to eliminate unlikely stacks between parse cycles.

A more serious problem with the reduce-first parser is that it is *incomplete*, in the sense that it will only recognise utterances with a left-branching derivation. However, not all gram-

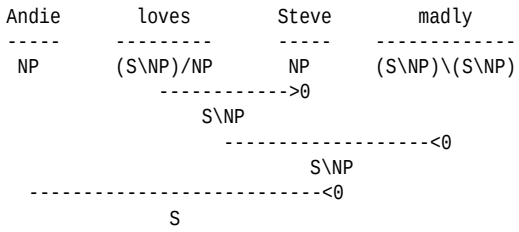
matical utterances in CCG can be derived in this way. Imagine, for example, that the parser is processing the sentence *Andie loves Steve madly*[3]. After shifting and reducing the first three words *Andie loves Steve* there will be three parse stacks on the workspace (with duplicates eliminated):

(3.4)



The next and final word to be shifted is *madly*, assumed to have the category of a VP post-modifier — $(S\backslash NP)\backslash(S\backslash NP)$. However, when shifted onto the workspace above, none of the stacks can be fully reduced and the parse will fail. Now, the utterance *Andie loves Steve madly* does have a derivation, represented as follows:

(3.5)



However, this derivation belongs to a class of derivations which cannot be found by the reduce-first parser. What these derivations have in common is that the string contains a word associated with a \ category, for example *madly* in the derivation above. In addition, the relevant \ category requires a right-branching analysis of the preceding substring in order to combine with it. However, the reduce-first parser effectively discriminates against right-branching analyses, always prioritising a left-branching analysis if one is available. This is a problem for any lexicon which includes any kind of post-modifier, whether relative clauses, VP-adverbs, or even VP coordination. The following sections discuss two different ways of overcoming this problem, by either modifying the reduce-first parser or expanding the lexicon.

3.1.1 Reduce-First Parsing with Derivation Rewrite Rules

One approach to overcoming the incompleteness problem in reduce-first parsing has its roots in Pareschi and Steedman's (1987) notion of *lazy parsing*. This assumes something akin to a

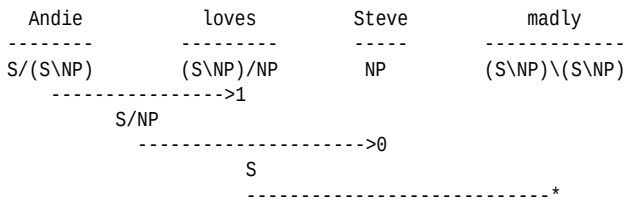
[3]This type of sentence is treated in Pareschi and Steedman (1987).

reduce-first parser, supplemented by a *reveal* operation, which applies whenever a \ category requires a right-branching derivation to its left in order for the parse to continue successfully. The reveal operation systematically converts left-branching derivations into right-branching ones, relying on the fact that the combinatory rules of functional application and functional composition, when conceptualised as 3-place relations, are functional in all three arguments. In other words, knowledge of any two of the left constituent, right constituent and result uniquely determines the third.

Pareschi and Steedman's approach was criticised by Hepple (1987) as having problems involving the soundness and completeness ,of the system. An improved version is to be found in Niv (1993), which presents a set of *derivation rewrite rules*, offering a sound and complete method of converting left-branching derivations into right-branching equivalents.

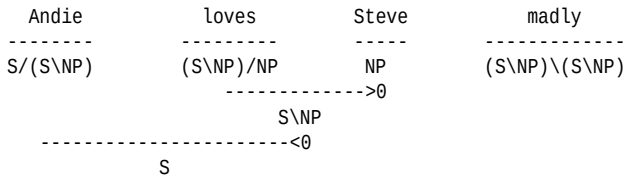
Niv's parser is a reduce-first parser, where lexical ambiguity is resolved by means of a post-parser unviable state filter, using statistical information learned from exposure to data to eliminate parse stacks which are unlikely to lead to a successful analysis. When the parser is processing the sentence *Andie loves Steve madly*, it initially pursues a reduce-first policy, preserving only the left-branching derivation, until the following state is reached, equivalent to the workspace in 3.4.

(3.6)



At this point, the derivation rewrite rules apply, so as to convert derivation (3.6) into a right-branching equivalent:

(3.7)



The right-adjacent constituents of *Andie loves Steve* are now visible to the parser, and it is a simple step to combine the VP-adverb *madly* with the rest. Note that this approach can presumably be simulated by a basic reduce-first parser operating on two distinct workspaces —

on one of the workspaces the full functionality of CCG can be applied, building left-branching analyses; on the other workspace only the rule of functional application can be used, resulting in right-branching analyses. If the left-branching, incremental parse breaks down at any point, the parser can use the other one as a backup.

If one wanted to implement Niv's derivation rewrite rules, the design of the parser must be more complex than the simple models introduced in previous sections. The alternative parse states in such a model would have to consist not simply of a stack of categories, but rather a complete derivation, including information about all previous combinations and the combinatory rules used to form them. In terms of the model defined above, this would be equivalent to expanding the elements in the stacks from pairs of the form $\alpha : \beta$, where α is a category and β is a semantic representation, to triples of the form $\alpha : \beta : \gamma$, where γ is the derivational history of the constituent, listing the two constituents it was formed from and the rule used.

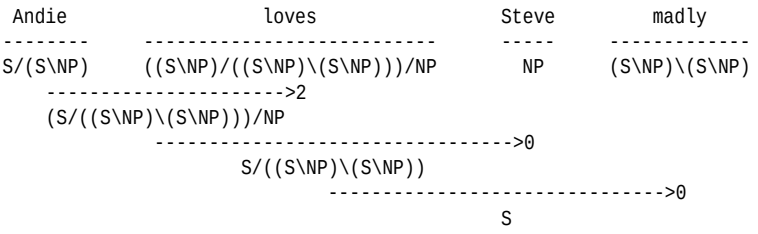
Thus, Niv's proposal for an incremental CCG parser, operating a reduce-first strategy but with a system of derivation rewrite rules to convert to right-branching derivations if necessary, is both complete and relatively efficient, at least compared to the naïve breadth-first shift-reduce parser from section 2.2. However, it requires the parser to be extended so as to allow for the representation and manipulation of entire derivations, complete with information about intermediate constituents and the rules used to form them.

3.1.2 Reduce-First Parsing with an Enriched Lexicon

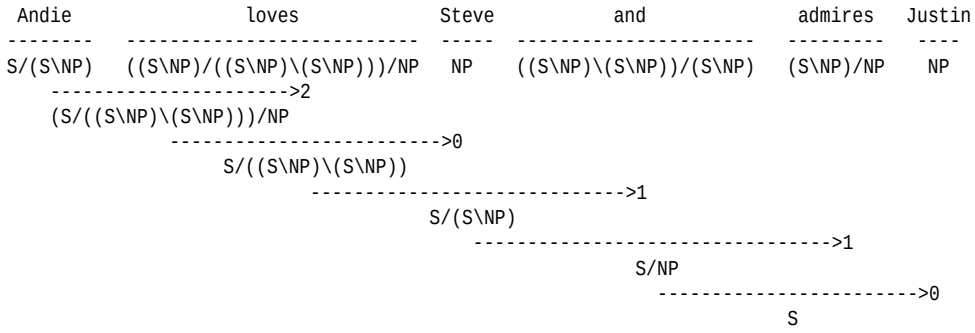
A second approach to parsing sentences like *Andie loves Steve madly*, without needing to complicate the parser at all, is to enrich the lexicon so as to ensure that this kind of sentence does indeed have an incremental left-branching derivation. Recall derivation (1.18) in section 1.2, where a type-raised category $N/(N\backslash N)$ for nouns was introduced, licensing a nominal post-modifer. This allowed a left-branching, incremental derivation for the noun phrase *the flowers sent for the patient*, where the substring *the flowers sent for* is a syntactic constituent.

Considering the problem of parsing sentences like *Andie loves Steve madly*, it appears reasonable to assume that, if nouns can optionally subcategorise for nominal postmodifiers, then verbs should also optionally subcategorise for verbal postmodifiers. So, in addition to the basic category $(S\backslash NP)/NP$, a transitive verb like *loves* can be associated with the VP-modifier category $((S\backslash NP)/((S\backslash NP)\backslash(S\backslash NP)))/NP$, which combines with the direct object, then with a postmodifier, and finally with the subject. Using this new category for transitive verbs, the

sentence *Andie loves Steve madly* can be derived in the following way.

(3.8)


Note here that using the lexical category $((S\backslash NP)/((S\backslash NP)\backslash(S\backslash NP)))/NP$ for transitive verbs allows a fully incremental, left-branching derivation of the sentence *Andie loves John madly*, compatible with an incremental reduce-first parser. This VP-postmodifier category for the verb *loves* can also be used to provide maximally left-branching derivations for VP-coordination sentences, assuming that coordinating conjunctions are normal lexical entries of the form $(X\backslash X)/X$, rather than being introduced syncategorematically:

(3.9)


In order to deal with the full range of verbal postmodifier and coordination phenomena in English in this manner, it is necessary to posit a full range of extra verbal categories, one for each different kind of verbal postmodifier. One of the disadvantages of this kind of treatment is the increase in the size of the lexicon, and the corresponding increase in non-determinism deriving from lexical ambiguity. Note that this is only a problem for parsing efficiency insofar as the parser does not have any means of eliminating lexical ambiguity. The next section will present a version of the shift-reduce parser which can do this in a particularly ruthless manner.

Another problem with this approach lies in the relationship between the competence grammar and the processor. Ideally the competence grammar should be just that — a pure reflection of linguistic competence, independent of any influence from the processor. If it is permitted to add extra lexical categories simply to allow for the possibility of incremental parsing, when the competence grammar already has the means of generating the strings, then this will surely have implications for the transparency of the link between grammar and processor.

However, from a processing perspective the lexical approach to incremental parsing of postmodified categories (including coordination) does have advantages. The parser can be kept as simple as possible, without needing any of the extra features posited by Niv's system. Thus, since the only part of the system that needs to be changed to allow for incremental parsing is the lexicon, and since the alteration to the lexicon is one of quantity not quality, the system proposed here is in many ways more economical than any approach which adds extra modules to the processor or extends the functionality of an existing module.

3.2 A Ruthless Reduce-First Parser

The reduce-first parser introduced in section 3.1 is able to eliminate non-determinism caused by shift-reduce ambiguities, and thus form the basis for a reasonably efficient practical parsing system. However, it makes no attempt at resolving lexical ambiguity. Parse stacks corresponding to every combination of possible lexical categories for each word in the utterance are pursued in parallel throughout the parsing process. This means that the number of stacks being processed will grow constantly as each new ambiguous word is parsed, and the device will have more work to do as the utterance gets longer. Of course, it *is* possible that the interpreter is solely responsible for resolving lexical ambiguity. However, there is another restriction which can be placed upon the reduce-first parsing model which will fulfil the same function — to delete any parse stack which does not reduce after a new element is shifted onto it.

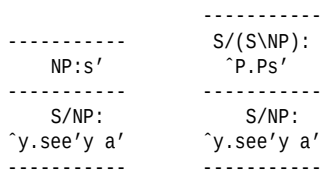
To take the same example utterance *Andie saw Steve* that we used above, this *ruthless* shift-reduce parser will function as follows. After the first two words *Andie saw* are shifted, there will be four stacks on the workspace, which will be identical to that in (2.9) above. The next step is to try and reduce each stack on the workspace. Only the fourth stack can reduce, so all the others are deleted, and the workspace will look as follows.

(3.10)

```
-----------
   S/NP:
^y.see'y a'
-----------
```

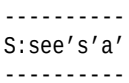
Next, the third and final word in the sentence *Andie saw Steve* is shifted onto the workspace. Since *Steve* has two distinct lexical categories, the resulting workspace consists of two stacks.

(3.11)



The parser then switches into reduce mode again. The second stack cannot reduce and is thus deleted. The first stack *can* reduce, by forward function application. Only one stack will remain on the workspace:

(3.12)



This ruthless reduce-first parser can be summarised by means of the following algorithm.

(3.13) **A Ruthless Reduce-First Parser**

```
Repeat until there are no more words to process:

  SHIFT:
  Look up the next word in the lexicon.
  Make sure there is a copy of the workspace
    for each of the word's lexical entries.
  Repeat once for each lexical entry:
    Push the entry onto each stack in its copy of the workspace.
  Merge all the copies of the workspace into one big workspace.

  REDUCE:
  Repeat for each stack on the workspace in turn:
    Reduce the top two elements of the stack
      by applying the relevant combinatory rule.
    If the stack cannot reduce then delete it from the workspace.
```

This shift-reduce parser deals with the problem of non-determinism deriving from lexical ambiguity in an extremely ruthless and powerful way. As well as operating the same reduce-first strategy as the parser in algorithm (3.1), the ruthless parser effectively eliminates any stack where the newly shifted lexical category fails to combine with the other element on the stack. One of the advantages of employing such a ruthless strategy lies in the efficiency of the parser. For those sentences which it does accept, it should process them in something like linear time. For example, in parsing the string *Andie saw Steve*, while the breadth-first parser ends up maintaining fifteen distinct analyses, and the reduce-first parser has eight (assuming

that equivalent stacks are not eliminated), the ruthless parser will only maintain a single parse stack. In this way the ruthless parser can eliminate lexical ambiguity without the need for an extra module such as Niv's (1993) unviable state filter.

However, the ruthless shift-reduce parser is even more incomplete than the reduce-first version. It will only recognise sentences with a *maximally left-branching* derivation, where every left-adjacent substring is a constituent. This will potentially place even more pressure on the lexicon to include categories which allow such maximally left-branching derivations for every grammatical sentence in the language.

Another advantage of the ruthless shift-reduce parsing strategy concerns the design of the interpreting module. A side effect of using this parser, which distinguishes it from the other shift-reduce parsers is that each stack left on the workspace after the reduce phase will only consist of a single element, with the result that the output of the parser can be seen as consisting of a set of labelled semantic representations, rather than a set of parse stacks. This leads to a comparably simpler conception of the interpreter. Instead of having to unpack parse stacks of varying sizes in order to determine the sensibleness of the various components, the interpreter can simply go through the set of semantic representations one by one and evaluate them.

3.3 Conclusion: Parsing with CCGs

This chapter has looked at ways of eliminating the non-determinism involved in shift-reduce parsing. A reduce-first parser was introduced, which eliminates shift-reduce conflicts by consistently prioritising the reduce operation over shift. This parser is more efficient than the naïve breadth-first shift-reduce parser, but suffers from incompleteness — it will only recognise sentences which have a left-branching derivation. Two methods of restoring completeness in reduce-first parsing were discussed.

Finally, a ruthless reduce-first parser was introduced, which reduces non-determinism deriving from lexical ambiguity by discarding any parse stacks which fail to reduce to a single element. Like the simple reduce-first parser, the ruthless model is incomplete, recognising only those sentences with maximally incremental derivations (i.e. where every left-adjacent substring is a constituent). However, it is extremely efficient, since the number of parse states under consideration appears to remain fairly constant, no matter how many words have been processed since the start of the sentence.

It was mentioned at the start of chapter 2 that CCGs are compatible with the family of bottom-up parsers in general. Now, the shift-reduce parsers are not the only members of this

family that have been used with CCG. Various proposals have been made that use a CKY, all paths bottom-up parser with CCGs or other flexible categorial grammars (Pareschi and Steedman (1987), Komagata (1999)). It has also been proposed that using a chart parser with beam-search might provide an explanation of garden path effects, provided that an apropriate method is found of ranking the alternatives (Steedman p.c). However, I follow Niv (1994) in restricting my attention to the shift-reduce family of bottom-up parsers, for two main reasons: (a) pushdown stacks are an inherently simpler type of data structure than charts, and do not require any kind if implicit or explicit indexing of the words in the sentence being processed; and (b) as pointed out in Niv (1994), chart parsing is inefficient for incremental analysis of sentences, since analyses of the entire prefix are awkward to compute from the developing chart.

The remainder of this thesis will be concerned with the way in which the semantic representations constructed by the parser are evaluated for referential felicity and semantic plausibility, so as to account for the garden path effects discussed in section 1.1. Especially important will be the way in which the relative strength of the garden path effect varies when the following two sentences, repeated from section 1.1, are uttered in different referential contexts.

(3.14) The doctor sent for the patient arrived.

(3.15) The flowers sent for the patient arrived.

I will henceforth assume the ruthless reduce-first parsing algorithm from (3.13), consulting the following lexicon.

(3.16) the :- $\{NP/N : \lambda P.def'P,\ (S/(S\backslash NP))/N : \lambda P\lambda Q.Q(def'P)\}$
doctor :- $\{N : \lambda x.doctor'x,\ N/(N\backslash N) : \lambda Q\lambda x.Q(doctor')x\}$
flowers :- $\{N : \lambda x.flowers'x,\ N/(N\backslash N) : \lambda Q\lambda x.Q(flowers')x\}$
patient :- $\{N : \lambda x.patient'x,\ N/(N\backslash N) : \lambda Q\lambda x.Q(patient')x\}$
sent :- $\{(S\backslash NP)/PP : \lambda y\lambda x.summon'yx,\ (N\backslash N)/PP : \lambda z\lambda P\lambda y.Py \wedge send'zy\ sb'\}$
for :- $\{PP/NP : \lambda x.x\}$
arrived :- $\{S\backslash NP : \lambda x.arrive'x\}$

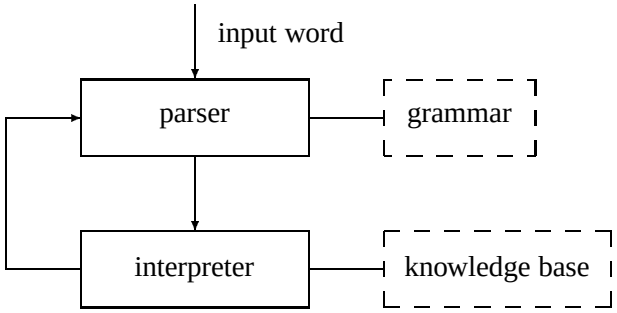
To provide a base point of comparison for the models of semantic evaluation to be presented in the following chapters, I present in Appendix A a trace of the ruthless reduce-first parser processing the garden path sentence in (3.14). This illustrates the operation of the parser with a null interpreter module. The trace for the sentence in (3.15) will be identical, the only difference being the substitution of the constant $flowers'$ for $doctor'$ in the semantic representations.

Chapter 4

Interpretation and Filtering

Recall the model of incremental natural language understanding presented in diagram (1.19) of section 1.3, repeated below.

(4.1)



Chapters 2 and 3 elaborated the design of the parser module. The competence grammar is assumed to be a CCG, where an analysis is a pair consisting of a category and a semantic representation. The lexicon which the parser can consult is a superset of the lexicon in (3.16), and the combinatory rules which it can use to reduce the parse stacks are forward and backward functional application, and forward and backward functional composition. The parser itself makes use of a workspace with slots for a number of alternative parse stacks to be pursued simultaneously. I assume the ruthless reduce-first parsing algorithm from (3.13), where shift-reduce conflicts are eliminated and lexical ambiguity restricted to those categories which can immediately combine with the parse stacks already on the workspace.

During each cycle, after the parsing module has completed its task, the workspace will contain a set of labelled semantic representations, one for each distinct analysis of the sentence

thus far. This collection serves as the input to the second module in (4.1), the interpreter. The role of the interpreter is twofold:

- To **integrate** the information contained within the semantic representations of the input string with the current extra-linguistic knowledge base of the processor, held in the token and fact databases.
- To **filter out** those analyses of the input string whose semantic representations are not sensible.

The integration of semantic representations with the knowledge base is assumed to involve two complementary transfers: (a) reference-semantic information is added to the referring expressions contained within the semantic representations, listing the entities in the token database which can serve as a referent for each referring expression[1]; (b) new information from the semantic representations must be added at some point to the knowledge base. The filtering process can discard analyses whose semantic representations are either referentially infelicitous (making assumptions which are inconsistent with the entities known to exist), or implausible (describing some state or event which is inconsistent with the way the world is known to be).

This chapter will be concerned with the design of the interpreter module. Of particular concern will be the means by which referential information is added to semantic representations by a device known as a *reference resolver*, and the criteria used by the interpreter to identify and eliminate referentially infelicitous analyses. When designing the interpreter module, condition (1.20) from section 1.3, repeated below, must be obeyed.

(4.2) The interpreter may not inspect or manipulate the syntactic information within an analysis.

This condition ensures that syntactic information contained within analyses remains invisible to the post-parser modules of the processor, maintaining the modularity and information encapsulation of the system.

Before starting our discussion of the interpreter module, it will help to clarify the criteria by which its success will be measured. The following three observations were made in the discussion of garden path sentences in chapter 1. It may be helpful to view these observations against the background of the ruthless reduce-first parse trace in Appendix A.

[1] See Haddock (1987), Mellish (1981)

Observation 1 When the string *the doctor sent for* is encountered in a referential context where either no doctors or exactly one doctor are known to exist, only the S/NP analysis can remain on the workspace after interpretation.

Observation 2 When the string *the doctor sent for* is encountered in a referential context where more than one doctor is known to exist, both the NP/NP and the $(S/(S\backslash NP))/NP$ analyses must remain on the workspace after interpretation.

Observation 3 When the string *the flowers sent for* is encountered in any referential context whatsoever, both the NP/NP and the $(S/(S\backslash NP))/NP$ analyses must remain on the workspace after interpretation, and the S/NP analysis can be discarded.

The ruthless reduce-first parser, when used on its own without any interpretation and filtering, maintains all grammatical analyses under the very end of the parse, exemplified in the trace in Appendix A. This chapter will examine ways of filtering out grammatical but non-sensible analyses so as to predict the behaviour described in these three observations.

4.1 A Model of Reference Resolution

One of the tasks which the interpreter module must perform is that of finding appropriate referents for referring expressions contained within the semantic representations. In order to do this, the device must consult some data structure containing a representation of the entities which are known to exist in the world and the properties which individuate them from other entities. In figure (4.1) this information was contained within the knowledge base, assumed to be divided into a token database and a fact database. In this chapter I will use simple discourse representation structures (Kamp and Reyle, 1993) to symbolise these databases. So, for example, in the following figure, the top half represents the token database and the bottom the fact database.

(4.3)

$e_1\ e_2\ e_3\ e_4$
$doctor'\,e_1$
$doctor'\,e_2$
$patient'\,e_3$
$send'\,e_3 e_1 e_4$

This figure represents the context in (1.2) from section 1.1[2]. Four entities are known to exist, two of these are doctors, one is a patient, and one of the doctors was summoned by the fourth entity.

The reference resolver performs the following steps:

- It scans the semantic representation from the bottom up, identifying referring expressions.
- Whenever it finds a definite description, in other words a term of the format $def'(\lambda x.\alpha)$ for any α, it will compare the condition α against the current knowledge state, identify the set of entities in the token database which satisfy α, and add this set γ to the definite description as a referential annotation — $def'_{\gamma}(\lambda x.\alpha)$.

It should be pointed out that semantic representations will almost certainly include other types of referring expression, for example, indefinite descriptions, pronouns, proper names etc. These are ignored by the algorithm in its present instantiation.

Imagine how this reference resolution device would interfere with the ruthless reduce-first processing of the garden path sentence *the doctor sent for the patient arrived*, represented in the trace in Appendix A, and assuming the knowledge base in (4.3).

Cycle 1 The first word *the* is shifted, and no reduction is possible. The reference resolver has two semantic representations to deal with: $\lambda P.def'P$ and $\lambda P\lambda Q.Q(def'P)$. Both of these contain the definite description $def'P$. The condition P is a variable, meaning that no individuating information has yet been presented to help the processor figure out what the expression refers to, so the semantic representations are left unchanged. The input to the second cycle of the processor is the same as for the basic unadorned parser.

Cycle 2 The second word *doctor* is shifted, all four stacks are reduced, and four semantic representations are sent to the reference resolver:

(a) $def'(\lambda x.doctor'x)$

(b) $\lambda P.P(def'(\lambda x.doctor'x))$

(c) $\lambda Q.def'(\lambda x.Q(doctor')x)$

(d) $\lambda Q\lambda P.P(def'(\lambda x.Q(doctor')x))$

[2]Certain entities and facts have been omitted for conciseness. Entities are represented as e_n

The first two include the definite description $def'(\lambda x.doctor'x)$. The reference resolver takes the condition $\lambda x.doctor'x$ and consults the knowledge state in (4.3) to identify the set of entities x which satisfy the condition $doctor'x$ i.e. $\{e_1, e_2\}$. This set is then added to the definite description as its referential annotation, so the first two semantic interpretations are changed to:

(a') $def'_{\{e_1,e_2\}}(\lambda x.doctor'x)$

(b') $\lambda P.P(def'_{\{e_1,e_2\}}(\lambda x.doctor'x))$

These annotated semantic representations contain the information that a definite description of the form *the doctor* has been processed and the processor has discovered two potential referents e_1 and e_2.

The third and fourth interpretations both contain a definite description of the form $def'(\lambda x.Q(doctor')x)$. This time the reference resolver takes the condition $\lambda x.Q(doctor')x$ and attempts to find individuals x which satisfy $Q(doctor')x$. I assume that this condition is equivalent, in some yet to be determined manner, to the conjunction $doctor'x \wedge Px$, for some as yet undefined property P. Again the set of entities in (4.3) which satisfy this condition is $\{e_1, e_2\}$[3]. This set is added to the two interpretations (c) and (d) to yield:

(c') $\lambda Q.def'_{\{e_1,e_2\}}(\lambda x.Q(doctor')x)$

(d') $\lambda Q\lambda P.P(def'_{\{e_1,e_2\}}(\lambda x.Q(doctor')x))$

Therefore the input to the third cycle of the processor will be the following collection of stacks, differing from those in the trace in Appendix A only in the addition of reference-semantic information:

(4.4)

NP:	(S/(S\NP)):	NP/(N\N):	(S/(S\NP))/(N\N):
def'{e1,e2} (^x.doct'x)	^Q.Q(def'{e1,e2} (^x.doct'x))	^Q.def'{e1,e2} (^x.Q(doct')x)	^Q^P.P(def'{e1,e2} (^x.Q(doct')x))

The operation of the reference resolver should now be clear. It can be summarised by the following recursive algorithm:

[3]Although only e_1 has another property which could satisfy P in (4.3) (i.e. the property of being summoned by e_4), in any reasonable sized knowledge base there would be many unrelated facts about all entities, so it is not possible to assume that the referential annotation in this case is simply $\{e_1\}$.

(4.5) **A Reference Resolver**

```
To reference resolve semantic representation A with respect to knowledge base KB:

  If A is a predicate-argument structure B(C) of a semantic type other than e,
  then reference resolve B and C with respect to KB.

  If A is a lambda expression ^x.B,
  then reference resolve B with respect to KB.

  If A is a conjunction B&C,
  then reference resolve B and C with respect to KB.

  If A is a definite description def'C(^x.B),
  then reference resolve B with respect to KB.
       If some entity in C satisfies B in KB,
       then replace C with C' where C' is the subset of C,
                 where every element satisfies B in KB.
       Else add a new entity E and a new fact B(E) to KB,
            and reference resolve def'C(^x.B) against KB again.
```

This algorithm first identifies the type of semantic representation it is being asked to process. If this is a complex structure other than a definite description (e.g. a lambda expression, a predeicate argument structure), it will reference resolve its sub-parts and then stop. If the semantic representation is a constant or a variable, the algorithm will do nothing. If the semantic representation is a definite description, however, the algorithm will call a subroutine which identifies the set of potential referents for the definite description. It is assumed that every definite description will already have a set of referents C already attached, even if it is simply the entire universe of discourse. The subroutine checks the knowledge base and returns the subset of C, C' which satisfies the relevant condition. If C' is empty, then the reference resolver will create a new entity in the token database and start over[4].

As mentioned above, this algorithm as presently stated ignores other kinds of referring expression. An improved version would have to deal with indefinite descriptions, pronouns, proper names etc.

4.2 Adjudication and Filtering

Now that we have seen how a reference resolver can integrate information from the semantic representation of the string being processed and the current knowledge base, it remains to be

[4]This algorithm as it stands does not provide a way of explaining problems of definite reference exemplified by *the rabbit in the hat*-type noun phrases. However, it could be extended to deal with these, using techniques from Haddock (1987) or Stone (1998).

considered how this information can be utilised by the interpreter in order to filter out analyses whose semantic representations are referentially infelicitous. An extremely simple model of referential adjudication can be developed based on three basic insights:

- If a semantic representation contains a fully specified definite description (i.e. one containing no predicate variables signalling there is more information to come), and the set of possible referents for this definite description contains more than one element, then the representation is referentially infelicitous.
- If a semantic representation contains a non-fully specified definite description, and the set of possible referents for this definite description contains only one element, then the representation is referentially infelicitous.
- All other semantic representations are referentially felicitous.

The first of these concerns semantic representations such as $def'_{\{e_1,e_2\}}(\lambda x.doctor'x)$ from above. Here the description is fully specified, but there are two possible referents — the equivalent of saying "the doctor did X" in a context where more than one doctor is known to exist and the listener cannot be sure which doctor is meant. The second insight is a reflection of the familiar Gricean *Maxim of Economy* (Grice, 1973), and concerns semantic representations such as $def'_{\{e_1\}}(\lambda x.doctor'x \wedge Px)$. In this case, the description contains a predicate variable P and is thus not yet fully specified — the processor is waiting for something else to be said about x. However, enough information has already been presented to identity the referent, since the set of possible referents contains a single element, e_1. In both of these cases the semantic representation can be usefully judged to be referentially infelicitous, since either not enough detail has been given, or more information than is strictly necessary is being anticipated.

From these insights, a straightforward model of referential felicity adjudication can be defined, summarised in the following algorithm:

(4.6) **A Referential Felicity Adjudicator**

```
To adjudicate semantic representation A for referential felicity:

  If A contains a definite description def'C(^x.B),
    where B does not contain a predicate variable,
    and the cardinality of C is greater than 1,
  then mark A as referentially infelicitous.

  If A contains a definite description def'C(^x.B),
    where B contains a predicate variable,
    and the cardinality of C is 1,
  then mark A as referentially infelicitous.

  Otherwise mark A as referentially felicitous.
```

Consider again the processing of the garden path sentence *the doctor sent for the patient arrived* against the knowledge base in (4.3) where two doctors are known to exist. At the end of the reference resolution phase of the second processor cycle, there are still four parse alternatives under consideration, presented in (4.4) above. When these are adjudicated for referential felicity, the first and second will be marked as being referentially infelicitous, since they contain a definite description with a fully specified description but whose set of potential referents has more than one element. The other two alternatives are marked as being felicitous. If we were to posit a sensibleness filter which occurred immediately after the referential felicity adjudicator, then the first and second stacks in (4.4) would be discarded before the third cycle of the process, and the lack of a garden path effect would successfully predicated, thereby explaining **Observation 2** in the preamble to this chapter.

This model of adjudication and immediate filtering also captures **Observation 1**. Imagine the same sentence *the doctor sent for the patient arrived* is being processed with respect to the following knowledge base, where a single doctor is known to exist, corresponding to situation (1.3) in section 1.1:

(4.7)

$e_1\ e_2\ e_3$
$doctor'e_1$
$nurse'e_2$
$patient'e_3$

During the second cycle of the processor, immediately after the reference resolution phase, there will be four semantic representations under consideration:

(a) $def'_{\{e_1\}}(\lambda x.doctor'x)$

(b) $\lambda P.P(def'_{\{e_1\}}(\lambda x.doctor'x))$

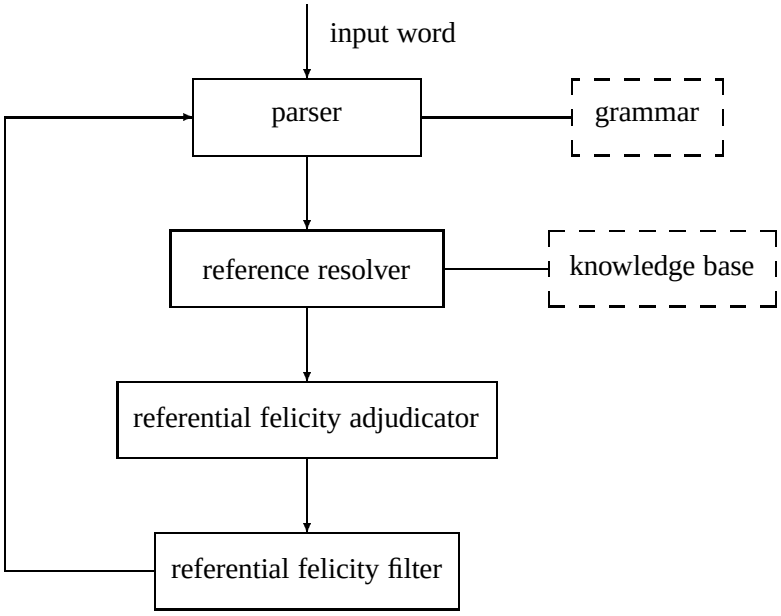
(c) $\lambda Q.def'_{\{e_1\}}(\lambda x.Q(doctor')x)$

(d) $\lambda Q\lambda P.P(def'_{\{e_1\}}(\lambda x.Q(doctor')x))$

When these semantic interpretations come to be adjudicated for referential felicity, (c) and (d) will be marked as being infelicitous, since they contain a definite description where the condition is not yet fully specified but the referent set is a singleton. Assuming that they are discarded immediately from consideration, the end result of the second phase of processing (i.e. after *the doctor*) is that only the first two stacks in (4.4) remain. Thus, the garden path effect is successfully predicted and **Observation 1** explained, since by the time the substring *the doctor sent for* has been processed, only the S/NP analysis will remain under consideration.

In conclusion, we now have a simple model of natural language understanding which can be summarised in the following diagram:

(4.8)



The parser is the ruthless reduce-first parser from algorithm (3.13), consulting a CCG, including the lexicon in (3.16). The interpreter consists of three devices operating in series and intervening between cycles of the parser: (a) a reference resolver, defined in algorithm (4.5); (b) a referential felicity adjudicator, defined in algorithm (4.6); and (c) a referential felicity filter operating under a policy of immediately discarding referential infelicitous semantic representations from the workspace. The adjudicator has the task of identifying and marking infelicitous expressions; the filter is the mechanism that actually deletes them from the workspace. The reasons for separating these two functions will become clear in the following section.

This model can successfully account for **Observation 1** and **Observation 2** from the preamble to this chapter, concerning the relative strength of the garden path effect when processing the sentence *the doctor sent for the patient arrived*.

4.3 A Problem with the Model

The diagram in (4.8) presents an integrated model of parsing, reference resolution and referential felicity adjudication and filtering which successfully account for **Observation 1** and **Observation 2** above, concerning the processing of the garden path sentence *the doctor sent for the patient arrived* in different contexts. But can this same model be used to explain **Observation 3**, repeated here?

Observation 3 When the string *the flowers sent for* is encountered in any referential context whatsoever, both the NP/NP and the $(S/(S\backslash NP))/NP$ analyses must remain on the workspace after interpretation, and the S/NP analysis can be discarded.

Imagine that the sentence *the flowers sent for the patient arrived* is being processed by the model in (4.8) in isolation i.e. in a context where there is no relevant knowledge about any flowers whatsoever. The process of parsing, reference resolving and filtering the first couple of words of this sentence will run as follows:

Cycle 1 The first word *the* is shifted. No reduction is possible. No referential annotations can be made and so no filtering can happen. At the end of cycle 1 the workspace will be identical to that in the trace in Appendix A.

Cycle 2 The second word *flowers* is shifted and the workspace is reduced to yield four stacks equivalent to those in cycle 2 of the trace in Appendix A. The reference resolver finds no entity in the empty token database which satisfies the condition $\lambda x.flowers'x$ so it

creates a new one, say e_{23}, adds it to the token database, where it serves as the referent of the definite description $def'(\lambda x.flowers'x)$. At the end of the reference resolution phase of cycle 2, the workspace will consist of the following four stacks:

(4.9)

NP:	(S/(S\NP)):	NP/(N\N):	(S/(S\NP))/(N\N):
def'{e23}	^Q.Q(def'{e23}	^Q.def'{e23}	^Q^P.P(def'{e23}
(^x.flow'x)	(^x.flow'x))	(^x.Q(flow')x)	(^x.Q(flow')x))

Assuming that the referential felicity adjudication and filter phases are the next to be performed by the processor, then stacks 3 and 4 on this workspace will be eliminated during cycle 2 and will not be available for further processing. This has the unfortunate result that a garden path effect in the sentence *the flowers sent for the patient arrived* is predicted, counter to **Observation 3** above.

Therefore, it is evident that the model of natural language understanding in figure (4.8) is unsatisfactory. The discussion of garden path effects in section 1.1 suggested that the reason for the lack of processing difficulty with this sentence is that, at some point during the third or fourth cycles of the processor, the semantic representations are evaluated for plausibility of the states and events they describe. At this point, the analysis based on the finite verb reading of *sent* will be discarded, since it is implausible that flowers should summon something. The analyses based on the past participle reading of *sent* should remain on the workspace so that the process can continue successfully.

Unfortunately, in model (4.8) the referential felicity filter occurs too soon for this to happen – the stacks which are needed to remain on the workspace at the start of cycle 3, so as to allow the past participle reading of *sent* to be integrated, have been deleted already. If the processor then eliminates all stacks formed from the remaining two analyses, since they will be semantically implausible, the processor will be unable to find any analysis of the string *the flowers sent for*.

This presents us with a conundrum. On the one hand, we want the processor to identity and delete referentially felicitous analyses of *the doctor* as soon as possible. On the other hand, we want the processor to retain referentially infelicitous analyses of the equivalent string *the flowers*, at least until the next cycle.

Two possible ways of dealing with this problem are as follows:

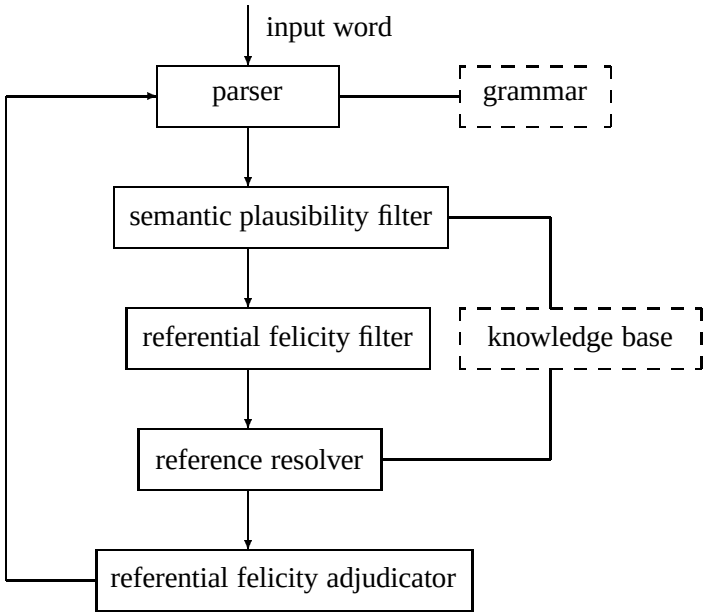
- Design a processor which discards referentially infelicitous analyses some finite number of cycles after they have been identified as such. In other words, the referential felicity

adjudicator will label infelicitous semantic representations with a counter and increment the counter once every cycle. Once the counter has reached a predetermined number, the referential felicity filter will discard the analysis from the workspace. For the purposes of the garden path data discussed in this thesis, the limit could be set to 1 – infelicitous expressions will be discarded during the cycle after they have been discovered to be so.

- Following Steedman (2000:ch.9) we could assume a beam-search approach to the problem, whereby referentially infelicitous analyses are not discarded from the workspace until a predetermined threshold of competing parse stacks is reached. Infelicitous expressions will be ranked low on some kind of hierarchy of importance, and will be deleted when the processor runs out of memory space.

I propose a variant of the first solution which does not require the addition of a counter to parse stacks. I retain the modules represented in the diagram in (4.8), but rearrange their ordering. Thus, instead of occurring immediately after the referential felicity adjudicator in the cycle, the referential felicity filter occurs *before* it. This will have the desired result that the definite description within the semantic representations of the string *the flowers* will not be filtered for referential felicity until after the string *the flowers sent* has been parsed and evaluated for semantic plausibility. This proposal suggests the following model of natural language understanding:

(4.10)



The semantic plausibility filter will examine the semantic representations of each analysis currently under consideration and eliminate all those which are inconsistent with the facts contained within the knowledge base. Assuming that the workspace at the end of the second processing cycle of the sentence *the flowers sent for the patient arrived* is similar to that in (4.9) above, with the addition of referential infelicity markers to stacks 3 and 4, the shift and reduce phases of the third cycle will result in the following three parse alternatives:

(4.11)

	infelicitous!	infelicitous!
S/PP:	NP/PP:	(S/(S\NP))/PP:
^y.summ'y(def'{e23}(^x.flow'x))	^z.def'{e23}(^y.flow'y&send'zyarb')	^z^P.P(def'{e23}(^y.flow'y&send'zyarb'))

At this point, the semantic plausibility filter evaluates each of the semantic representations in (4.11) against the facts in the knowledge base, and decided whether they are consistent or not. Assuming that the knowledge base contains the following general facts:

(a) $\forall x.(\exists y.summon'(x,y)) \Rightarrow human'(x)$

(b) $\forall x.flowers'(x) \Rightarrow \neg human'(x)$

The first of these states that only humans can summon things, and the second that flowers are not humans. The semantic representation in the first parse stack in (4.11), repeated as (c) below, is converted by the plausibility filter into the first order equivalent (d), where lambda operators have turned into existential quantifiers:

(c) $\lambda y.summon'y(def'_{\{e_{23}\}}(\lambda x.flowers'x))$

(d) $\exists x \exists y.flowers'(x) \wedge summon'(x,y)$

A first order theorem prover will find the semantic representation in (d) inconsistent with any monotonic knowledge base containing the facts (a) and (b). Thus, stack 1 will be discarded from the workspace at the end of the semantic plausibility filter phase, and stacks 2 and 3 will remain.

The next phase in the process involves the referential felicity filter. The workspace now contains two parse stacks, both of which are marked as being referentially infelicitous. I assume the filter operates under the following strategy:

(4.12)
```
If the workspace contains at least one referentially felicitous parse stack,
   then delete all referentially infelicitous ones.
   If the workspace contains only referentially infelicitous parse stacks,
   then remove their infelicity markings and retain them.
```

Since the only parse stacks remaining are both referentially infelicitous, they are thus retained by the filter and stripped of their infelicity annotations, resulting in the following workspace at the end of the third cycle:

(4.13)

```
---------------------------------     ---------------------------------------
             NP/PP:                                (S/(S\NP))/PP:
^z.def'{e23}(^y.flow'y&send'zysb')     ^z^P.P(def'{e23}(^y.flow'y&send'zysb'))
---------------------------------     ---------------------------------------
```

Since the postmodified noun phrase analyses of the string *the flowers sent* remain on the workspace at the end of the third parse cycle, the lack of a garden path effect in the sentence *the flowers sent for the patient arrived* is successfully predicted, and **Observation 3** from the preamble to this chapter is explained by the model in (4.10).

In conclusion, this chapter has developed an integrated model of semantic interpretation which is sufficient to account for the evidence about garden path effects discussed in chapter 1. The model is a simple series of modules, represented diagrammatically in figure (4.10). The processor as a whole operates cyclically, with each cycle being initiated by a new input word entering the system. The lexical entries for the new word are integrated into the workspace by the parser, defined in algorithm (3.13), which then passes the possible semantic representations over to the interpreter module. The interpreter module itself is a series of submodules: (a) the semantic plausibility filter checks whether the states and events described within the semantic representations are consistent with the facts in the knowledge base and, if not, arranges for the analysis to be discarded; (b) the referential felicity filter is responsible for discarding any analyses whose semantic interpretations have, in previous cycles, been judged to be referentially infelicitous; (c) the reference resolver, defined in algorithm (4.5), finds a set of possible referents for the referring expressions within the various semantic representations currently being considered; and finally (e) the referential felicity adjudicator, defined in algorithm (4.6), decides which semantic representations are referentially infelicitous and marks them as such.

One of the most important features of this model is the separation of the referential felicity adjudication and filtering modules, and the positioning of the latter before the former in the processing cycle. In section 4.3 this was argued to be necessary to explain the fact that the sentence *the doctor sent for the patient arrived* provokes a garden path effect when processed in isolation, whereas the structurally similar sentence *the flowers sent for the patient arrived* does not. In the first of these, the postmodified reading of *the doctor* is adjudged to be referentially infelicitous, and is hence discarded during the subsequent cycle, leading to the garden

path effect. In the second sentence however, although the postmodified analysis of *the flowers* is also adjudged to be referentially infelicitous during the second cycle, this judgement is reversed during the third cycle after the non-postmodifoed analysis is eliminated by the semantic plausibility filter – since the postmodified analysis remains under consideration, the garden path effect is avoided.

The model of natural language understanding presented in this chapter and summarised in the diagram in (4.10) has been implemented in the Java programming language as part of this project. The implementation is a transparent version of the model, which accepts input text one word at a time, and interrupts the user when understanding fails i.e. when the processor can find no analysis of the utterance so far which is both grammatical and sensible. The source code for the implementation is archived along with this thesis.

Chapter 5

Conclusion

Chapter 1 presented evidence from the study of garden path effects in human sentence processing which suggest that: (a) Natural language understanding is an incremental process, utterances being integrated with the processor's knowledge state more or less word-by-word; (b) Structural ambiguities are resolved mid-sentence, and not all grammatical analyses are maintained until the end of the utterance; (c) Ambiguity resolution is determined by the semantic sensibleness of the competing semantic representations of the utterance processed thus far; and (d) A sensible semantic representation is one which is both referentially felicitous with respect to the current knowledge state, and where the described states and events are plausible.

Chapter 1 went on to précis an argument from Steedman (2000) to the effect that: (a) In order to account for garden path effects a parser must make available semantic representations for sentence fragments like *the flowers sent for*, to be evaluated for referential felicity and plausibility; (b) By taking a traditional categorial grammar and adding operations of functional composition and type raising, a grammatical formalism can be created which licenses these fragments as syntactic constituents with compositionally derived semantic representations; and (c) This grammatical formalism, known as Combinatory Categorial Grammar (CCG), is compatible with a very simple model of parsing, and is thus superior to other grammatical formalisms which require the parser to be supplemented with extra-grammatical apparatus in order to construct semantic representations for incomplete constituents.

Chapters 2 and 3 present a taxonomy of shift-reduce parsing for CCGs. Chapter 2 notes that CCGs are compatible with one of the simplest classes of parser — the bottom-up shift-reduce parser family. Two versions of this parser were then introduced: (a) the basic non-deterministic shift-reduce parser; and (b) a version which uses naïve breadth-first search to simulate the

non-determinism deriving from lexical ambiguity, shift-reduce conflicts and rule choice. The result is a parser which is complete but extremely inefficient, the number of parse stacks being considered increasing exponentially as more words are processed.

Chapter 3 examined ways of eliminating the non-determinism in a shift-reduce parser. A reduce-first parser was introduced, which eliminates non-determinism resulting from shift-reduce conflicts by consistently prioritising the reduce operation. This reduce-first parser is more efficient than the naïve breadth-first approach, but will only recognise utterances which have a left-branching derivation. One way to overcome this is to expand the lexicon so that every sentence has such an incremental derivation, for instance by allowing verbs to optionally subcategorise for VP-postmodifiers. Finally, a more efficient version of the reduce-first parser was introduced, which reduces non-determinism deriving from lexical ambiguity by discarding all parse stacks which fail to reduce. The resulting parser appears to offer something like constant complexity, since the number of parse stacks being considered remains fairly as more words are processed.

Chapter 4 developed a model of analysis interpretation, where the competing semantic representations constructed by the parser are integrated with the information contained within a knowledge base encoding the current state of knowledge of the processor. The interpreter consists of a series of submodules: (a) the semantic plausibility filter checks whether the states and events described within the semantic representations are consistent with the facts in the knowledge base and, if not, arranges for the analysis to be discarded; (b) the referential felicity filter is responsible for discarding any analyses whose semantic interpretations have, in previous cycles, been judged to be referentially infelicitous; (c) the reference resolver finds a set of possible referents for the referring expressions within the various semantic representations currently being considered; and (e) the referential felicity adjudicator decides which semantic representations are referentially infelicitous and marks them as such.

The model developed in this project is sufficient to explain the data upon which it is motivated, specifically the garden path sentences discussed in chapter 1, and the differing garden path effects they engender in different contexts. Whether the model can account for other aspects of human sentence processing remains to be seen. I believe the model could be improved upon in future work in the following ways:

- Different models of parser could be examined in more detail. The model presented here, the ruthless shift-reduce parser was chosen because of its extreme efficiency. However, it was pointed out that the more powerful and effective the interpreter is, the less effi-

cient the parsing module itself has to be. Therefore, it should be possible to bring the parser and competence grammar back into alignment, by using a more complete but less efficient parser, if the interpreter can be sufficiently improved.

- The form of semantic representation used here, basically a form of untyped lambda calculus, is just one choice out of many. There has been a lot of work on alternative formalisms for knowledge representation in computational linguistics (for example Discourse Representation Theory, Flat Semantics etc.), and it may be advantageous to try these out. Specifically, it would be interesting to integrate the processor with a more powerful all-round reasoning system, such as a Prolog knowledge base. This way, some of the ad hoc features of the interpreter could be made more general.

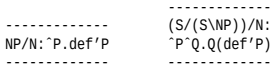
- A wider range of phenomena from human sentence processing could be brought within the model, for example, other types of attachment ambiguity.

Appendix A

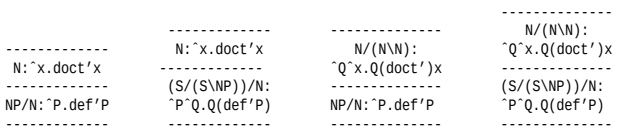
A Reduce-First Parse Trace

A trace of the ruthless reduce-first parser processing the sentence *the doctor sent for the patient arrived* with a null interpreter module

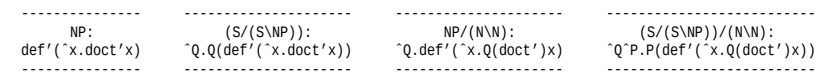
Cycle 1: Shift *the*



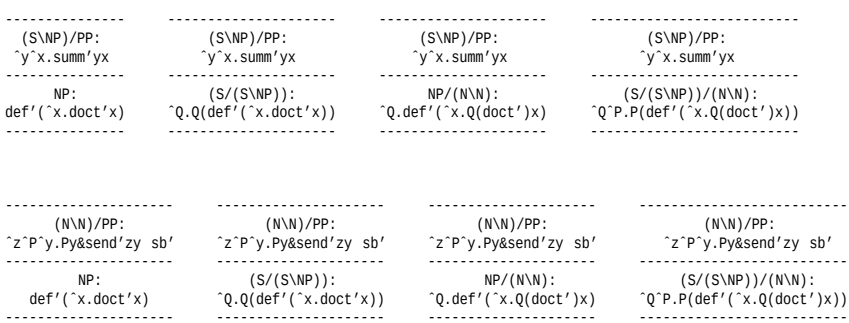
Cycle 2: Shift *doctor*



Reduce — all stacks reduce



Cycle 3: Shift *sent*



Reduce — stacks 2, 7, 8 reduce; others eliminated

```
-------------------------     -----------------------------     ---------------------------------
         S/PP:                          NP/PP:                          (S/(S\NP))/PP:
ˆy.summ'y(def'(ˆx.doct'x))     ˆz.def'(ˆy.doct'y&send'zy sb')     ˆzˆP.P(def'(ˆy.doct'y&send'zy sb'))
-------------------------     -----------------------------     ---------------------------------
```

Cycle 4: Shift *for*

```
-------------------------     -----------------------------     ---------------------------------
       PP/NP:ˆx.x                      PP/NP:ˆx.x                        PP/NP:ˆx.x
-------------------------     -----------------------------     ---------------------------------
         S/PP:                          NP/PP:                          (S/(S\NP))/PP:
ˆy.summ'y(def'(ˆx.doct'x))     ˆz.def'(ˆy.doct'y&send'zy sb')     ˆzˆP.P(def'(ˆy.doct'y&send'zy sb'))
-------------------------     -----------------------------     ---------------------------------
```

Reduce — all stacks reduce

```
-------------------------     -----------------------------     ---------------------------------
         S/NP:                          NP/NP:                          (S/(S\NP))/NP:
ˆy.summ'y(def'(ˆx.doct'x))     ˆz.def'(ˆy.doct'y&send'zyarb')     ˆzˆP.P(def'(ˆy.doct'y&send'zyarb'))
-------------------------     -----------------------------     ---------------------------------
```

Cycle 5: Shift *the*

```
-------------------------     -----------------------------     ---------------------------------
      NP/N:ˆP.def'P                   NP/N:ˆP.def'P                     NP/N:ˆP.def'P
-------------------------     -----------------------------     ---------------------------------
         S/NP:                          NP/NP:                          (S/(S\NP))/NP:
ˆy.summ'y(def'(ˆx.doct'x))     ˆz.def'(ˆy.doct'y&send'zy sb')     ˆzˆP.P(def'(ˆy.doct'y&send'zy sb'))
-------------------------     -----------------------------     ---------------------------------
```

```
-------------------------     -----------------------------     ---------------------------------
(S/(S\NP))/N:ˆPˆQ.Q(def'P)      (S/(S\NP))/N:ˆPˆQ.Q(def'P)          (S/(S\NP))/N:ˆPˆQ.Q(def'P)
-------------------------     -----------------------------     ---------------------------------
         S/NP:                          NP/NP:                          (S/(S\NP))/NP:
ˆy.summ'y(def'(ˆx.doct'x))     ˆz.def'(ˆy.doct'y&send'zy sb')     ˆzˆP.P(def'(ˆy.doct'y&send'zy sb'))
-------------------------     -----------------------------     ---------------------------------
```

Reduce — stacks 1, 2, 3 reduce; others eliminated

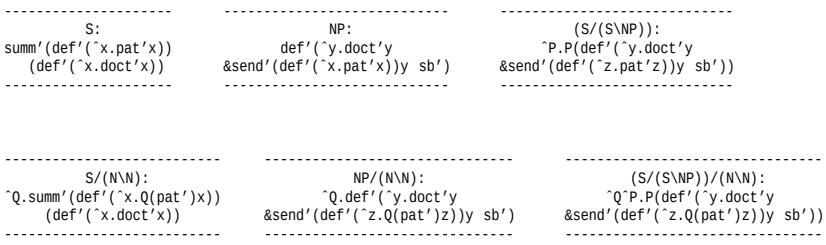
```
-------------------------------     ---------------------------------     -----------------------------------------
             S/N:                              NP/N:                              (S/(S\NP))/N:
ˆP.summ'(def'P)(def'(ˆx.doct'x))     ˆP.def'(ˆy.doct'y&send'(def'P)y sb')     ˆQˆP.P(def'(ˆy.doct'y&send'(def'Q)y sb'))
-------------------------------     ---------------------------------     -----------------------------------------
```

Cycle 6: Shift *patient*

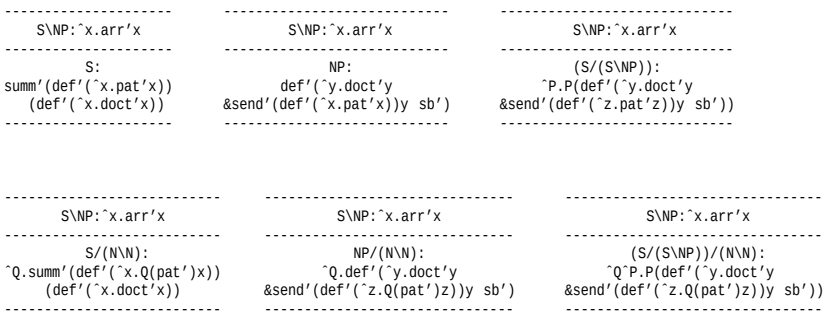
```
-------------------------------     ---------------------------------     -----------------------------------------
          N:ˆx.pat'x                          N:ˆx.pat'x                          N:ˆx.pat'x
-------------------------------     ---------------------------------     -----------------------------------------
             S/N:                              NP/N:                              (S/(S\NP))/N:
ˆP.summ'(def'P)(def'(ˆx.doct'x))     ˆP.def'(ˆy.doct'y&send'(def'P)y sb')     ˆQˆP.P(def'(ˆy.doct'y&send'(def'Q)y sb'))
-------------------------------     ---------------------------------     -----------------------------------------
```

```
-------------------------------     ---------------------------------     -----------------------------------------
     N/(N\N):ˆQˆx.Q(pat')x               N/(N\N):ˆQˆx.Q(pat')x                N/(N\N):ˆQˆx.Q(pat')x
-------------------------------     ---------------------------------     -----------------------------------------
             S/N:                              NP/N:                              (S/(S\NP))/N:
ˆP.summ'(def'P)(def'(ˆx.doct'x))     ˆP.def'(ˆy.doct'y&send'(def'P)y sb')     ˆQˆP.P(def'(ˆy.doct'y&send'(def'Q)y sb'))
-------------------------------     ---------------------------------     -----------------------------------------
```

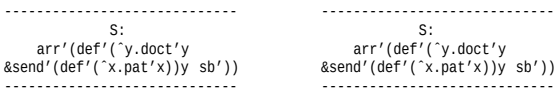
Reduce — all stacks reduce



Cycle 7: Shift *arrived*



Reduce — stacks 2, 3 reduce; others eliminated



End of Trace

Bibliography

Ajdukiewicz, K. (1935). Die syntaktische konnexität. In McCall, S., editor, *Polish Logic 1920-1939*, pages 207–231. Oxford University Press. Translated from Studia Philosophica, 1, 1-27.

Altmann, G. and Steedman, M. (1988). Interaction with context during human sentence processing. *Cognition*, 30:191–238.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Bever, T. (1970). The cognitive basis for linguistic structures. In Hayes, J., editor, *Cognition and the Development of Language*, pages 279–362. Wiley, New York.

Crain, S. and Steedman, M. (1985). On not being led up the gardem path: the use of context by the psychological parser. In Kartunnen, L., Dowty, D., and Zwicky, A., editors, *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pages 320–358. Cambridge University Press.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic*, volume 1. North Holland, Amsterdam.

Dowty, D. (1988). Type-raising, functional composition, and non-constituent coordination. In Oehrle, R. T., Bach, E., and Wheeler, D., editors, *Categorial Grammars and Natural Language Structures*, pages 153–198. Reidel, Dordrecht.

Fodor, J., Bever, T., and Garrett, M. (1974). *The Psychology of Language*. McGraw-Hill, New York.

Frazier, L. (1978). *On Comprehending Sentences*. PhD thesis, University of Connecticut.

Grice, H. (1973). Logic and conversation. In Cole, P. and Morgan, J., editors, *Speech Acts*, volume 3 of *Syntax and Semantics*, pages 41–58. Academic Press, New York.

Haddock, N. (1987). Incremental interpretation and combinatory categorial grammar. In Haddock, N., Klein, E., and Morrill, G., editors, *Categorial Grammar, Unification Grammar and Parsing*, number 1 in Working Papers in Cognitive Science, pages 71–84. Centre for Cognitive Science, University of Edinburgh.

Hepple, M. (1987). Methods for parsing combinatory grammars and the spurious ambiguity problem. Master's thesis, University of Edinburgh.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic*. Kluwer.

Kimball, J. (1973). Seven principles of surface structure parsing in natural language. *Cognition*, 2:15–47.

Komagata, N. (1999). *Information Structure in Texts: A Computational Analysis of Contextual Appropriateness in English and Japanese*. PhD thesis, University of Pennsylvania.

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press.

Marslen-Wilson, W. and Tyler, L. K. (1980). The temporal structure of spoken language understanding. *Cognition*, 8:1–71.

Mellish, C. S. (1981). *Coping with Uncertainty: Noun Phrase Interpretation and Early Semantic Analysis*. PhD thesis, University of Edinburgh.

Niv, M. (1993). *A Computational Model of Syntactic Processing: Ambiguity Resolution from Interpretation*. PhD thesis, University of Pennsylvania.

Niv, M. (1994). A psycholinguistically motivated parser for ccg. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, Las Cruces, NM*, pages 125–132. Morgan Kaufmann.

Pareschi, R. (1987). Combinatory grammar, logic programming, and natural language processing. In Haddock, N., Klein, E., and Morrill, G., editors, *Categorial Grammar, Unification Grammar and Parsing*, number 1 in Working Papers in Cognitive Science, pages 85–114. Centre for Cognitive Science, University of Edinburgh.

Pareschi, R. and Steedman, M. (1987). A lazy way to chart parse with categorial grammars. In *Proceedings of the 25nd Annual Meeting of the Association for Computational Linguistics, Stanford, CA*, pages 81–88. Morgan Kaufmann.

Pulman, S. (1986). Grammars, parsers and memory limitations. *Language and Cognitive Processes*, 1(3):199–225.

Schieber, S. and Johnson, M. (1993). Variations on incremental interpretation. *Journal of Psycholinguistic Research*, 22(2):287–318.

Stabler, E. (1991). Avoid the pedestrian's paradox. In et al., R. B., editor, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 199–237. Kluwer, Netherlands.

Steedman, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, 61:523–568.

Steedman, M. (2000). *The Syntactic Process*. MIT Press.

Stone, M. (1998). *Modality in Dialogue: Planning Pragmatics and Computation*. PhD thesis, University of Pennsylvania.

Tanenhaus, M., Garnsey, S., and Boland, J. (1990). Combinatory lexical information and language comprehension. In Altmann, G., editor, *Cognitive Models of Speech Processing*. MIT Press.