PART III
Computation and Performance

BLANK PAGE

Chapter 8

Combinators and Grammars

"I once asked Bravura whether there were any Kestrels in his forest. He seemed somewhat upset by the question, and replied in a strained voice: 'No! Kestrels are not allowed in this forest!'"
Raymond Smullyan, *To Mock a Mockingbird*

What does the theory presented in the earlier chapters actually tell us? Why should natural grammars involve combinatory rules, rather than the intuitively more transparent apparatus of the λ-calculus? Why are the combinators in question apparently confined to Smullyan's Bluebird, Thrush, and Starling—that is, to composition, type-raising, and substitution? Why are the syntactic combinatory rules further constrained by the Principles of Consistency and Inheritance? What expressive power does this theory give us? How can grammars like this be parsed?

8.1 Why Categories and Combinators?

There is a strong equivalence between (typed and untyped) combinatory systems and the (typed and untyped) λ-calculi, first noted by Schönfinkel (1924), elaborated by Curry and Feys (1958), and developed and expounded by Rosenbloom (1950), Stenlund (1972), Burge (1975), Barendregt (1981), Smullyan (1985, 1994), and Hindley and Seldin (1986). Even quite small collections of combinators of the kind already encountered are sufficient to define applicative systems of expressive power equal to that of the λ-calculus, as will be demonstrated below.

The difference between the λ-calculi and the combinatory systems is that the latter avoid the use of bound variables. One interest of this property lies in the fact that bound variables can be a major source of computational overhead—for example in the evaluation of expressions in programming languages related to the λ-calculus, such as LISP. The freedom that their users demand to use the same identifier for variables that are logically distinct in the sense of having distinct bindings to values in distinct environments means that all the various bindings must be stored during the evaluation. This cost is serious enough that considerable ingenuity is devoted to minimizing it by the designers of such "functional" programming languages. One tactic, originating with Turner (1979b), is to avoid the problem entirely, by compiling languages like LISP into equivalent variable-free combinatory expressions, which can then be evaluated by structural, graph reduction techniques akin to algebraic simplification. We will see that there some rather striking similarities between the combinatory system that Turner proposes and the one that is at work in natural languages.

However, it seems quite unlikely that a pressure to do without variables for reasons of computational efficiency is at work in natural language interpretation.[1] The computational advantage of the combinatory systems is highly dependent upon the precise nature of the computations involved, and it is far from obvious that these particular types of computation are characteristic of linguistic comprehension (although the extensive involvement of higher-order functions in CCG is one property that does exacerbate the penalties incurred from the use of bound variables). Furthermore, the wide acceptance of the idea that the pronoun in sentences like *Every farmer in the room thinks he is a genius* is semantically a bound variable, as assumed in the analysis of such phenomena in section 4.4 in chapter 4, suggests that there is no overall prohibition against such devices at the level of Logical Form or predicate-argument structure. The binding conditions, and in particular Condition C, which are discussed in terms of CCG in Chierchia 1988 and Steedman 1997, are also phenomena that are most naturally thought of in terms of scope (although it has to be said that they do not look much like the properties of the *usual* kind of variables).[2]

It seems more likely that natural grammars reflect a combinatory semantics because combinator-like operations such as composition are themselves cognitively primitive and constitute a part of the cognitive substrate from which the language faculty has developed. Such primitive and prelinguistic cognitive operations as learning how to reach one's hand around an obstacle to a target have many of the properties of functional composition, if elementary movements are viewed as functions over locations. The onset of the ability to construct such composite motions appears to immediately precede the onset of language in children (Diamond 1990, 653–655). Similarly, a notion very like type-raising seems to be implicit in the kind of association between objects and their characteristic roles in actions that is required in order to use those objects as tools in planned action. (The idea that tool use and motor planning are immediate precursors of language goes back to de Laguna's (1927) observations on Köhler's's (1925) work on primate tool use and has been investigated more recently by Bruner (1968), Greenfield, Nelson and Saltzman (1972), Greenfield (1991), and Deacon (1988, 1997), among many others.)

To the extent that languages adhere to the Principle of Head Categorial Uniqueness and project unbounded dependencies from the same categories that define canonical word order, the presumed universal availability of combinatory operations in principle allows the child to acquire the full grammar of the language on the basis of simple canonical sentences alone, on the assumption of chapter 2, that the child has access (not necessarily error-free, and not necessarily unambiguously) to their interpretations. (We will return briefly to the problems induced by exceptions to Head Categorial Uniqueness in chapter 10.)

To see whether this hypothesis is reasonable, we must begin by examining the specific combinators that have been identified above—composition, type-raising, and Schönfinkel's **S**—and ask what class of concepts can be defined using them.

8.2 Why Bluebirds, Thrushes, and Starlings?

The equivalence between combinatory systems and the λ-calculus is most readily understood in terms of a recursive algorithm for converting terms in the λ-calculus into equivalent combinatory expressions. Surprisingly small collections of combinators can be shown in this way to completely support this equivalence. One of the smallest and most elegant sets consists of three combinators, **I**, **K**, and the familiar **S** combinator. The algorithm can be represented as three cases, as follows:[3]

(1)
$$\begin{aligned} \lambda x.x &= \mathbf{I} \\ \lambda x.y &= \mathbf{K}y \\ \lambda x.AB &= \mathbf{S}(\lambda x.A)(\lambda x.B) \end{aligned}$$
where x is not free in y

The combinators **I** and **K** have not been encountered before, but their definitions can be read off the example: **I** is the identity operator, and **K**, Smullyan's Kestrel, is vacuous abstraction or the definition of a constant function. This algorithm simply says that these two combinators represent the two ground conditions of abstracting over the variable itself and abstracting over any other variable or constant, and that the case of abstracting over a compound term consisting of the application of a function term A to an argument B is the Starling combinator **S** applied to the results of abstracting over the function and over the argument. (Given the earlier definition of **S**, it is easy to verify that this equivalence holds.) Since the combinator **I** can in turn be defined in terms of the other two combinators (as **SKK**), the algorithm (attributed in origin to Rosser (1942) in Curry and Feys 1958, 237) is often referred to as the "**SK**" algorithm. It is obvious that the algorithm is complete, in the sense that it will deliver a combinatory equivalent of any λ-term. It therefore follows that any combinator, including composition and type-raising, can be defined in terms of **S** and **K** alone.

The **SK** algorithm is extremely elegant, and quite general, but it gives rise to extremely cumbersome combinatory expressions. Consider the following examples, adapted from Turner 1979b. The successor function that maps an integer onto the integer one greater might be defined as follows in an imaginary functional programming language:

(2) $succ = \lambda x.plus\ 1\ x$

The obvious variable-free definition of this trivial function is the following:

(3) $succ = plus\ 1$

However, the **SK** algorithm produces the much more cumbersome (albeit entirely correct) expression shown in the last line of the following derivation:

(4)
$$\begin{array}{lll} succ & = & \lambda x.plus\ 1\ x \\ & \Rightarrow & \mathbf{S}\lambda x.plus\ 1\lambda x.x \\ & \Rightarrow & \mathbf{S}(\mathbf{S}\lambda x.plus\lambda x.1)\mathbf{I} \\ & \Rightarrow & \mathbf{S}(\mathbf{SK}plus\mathbf{K}1)\mathbf{I} \end{array}$$

The following is the familiar recursive definition of the factorial function (where *cond A B C* means "if A then B else C"):[4]

(5) $fact = \lambda x.cond(equal\ 0\ x)1(times\ x(fact(minus\ x\ 1)))$

It yields the following monster:

(6)
$$\begin{array}{l} \mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{K}\ cond)\ (\mathbf{S}(\mathbf{S}(\mathbf{K}\ equal)(\mathbf{K}\ 0))\mathbf{I}))(\mathbf{K}\ 1)) \\ \qquad (\mathbf{S}(\mathbf{S}(\mathbf{K}\ times)\mathbf{I})(\mathbf{S}(\mathbf{K}\ fact) \\ \qquad (\mathbf{S}(\mathbf{S}(\mathbf{K}\ minus)\mathbf{I})(\mathbf{K}\ 1)))) \end{array}$$

What is wrong with the **SK** algorithm is that it fails to distinguish cases in which either the function or the argument or both are terms in which the variable x does not occur (is not free) from the general case in which both function and argument are terms in x. It is only in the latter case that the combinator **S** is appropriate. Curry and Feys (1958, 190–194) offer the following alternative algorithm:[5]

(7) a. $\lambda x.x \quad = \quad \mathbf{I}$
b. $\lambda x.y \quad = \quad \mathbf{K}y$
c. $\lambda x.fx \quad = \quad f$
d. $\lambda x.fA \quad = \quad \mathbf{B}f(\lambda x.A)$
e. $\lambda x.Ay \quad = \quad \mathbf{C}(\lambda x.A)y$
f. $\lambda x.AB \quad = \quad \mathbf{S}(\lambda x.A)(\lambda x.B)$
where x is not free in f, y

This algorithm distinguishes the case (7c) (corresponding to η-reduction), in which the expression to be abstracted over consists of a function term that does not contain the variable and an argument term that *is* the variable. This case immediately preempts a great many applications of **K** (to constants), **I** (for the variable), and **S** (for the application). For example, it immediately gives us what we want for the successor function:

(8) $\mathit{succ} \quad = \quad \lambda x.\mathit{plus}\ 1\ x$
$\qquad\qquad \Rightarrow \quad \mathit{plus}\ 1$

The new algorithm also distinguishes the cases (7d) and (7e), where either the argument term or the function term do not include the variable. These cases correspond to the familiar functional composition combinator **B**, and the "commuting" combinator **C**, which has not been encountered in natural syntax before, but whose definition is as follows:[6]

(9) $\mathbf{C}fxy \quad \equiv \quad fyx$

This algorithm gives rise to much terser combinatory expressions. For example, the earlier definition of factorial comes out as follows:

(10) $\mathbf{S}(\mathbf{C}(\mathbf{B}\mathit{cond}(\mathit{equal}\ 0))\ 1)(\mathbf{S}\mathit{times}(\mathbf{B}\mathit{fact}(\mathbf{C}\mathit{minus}\ 1)))$

Like the **SK** set, this set of combinators is complete with respect to the λ-calculi. This result is obvious, since it includes **S** and **K**. More interestingly, in includes other subsets that are also complete. The most interesting of these is the set **BCSI**. This set is complete with respect to the λ-calculus with the single exception that **K** itself is not definable. This set therefore corresponds to the λ-calculus without vacuous abstraction, which is known as the $\lambda_{\mathbf{I}}$-calculus (Church 1940), as distinct from the $\lambda_{\mathbf{K}}$-calculus. Vacuous abstraction is the operation that figured as an irrelevant side effect of Huet's unification algorithm in the discussion in chapter 7 of work by Dalrymple, Shieber, and Pereira (1991; see also Shieber, Pereira and Dalrymple 1996), who used it as an operation on predicate-argument structures, to recover interpretations for VP ellipsis. It is therefore interesting to note the existence of calculi and combinatory systems that exclude it, corresponding to linear, relevance, and intuitionistic logics, and to recall that it is not represented among the syntactic combinatory rules either.

Turner (1979a,b) and others have proposed further cases to optimise and extend similar translations of λ-terms into combinatory equivalents, including combinators corresponding to **T**, the type-raising combinator (which is a natural partner to **C** in (7)), to Curry's Φ, the combinator that is implicit in the coordination rule proposed earlier, and to the "paradoxical" fixed-point combinators that are required to complete the combinatory definition of recursive functions like (10).

What then can we say concerning the nature and raison d'être of the combinatory system **BTS** that we have observed in natural language syntax? The most obvious question is whether this set of combinators is complete. To begin with, note that the linguistic combinatory rules, unlike the systems discussed in most of the literature cited above (but see Church 1940; Barendregt 1981, app. A; Hindley and Seldin 1986), are a *typed* combinatory system. That is to say, rules like the forward composition rule of chapter 3 are defined in terms of (variables over) the types of the domain and range of the input functions and the function that results. Indeed, the syntactic categories of a categorial grammar are precisely types, in that sense. So we are talking about completeness with respect to the simply typed λ^{τ}-calculi. Since mathematicians and computer scientists usually think of functions in this way, the typed λ-calculi are useful and interesting objects.

Interestingly, the paradoxical combinators such as Curry's **Y** and Smullyan's $\lambda x.xx$ are not definable in the typed systems. Since the existence of such fixed-point combinators is what allows the definition of recursion within the pure λ-calculus, recursive functions like *fact* cannot be defined within the pure λ^{τ}-calculi. There is also an interesting relation (discussed by Fortune, Leivant and O'Donnell 1983) to type systems in programming languages like PASCAL and ML.

Exactly the same correspondence holds between typed combinators and the typed λ-calculi as we have seen for the untyped versions. In particular, the **SK** system is complete with respect to the $\lambda^{\tau}_{\mathbf{K}}$-calculus. The **BCSI** system is similarly complete with respect to the $\lambda^{\tau}_{\mathbf{I}}$-calculus. Since the type-raising combinator **T** is equivalent to the combinatory expression **CI**, and since the linguistically observed set **BTS** includes **B** and **S**, it seems highly likely that **BTS** is related to **BCSI** and hence also to the $\lambda^{\tau}_{\mathbf{I}}$-calculus. Certainly **C** is definable in terms of **T** and **B**, as shown by Church (see Smullyan 1985, 113).[7]

The only qualification to the correspondence that I have been able to identify is that the combinator **I** itself does not appear in general to be definable in terms of **BTS**. A combinator corresponding to a special case of **I**, of type $(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta)$, can be defined as **CT**. This is not the true **I** combinator, for it will not map an atom onto itself. Nevertheless, **CT** constitutes the identity functional for first-order functions and all higher types, so this does not seem a very important deviation. We may assume that the $\lambda_{\mathbf{I}}^{\tau}$-calculus constitutes an upper bound on the expressive power of the **BTS** system and that the two are essentially equivalent.[8]

It follows immediately that all of the important constraints on the system as a theory of natural grammars stem from directional constraints imposed upon syntactic combinatory rules by the twin Principles of Consistency and Inheritance, discussed in chapter 4. This observation raises the further question of the expressive or automata-theoretic power of CCG.

8.3 Expressive Power

The way the Dutch cross-serial verb construction was captured in examples like (2) of chapter 6 suggests that CCG is of greater strong generative power than context-free grammar.[9] The Dutch construction intercalates the dependencies between arguments and verbs, rather than nesting them, and therefore requires this greater power, at least for strongly adequate capture. Whether standard Dutch can be shown on the basis of this construction not to be a weakly context-free *language* is of course another question. Huybregts (1984) and Shieber (1985) have shown that a related construction in related dialects of Germanic is not even weakly context-free. It is therefore clear that Universal Grammar has more than context-free power, and the further question of whether standard Dutch happens to exploit this power in a way that makes the *language* non-context-free (as opposed to the strongly adequate grammar) is of only technical interest.

The question is, how *much* more power do cross-serial dependencies demand and does CCG offer? An interesting class of languages to consider is the class of indexed grammars, which are discussed by Gazdar (1988) with reference to the Dutch construction. More recently Vijay-Shanker and Weir (1990, 1994) have argued that several apparently unrelated near-context-free grammar formalisms, including the present one, are weakly equivalent to the least powerful level of indexed grammars, the so-called linear indexed grammars.[10]

This section presents an informal version of their argument.

Indexed grammars (IGs) are grammars that, when represented as phrase structure rewriting systems, allow symbols on both sides of a production to be associated with features whose values are *stacks*, or unbounded pushdown stores. We can represent such rules as follows, where the notation [...] represents a stack-valued feature under the convention that the top of the stack is to the left, and where α and β are nonterminal symbols and W_1 and W_2 are strings of nonterminals and terminals, in the general case including nonterminals bearing the stack feature:

(11) $\alpha_{[...]} \quad \rightarrow \quad W_1 \ \beta_{[...]} \ W_2$

Such rules have the effect of passing a feature encoding arbitrarily many long-range dependencies from a parent α to one or more daughters β. The rules are allowed to make two kinds of modification to the stack value: an extra item may be "pushed" or added on top of the stack, or the topmost item already on the stack may be "popped" or removed. These two types of rule can be represented as similar schemata, as follows:

(12) a. "pushing:" $\alpha_{[...]} \quad \rightarrow \quad W_1 \ \beta_{[i,...]} \ W_2$
b. "popping:" $\alpha_{[i,...]} \quad \rightarrow \quad W_1 \ \beta_{[...]} \ W_2$

In general, IGs may include rules that pass stack-valued features to more than one daughter. The most restrictive class of indexed grammars, linear indexed grammars (LIGs), allows the stack-valued feature to pass to only one daughter; that is, W_1 and W_2 are restricted to strings of terminals and nonterminals *not* bearing the stack feature.

It is easy to show that Linear Indexed Grammar (LIG) offers a formalism that will express cross-serial dependencies. I will simplify the Dutch problem for illustrative purposes and assume that the goal is to generate a language whose strings all have some number of nouns on the left, followed by the same number of verbs on the right, with the dependencies identified by indices in the grammar. The following simple grammar (adapted from Gazdar 1988) will do this.

(13)
$$\begin{array}{llll} S_{[...]} & \rightarrow & n & S_{[v,...]} \\ S_{[...]} & \rightarrow & S'_{[...]} & \\ S'_{[v,...]} & \rightarrow & S'_{[...]} & v \\ S'_{[\,]} & \rightarrow & \varepsilon & \end{array}$$

The derivation tree for the string $n_1 \ n_2 \ n_3 \ v_1 \ v_2 \ v_3$ is shown in figure 8.1.

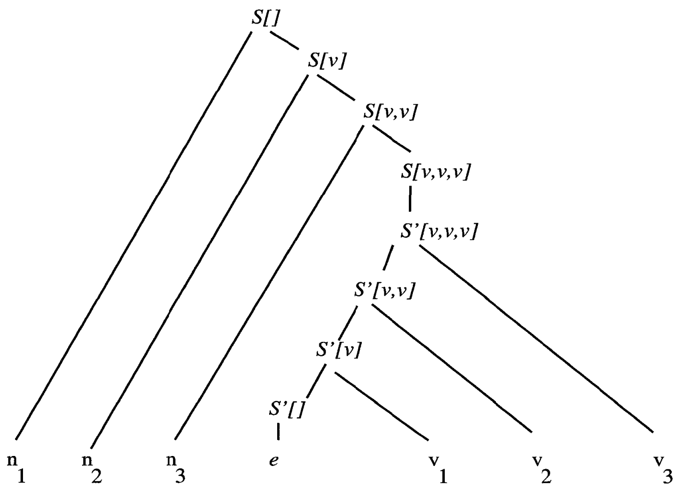

Figure 8.1
LIG derivation for n^3v^3

This is rather reminiscent of the structure produced by the (linguistically incorrect) CCG derivation using crossed composition but lacking type-raised categories shown in figure 6.2. While this particular grammar is weakly equivalent to a context free grammar (since a^nb^n is a context-free language, although a context-free grammar assigns different dependencies), it is equally easy to write a related grammar for the language $a^nb^nc^n$, which is not a context-free language.

Vijay-Shanker and Weir (1990, 1994) identify a characteristic automaton for these grammars, and show on the basis of certain closure properties that it defines what they call an "abstract family of languages" (AFL), just as the related pushdown automaton does. They provide polynomial time recognition and parsing results, of the order of n^6. These results crucially depend upon the linearity property, because it is this property that ensures that two branches of a derivation cannot share information about an unbounded number of earlier steps in the derivation (Vijay-Shanker and Weir 1994, 591–592). This fact both limits expressivity and permits efficient divide-and-conquer algorithms to apply.

Weir (1988) and Weir and Joshi (1988) were the first to observe that there is a close relation between linear indexed rules and the combinatory rules of CCG. Function categories like *give* and *zag helpen voeren* can be equated with indexed categories bearing stack-valued features, as follows:

(14) give := $(VP/NP_2)/NP_1 \equiv VP_{[NP_1,NP_2]}$
zag helpen voeren := $(((S\backslash NP_4)\backslash NP_3)\backslash NP_2)\backslash NP_1 \equiv S_{[NP_1,NP_2,NP_3,NP_4]}$

Note that the LIG categories no longer encode directionality—it is up to the LIG rules to do that.

Combinatory rules can be translated rather directly in terms of such categories into sets of LIG productions of the form shown on the right of the equivalences in (15) and (16). Since LIG categories do not capture directionality, the grammar for a particular language will be made up of more specific instances of these schemata involving just those categories that do in fact combine in the specified order for that language.[11]

(15) $X/Y \quad Y \quad \Rightarrow \quad X \quad \equiv \quad X'_{[\ldots]} \quad \rightarrow \quad X'_{[Y,\ldots]} \quad Y_{[\,]}$

(16) $X/Y \quad Y/Z \quad \Rightarrow \quad X/Z \quad \equiv \quad X'_{[Z,\ldots]} \quad \rightarrow \quad X'_{[Y,\ldots]} \quad Y_{[Z]}$

Rule (15) is forward application, realized as a binary LIG rule of the "push" variety. Rule (16) is first-order forward composition, **B**, and involves both pushing a *Y* and popping a *Z*. Crucially, the stack, represented as ..., is passed to only one daughter. The same is true for the substitution rule:

(17) $Y/Z \quad (X\backslash Y)/Z \quad \Rightarrow \quad X/Z \quad \equiv \quad X'_{[Z,\ldots]} \quad \rightarrow \quad Y_{[Z]} \quad X'_{[Z,Y,\ldots]}$

The same linearity property also holds for the rules corresponding to $\mathbf{B}^2, \mathbf{B}^3$ and so on, because the set of arguments of the function into *Y* is bounded. It would *not* hold for an unbounded schema for a rule corresponding to $\mathbf{B}^n$. This rule, which can be written in the present notation as follows, involves *two* stack-valued features, written $\ldots_1$ and $\ldots_2$:

(18) $X/Y \quad Y/Z\$ \quad \Rightarrow \quad X/Z\$ \quad \equiv \quad X'_{[\ldots_1,Z,\ldots_2]} \quad \rightarrow \quad X'_{[Y,\ldots_2]} \quad Y_{[\ldots_1,Z]}$

It is not currently known precisely what strong generative power such generalized rules engender. They may not take us to the full power of IGs, because the translation from CCG forces us to regard the left-hand side of the rule as bearing a *single* stack feature, which the production nondeterministically breaks into two stack fragments that pass to the daughters. This is not the same as passing the same stack to two daughters—crucially, the two branches of the derivation that it engenders do not share any information, and therefore seem likely to permit efficient divide-and-conquer parsing techniques.

Weir and colleagues treat type-raising as internal to the lexicon, rather than as a rule in syntax, However, Hoffman (1993, 1995b) has pointed out that a similar increase in power over LIG follows from the involvement of type-raised categories like $T/(T\backslash NP)$ if T is regarded as a true variable, rather than a finite schematization. To a first approximation, the indexed category corresponding to a type-raised category looks like this:

(19) $T/(T\backslash Y) \equiv X'_{[X'_{[Y,\ldots 1]},\ldots 1]}$

Again, the LIG category does not capture the order information, and in particular the order-preserving character, of the original. That has to be captured in the LIG productions, in such facts as that for every instance of rule (15) there is a rule like the following:

(20) $X'_{[\ldots 1]} \quad \rightarrow \quad X'_{[X'_{[Y,\ldots 1]},\ldots 1]} \quad X'_{[Y,\ldots 1]}$

The LIG equivalent of a raised category has two copies of the stack $\ldots_1$. However, as far as functional application goes, it is simply a function like any other—that is, an instance of $\alpha_{[\beta,\ldots]}$. It follows that this rule is simply another instance of (15). Again, no information is shared across branches of the derivation.

However, by the same reasoning, when *two* of the raised categories compose, even via the first-order composition rule (16), so that Y is $T_{[X,\ldots]}$, their two distinct stack variables give rise to a nonlinear production, as follows:

(21) $X_{X_{[Z,\ldots 1,\ldots 2]}} \quad \rightarrow \quad X_{[X_{[Y,\ldots 2]},\ldots 2]} \quad X_{[Z,X_{[Y,\ldots 1]},\ldots 1]}$

This composition has the characteristic noted earlier of nondeterministically partitioning a single stack feature on the left-hand side into two fragments, passed as stack features to the daughters. In effect, the variable transforms bounded composition into the unbounded variety. Again no information is shared across the two branches of the derivation.

Hoffman shows how the language $a^nb^nc^nd^ne^n$ (which is outside the set of linear-indexed languages) can be defined by exploiting this behavior of variables in type-raised categories. It is therefore known that if this property is allowed in CCGs, it raises their power strictly beyond LIG. What is not currently known is how *much* beyond LIG it takes us, or whether CCLs so defined are a subset of IL, the full set of indexed languages.

Alternatively, we can, as suggested earlier, confine ourselves to LIG power by eschewing the general interpretation of composition and type-raising and by interpreting the variables involved in each as merely finite schemata. Such a limitation allows all derivations encountered in parts I and II of the book and keeps CCG weakly equivalent to LIG.[12]

The advantages of keeping to such a limitation are potentially important, as Vijay-Shanker and Weir (1990, 1994) show. As noted earlier, because LIGs pass the stack to only one branch, they limit expressive power and allow efficient algorithms to apply. As a result, Vijay-Shanker and Weir have been able to demonstrate polynomial worst-case complexity results for recognizing and parsing CCGs and TAGs, which are also weakly equivalent to LIGs. It is currently unclear whether similar advantages obtain for the more general class of CCGs. The important fact that neither generalised composition nor variables in type-raised categories pass any one stack feature to more than one daughter gives reason to suppose that they too may be polynomially recognizable using divide-and-conquer techniques.

As Gazdar (1988) has pointed out, it is not clear that the linguistic facts allow us to keep within either of these bounds. The full generality of the Dutch verb-raising construction in noncoordinate sentences can be captured with weakly LIG-equivalent rules, but they allow functions of arbitrarily high valency to be grown. If such functions can coordinate, then we need the full power of IG. This result follows immediately from the fact that the unbounded coordination combinator Φ^n corresponds to a production that passes the same stack feature to two daughters:

(22) $X_{[\ldots 1]} \quad \rightarrow \quad X_{[\ldots 1]} \quad CONJ \quad X_{[\ldots 1]}$

The crucial cases for Dutch are those in which unboundedly long sequences of nouns or verbs of unbounded valency coordinate. However, once the valency or number of arguments gets beyond four, the limit found in the Dutch and English lexicon, the sentences involved become increasingly hard to process, and hard to judge.

Rambow (1994a) makes a similar argument for the translinear nature of scrambling in German. However, this argument depends on the assumption that unbounded scrambling is complete to unbounded depth of embedding. Because these sentences also go rapidly beyond anything that human processors can handle, any argument that either kind of sentence is grammatical depends on assumptions about what counts as a "natural generalization" of the construction, parallel to a famous argument of Chomsky's (1957) concerning the non-finite-state nature of center embedding.

Joshi, Rambow and Becker (to appear) have made the point that this analogy may not hold. They note that all such arguments—including Chomsky's—fall if a lesser automaton or AFL that covers all and only the acceptable cases is ever shown to exist. The status of any residual marginal cases is then decided by that automaton. It is only because no one has yet identified such a finite-state automaton that Chomsky's claim that context-free grammars constitute a lower bound on competence still stands, and is unlikely ever to be overthrown.[13]

It follows that if LIG alone can be shown to be of sufficient power to provide strongly adequate grammars for the core examples, or alternatively if unbounded composition rules and variable-based type-raising are indeed of lesser power than IG, and if a class of automata characterizing an AFL can be identified, the question of whether that lesser power provides an upper bound on natural complexity comes down to the question of whether some exceedingly marginal coordinations and scramblings are acceptable or not. If an automaton exists that is strongly adequate to recognize all and only the sentences that we are certain about, then we might well let that fact decide the margin, in the absence of any other basis for claiming a natural generalization. This is a question for further research, but however it turns out, CCG should be contrasted in this respect with multimodal type-logical approaches of the kind reviewed by Moortgat (1997), which Carpenter (1995) shows to be much less constrained in automata-theoretic terms.

8.4 Formalizing Directionality in Categorial Grammars

In chapter 4, I claimed that the Principles of Adjacency, Consistency, and Inheritance are simple and natural restrictions for rules of grammar. In chapters 6 and 7, I claimed that a number of well-known crosslinguistic universals follow from them. We have just seen that low automata-theoretic power and a polynomial worst-case parsing complexity result also follow from these principles. So quite a lot hinges on the claim that these principles *are* natural and nonarbitrary.

The universal claim further depends upon type-raising's being limited (at least in the case of configurational languages) to the following schemata:[14]

(23) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}\backslash X)$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}\backslash(\mathrm{T}/X)$

If the following patterns (which allow constituent orders that are not otherwise permitted) were allowed, the regularity would be unexplained. In the absence of further restrictions, grammars would collapse into free order:

(24) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X)$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}\backslash(\mathrm{T}\backslash X)$

But what are the principles that limit combinatory rules of grammar, to include (23) and exclude (24)? And how can we move type-raising into the lexicon without multiplying NP categories unnecessarily?

The intuition here is that *we want to make type-raising sensitive to the directionality of the lexically defined functions that it combines with.* However, the solution of combining type-raising with the other combination rules proposed by Gerdeman and Hinrichs (1990) greatly expands their number.[15]

The fact that directionality of arguments is inherited under the application of combinatory rules, according to the Principle of Inheritance, strongly suggests that directionality is a property of arguments themselves, just like their categorial type, *NP* or whatever, as suggested in Steedman 1987, and as in Zeevat, Klein and Calder 1987 and Zeevat 1988.

Our first assumption about the nature of such a system might exploit a variant of the notation used in the discussion of LIGs above (cf. Steedman 1987), in which a binary feature marks an argument of a function as "to the left" or "to the right." In categorial notation it is convenient to indicate this by subscripting the symbol $\leftarrow$ or $\rightarrow$ to the argument in question. Since the slash in a function will now be nondirectional, both $\backslash$ and $/$ can be replaced by a single nondirectional slash, also written $/$, so that for example the transitive verb category is written as follows:[16]

(25) enjoys := $(S/NP_{\leftarrow})/NP_{\rightarrow}$

(The result S has no value on this feature until it unifies with a function as its argument, so it bears no directional indication. It is just an unbound variable.)

In this notation the (noncrossed) forward composition rule is written as follows:

(26) *Forward composition*
$X/Y_{\rightarrow} \quad Y/Z_{\rightarrow} \quad \Rightarrow_{\mathbf{B}} \quad X/Z_{\rightarrow} \quad (>\mathbf{B})$

The forbidden rule (6) of chapter 4, which violates the Principle of Inheritance, would be written as follows:

(27) $X/Y_{\rightarrow} \quad Y/Z_{\rightarrow} \quad \not\Rightarrow \quad X/Z_{\leftarrow}$

However, given the definition of directionality as a feature of Z, this is not a rule of composition at all. As long as the combinatory rules are limited to operations like composition, only rules obeying the Principle of Inheritance are permitted.

The feature in question does not have the equally desirable effect of limiting type-raising rules to the order-preserving kind in (23). Those rules are now written as follows:

(28) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{\leftarrow})_{\rightarrow}$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{\rightarrow})_{\leftarrow}$

Since the input to the rule, X, is unmarked on this feature, there is nothing to stop us from writing the order-*changing* rules in (24):

(29) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{\rightarrow})_{\rightarrow}$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{\leftarrow})_{\leftarrow}$

This is very bad. Although we can easily exclude the latter rules to define grammars for languages like English, we could with equal ease exploit the same degree of freedom to define a language in which *only* order-changing type-raising is allowed, so that the directionality of functions in the lexicon would be systematically overruled. Thus, we could have a language with an SVO lexicon, but OVS word order. Worse still, we could equally well have a language with one of each kind of type-raising rule—say, with an SVO lexicon but a VOS word order. Such languages seem unreasonable, and would certainly engender undesirably cynical attitudes toward life in any child faced with the task of having to acquire them.

Zeevat, Klein and Calder (1987) and Zeevat (1988) offer an ingenious, but partial, solution to this problem. Of the two sets of rules (28) and (29), it is actually the order-*changing* pair in (29) that looks most reasonable, in that the raised function can at least be held to inherit the *same* directionality as its argument. That is, both rules are instances of a schema in which the directionality value is represented as a variable, say, D. In the present notation they can both be conveniently represented by the following single rule:

(30) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{D})_{D}$

This rule has the attractive properties of being able to combine with either rightward- or leftward-combining arguments and of inheriting its own directionality from them. Since it therefore *only* combines to the left with leftward arguments and to the right with rightward ones, it offers a way around the problem of having multiple type-raised categories for arguments. We can simply apply this rule across the board to yield one type for NPs. In fact, we can do this off-line, in the lexicon, as Zeevat, Klein, and Calder propose.

However, there is a cost in theoretical terms. As noted earlier, since this is the direction-changing rule, the lexicon must reverse the word order of the language. An SVO language like English must have an OVS lexicon. This is in fact what Zeevat, Klein and Calder (1987) propose (see Zeevat 1988, 207–210).

Despite this disadvantage, there is something very appealing about this proposal. It would be very nice if there were a different treatment of the directionality feature that preserved its advantages without implicating this implausible assumption about the lexicon. Of course, as a technical solution we might encode the values $\rightarrow$ and $\leftarrow$ as list structures $[0,1]$ and $[1,0]$, and write a similar single order-preserving rule as follows, using variables over the elements:

(31) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X_{[x,y]})_{[y,x]}$

But such a move explains nothing, for we could equally well exploit this device to write the order-changing rule or, by using constants rather than variables, define any mixture of the two. What is wrong is that directionality is being represented as an abstract feature, without any grounding in the properties of the string itself. If instead we define the feature in question in terms of string positions, in a manner that is familiar from the implementation of definite clause grammars (DCGs) in logic programming, we can attain a more explanatory system, in which the following results emerge:

1. The Principle of Inheritance is explained as arising from inheritance of this feature under unification of categories.
2. A single order-preserving type-raised category combining either to the right or to the left can be naturally specified.
3. No comparable single order-*changing* type-raised category can be specified (although a completely order-free category can).

Since these matters are somewhat technical, and since they impinge very little upon linguistics, this whole discussion is relegated to an appendix to the present chapter, which many readers may wish to skip entirely. Since the notation becomes quite heavy going, it is emphasised here that *it is not a proposal for a new CCG notation.* It is a semantics for the metagrammar of the *present* CCG notation.

Appendix: Directionality as a Feature

This appendix proposes an interpretation, grounded in string positions, for the symbols / and \ in CCG. This interpretation is easiest to present using unification as a mechanism for instantiating underspecified categories and feature value bundles, a mechanism that has been implicit at several points in the earlier discussion.

For a full exposition of the concept of unification, the reader is directed to Shieber 1986. The intuition behind the notion is that of an operation that amalgamates compatible terms and fails to amalgamate incompatible ones. The result of amalgamating two compatible terms is the most general term that is an instance of both the original terms. For example, the following pairs of terms unify, to yield the results shown:

(32)
$$\begin{array}{lll} x & a' & \Longrightarrow\ a' \\ f'(g'a') & x & \Longrightarrow\ f'(g'a') \\ f'x & f'(g'y) & \Longrightarrow\ f'(g'y) \\ f'a'x & f'yy & \Longrightarrow\ f'a'a' \end{array}$$

The following pairs of terms do not unify:

(33)
$$\begin{array}{lll} a' & b' & \Longrightarrow\ \mathit{fail} \\ f'x & g'y & \Longrightarrow\ \mathit{fail} \\ f'a'b' & f'yy & \Longrightarrow\ \mathit{fail} \end{array}$$

(Constants are distinguished from variables in these terms by the use of primes.)

Besides providing a convenient mechanism for number and person agreement, unification-based formalisms provide a convenient way of implementing combinatory rules in which X, Y, and so on, can be regarded as variables over categories that can be instantiated or given values by unification with categories like NP or $S\backslash NP$. This observation provides the basis for a transparent implementation of CCG in the form of definite clause grammar (DCG; Pereira and Warren 1980) in programming languages like Prolog (and its higher-order descendant λ-Prolog), and in fact such an implementation has been implicit at a number of points in the exposition above—for instance in the discussion of agreement in chapter 3. A example of a simple (but highly inefficient) program of this kind for use as a proof checker for the feature-based account of directionality that follows is given in Steedman 1991c.

The unification-based implementation has the important attraction of forcing the Principle of Combinatory Type Transparency to apply to combinatory rules interpreted in this way, because of the resemblance between a model-theoretic semantics for unification and the set-theoretic representations of categories (see van Emden and Kowalski 1976; Stirling and Shapiro 1986; Miller 1991, 1995).

One form of DCG equivalent of CFPSG rewrite rules like (34a) is the Prolog inference rule (34b), in which `:-` is the Prolog leftward logical implication operator, and `P0, P1, P` are variables over string positions. such as the positions 1, 2, and 3, in (34c) (see Pereira and Shieber 1987 for discussion):

(34) a. $S \rightarrow NP\ VP$
b. `s(P0,P) :- np(P0,P1),vp(P1,P).`
c. $_1$ dexter $_2$ walks $_3$

The Prolog clause (34b) simply means that there is a sentence between two string positions `P0` and `P` if there is an NP between `P0` and some other position `P1`, and a VP between the latter position and `P`. This device achieves the effect of declarativizing string position and has the advantage that, if lists are used to represent strings, the Prolog device of difference-list encoding can be used to represent string position implicitly, rather than explicitly as in (34c) (see Pereira and Warren 1980; Stirling and Shapiro 1986).

The basic form of a combinatory rule under the Principle of Adjacency is $\alpha\ \beta \Rightarrow \gamma$. However, this notation leaves the linear order of α and β implicit. We therefore temporarily expand the notation, replacing categories like *NP* by 4-tuples, of the form $\{\alpha, DP_\alpha, L_\alpha, R_\alpha\}$, comprising (a) a *type* such as *NP*; (b) a *distinguished position*, which we will come to in a minute; (c) a *left-end position*; and (d) a *right-end position*. The latter two elements are the exact equivalent of the DCG positional variables.

The Principle of Adjacency then finds expression in the fact that all legal combinatory rules must have the form in (35), in which the right-end of α is the same as the left-end of β:

(35) $\{\alpha, DP_\alpha, P_1, P_2\}\ \ \{\beta, DP_\beta, P_2, P_3\} \Rightarrow \{\gamma, DP_\gamma, P_1, P_3\}$

I will call the position P_2, to which the two categories are adjacent, the "juncture."

The distinguished position of a category is simply the one of its two ends that coincides with the juncture when it is the "canceling" term *Y*, which from now on we can refer to as the "juncture term" in a combination. A rightward combining function, such as the transitive verb *enjoy*, specifies the distinguished position of its argument (here underlined for salience) as being that argument's *left*-end. So this category is written in full as in (36a), using a *nondirectional* slash /:

(36) a. enjoy $:= \{\{VP, DP_{vp}, L_{vp}, R_{vp}\}/\{NP, \underline{L_{np}}, \underline{L_{np}}, R_{np}\}, DP_{verb}, L_{verb}, R_{verb}\}$
b. enjoy $:= \{VP/\{NP, \underline{L_{np}}, \underline{L_{np}}, R_{np}\}, \_, L_{verb}, R_{verb}\}$

The notation in (36a) is rather overwhelming. When positional features are of no immediate relevance in such categories, they will be suppressed, either by representing the whole category by a single symbol or by representing anonymous variables whose identity and binding is of no immediate relevance as "_."[17] For example, when we are thinking of such a function *as* a function, rather than as an argument, we will write it as in (36b), where *VP* stands for $\{VP, DP_{vp}, L_{vp}, R_{vp}\}$ and the distinguished position of the verb is written _. It is important to note that although the binding of the NP argument's distinguished position to its left-end L_{np} means that *enjoy* is a rightward function, the distinguished position is *not* bound to the actual right-end of the verb, R_{verb}, as in the following version of (36b):

(37) *enjoy := $\{VP/\{NP, \underline{R_{verb}}, \underline{R_{verb}}, R_{np}\}, \_, L_{verb}, \underline{R_{verb}}\}$

It follows that the verb can potentially combine with an argument elsewhere, just so long as it is to the right. This property was crucial to the earlier analysis of heavy NP shift. Coupled with the parallel independence in the position of the result from the position of the verb, it is the point at which CCG parts company with the directional Lambek calculus, as we will see.

In the expanded notation the rule of forward application is written as follows:

(38) $\{\{X,DP_x,P1,P3\}/\{Y,P2,P2,P3\},\_,P1,P2\} \quad \{Y,P2,P2,P3\} \quad \Rightarrow \quad \{X,DP_x,P1,P3\}$

The fact that the distinguished position must be one of the two ends of an argument category, coupled with the requirement of the Principle of Adjacency, means that *only* the two order-preserving instances of functional application can exist, and only consistent categories can unify with those rules.

A combination under this rule proceeds as follows. Consider example (39), the VP *enjoy musicals*. (In this example the elements are words, but they could be any constituents.)

(39)
$$\begin{array}{lclcl} 1 & \text{enjoy} & 2 & \text{musicals} & 3 \\ & \{VP/\{NP, L_{arg}, L_{arg}, R_{arg}\}, \_, L_{fun}, R_{fun}\} & & \{NP, DP_{np}, L_{np}, R_{np}\} & \end{array}$$

The derivation continues as follows. First the positional variables of the categories are bound by the positions in which the words occur in the string, as in (40), which in the first place we will represent explicitly, as numbered string positions:[18]

(40)
$$\begin{array}{lclcl} 1 & \text{enjoy} & 2 & \text{musicals} & 3 \\ & \{VP/\{NP, L_{arg}, L_{arg}, R_{arg}\}, \_, 1, 2\} & & \{NP, DP_{np}, 2, 3\} & \end{array}$$

Next the combinatory rule (38) applies, to unify the argument term of the function with the real argument, binding the remaining positional variables including the distinguished position, as in (41) and (42):

(41)
$$\begin{array}{ccccc} 1 & \text{enjoy} & 2 & \text{musicals} & 3 \\ & \{VP/\{NP,L_{arg},L_{arg},R_{arg}\},\_,1,2\} & & \{NP,DP_{np},2,3\} & \\ & \{X/\{Y,P2,P2,P3\},\_,P1,P2\} & & \{Y,P2,P2,P3\} & \end{array}$$

(42)
$$\begin{array}{ccccc} 1 & \text{enjoy} & 2 & \text{musicals} & 3 \\ & \{VP/\{NP,2,2,3\},\_,1,2\} & & \{NP,2,2,3\} & \\ \hline & \multicolumn{3}{c}{\{VP,1,3\}} & \end{array}$$

At the point when the combinatory rule applies, the constraint implicit in the distinguished position must actually hold. That is, the distinguished position must be adjacent to the functor.

Thus, the Consistency property of combinatory rules follows from the Principle of Adjacency, embodied in the identification of the distinguished position of the argument terms with the juncture $P2$, the point to which the two combinands are adjacent, as in the application example (38).

The Principle of Inheritance also follows directly from these assumptions. The fact that rules correspond to combinators like composition forces directionality to be inherited, like any other property of an argument such as being an NP. It follows that only instances of the two very general rules of composition shown in (43) are allowed, as a consequence of the three principles:

(43) a. $\{\{X,DP_x,L_x,R_x\}/\{Y,P2,P2,R_y\},\_,P1,P2\}\ \ \{\{Y,P2,P2,R_y\}/\{Z,DP_z,L_z,R_z\},\_,P2,P3\}$
$\Rightarrow_{\mathbf{B}}\ \ \{\{X,DP_x,L_x,R_x\}/\{Z,DP_z,L_z,R_z\},\_,P1,P3\}$

b. $\{\{Y,P2,L_y,P2\}/\{Z,DP_z,L_z,R_z\},\_,P1,P2\}\ \ \{\{X,DP_x,L_x,R_x\}/\{Y,P2,L_y,P2\},\_,P2,P3\}$
$\Rightarrow_{\mathbf{B}}\ \ \{\{X,DP_x,L_x,R_x\}/\{Z,DP_z,L_z,R_z\},\_,P1,P3\}$

To conform to the Principle of Consistency, it is necessary that L_y and R_y, the ends of the canceling category Y, be distinct positions—that is, that Y not be coerced to the empty string. This condition has always been explicit in the Principle of Adjacency (Steedman 1987, 405, and see above), although in any Prolog implementation such as that in Steedman 1991c it has to be explicitly imposed. These schemata permit only the four instances of the rules of composition proposed in Steedman 1987 and Steedman 1990, and chapter 4, repeated here as (44) in the basic CCG notation:

(44) *The possible composition rules*

a. $X/Y \quad Y/Z \quad \Rightarrow_{\mathbf{B}} \quad X/Z$ $\quad (>\mathbf{B})$

b. $X/Y \quad Y\backslash Z \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z$ $\quad (>\mathbf{B}_\times)$

c. $Y\backslash Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z$ $\quad (<\mathbf{B})$

d. $Y/Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X/Z$ $\quad (<\mathbf{B}_\times)$

"Crossed" rules like (44b,d) are still allowed (because of the nonidentity noted in the discussion of (36) between the distinguished position of arguments of functions and the position of the function itself). They are distinguished from the corresponding noncrossing rules by further specifying DP_z, the distinguished position on Z.[19] However, no rule violating the Principle of Inheritance, like (27), is allowed: such a rule would require a *different* distinguished position on the two Zs and would therefore not be functional composition at all.[20] This is a desirable result: as shown in the earlier chapters, the non-order-preserving instances (44b, d) are required for the grammar of English and Dutch. In configurational languages like English they must of course be carefully restricted with regard to the categories that may unify with Y.

The implications of the present formalism for the type-raising rules are less obvious. Type-raising rules are unary, and probably lexical, so the Principle of Adjacency does not obviously apply. However, as noted earlier, we only want the *order-preserving* instances (23), in which *the directionality of the raised category is the reverse of that of its argument.* But how can this reversal be anything but an arbitrary property?

Because the directionality constraints are defined in terms of string positions, the distinguished position of the subject argument of a predicate *walks*—that is, the right-edge of that subject—is equivalent to the distinguished position of the predicate that constitutes the argument of an order-preserving raised subject *Dexter*—that is, the *left*-edge of that predicate. It follows that both of the order-preserving rules are instances of the single rule (45) in the extended notation:

$$\text{(45)}\quad \underline{\{X, DP_{arg}, L_{arg}, R_{arg}\}}$$
$$\Rightarrow \{\mathrm{T}/\{\mathrm{T}/\underline{\{X, \underline{DP_{arg}}, \underline{L_{arg}}, \underline{R_{arg}}\}}, \underline{DP_{arg}}, L_{pred}, R_{pred}\}, -, \underline{L_{arg}}, \underline{R_{arg}}\}$$

The crucial property of this rule, which forces its instances to be order-preserving, is that *the distinguished-position variable DP_{arg} on the argument of the predicate in the raised category is the same as that on the argument of the raised category itself.* (The two distinguished positions are underlined in (45).) Notice that this choice forces the raised NP and its argument to be string adjacent; it is exactly the opposite choice from the one that we took in allowing lexical categories like (36) to unify with arguments anywhere in the specified direction.[21] Of course, the position is unspecified at the time the rule applies, and it is simply represented as an unbound unification variable with an arbitrary mnemonic identifier. However, when the category combines with a predicate, this variable will be bound by the directionality specified in the predicate itself. Since this condition will be transmitted to the raised category, *it will have to coincide with the juncture of the combination.* Combination of the categories in the nongrammatical order will therefore fail, just as if the original categories were combining without the mediation of type-raising.

Consider the following example. Under rule (45), the categories of the words in the sentence *Dexter walks* are as shown in (46), before binding.

(46)
$$\begin{array}{lcr}
1 \quad \text{Dexter} \quad 2 & & \text{walks} \quad 3 \\
\{S/\{S/\{NP,DP_g,L_g,R_g\},DP_g,L_{pred},R_{pred}\},\_,L_g,R_g\} & & \{S/\{NP,R_{np},L_{np},R_{np}\},DP_w,L_w,R_w\}
\end{array}$$

Binding of string positional variables yields the categories in (47).

(47)
$$\begin{array}{lcr}
1 \quad \text{Dexter} \quad 2 & & \text{walks} \quad 3 \\
\{S/\{S/\{NP,DP_g,1,2\},DP_g,L_{pred},R_{pred}\},\_,1,2\} & & \{S/\{NP,R_{np},L_{np},R_{np}\},DP_w,2,3\}
\end{array}$$

The combinatory rule of forward application (38) applies as in (48), binding further variables by unification. In particular, DP_g, R_{np}, DP_w, and $P2$ are all bound to the juncture position 2, as in (49):

(48)
$$\begin{array}{cc}
1 \quad \text{Dexter} \quad 2 & \text{walks} \quad 3 \\
\{S/\{S/\{NP,DP_g,1,2\},DP_g,L_{pred},R_{pred}\},\_,1,2\} & \{S/\{NP,R_{np},L_{np},R_{np}\},DP_w,2,3\} \\
\{X/\{Y,P2,P2,P3\},\_,P1,P2\} & \{Y,P2,P2,P3\}
\end{array}$$

(49)
$$\begin{array}{cc}
1 \quad \text{Dexter} \quad 2 & \text{walks} \quad 3 \\
\{S/\{S/\{NP,2,1,2\},2,2,3\},1,2\} & \{S/\{NP,2,1,2\},2,2,3\} \\
\hline
\multicolumn{2}{c}{\{S,1,3\}}
\end{array}$$

By contrast, the same categories in the opposite linear order fail to unify with any combinatory rule. In particular, the backward application rule fails, as in (50):

(50)
$$\begin{array}{cc}
1 \quad {*}\text{Walks} \quad 2 & \text{Dexter} \quad 3 \\
\{S/\{NP,R_{np},L_{np},R_{np}\},\_,1,2\} & \{S/\{S/\{NP,DP_g,2,3\},DP_g,L_{pred},R_{pred}\},\_,2,3\} \\
\{Y,P2,P1,P2\} & \{X/\{Y,P2,P1,P2\},\_,P2,P3\}
\end{array}$$

(Combination is blocked because 2 cannot unify with 3.)

On the assumption implicit in (45), the only permitted instances of type-raising are the two rules given earlier as (23). The earlier results concerning word order universals under coordination are therefore captured. Moreover, we can now think of these two rules as a single underspecified order-preserving rule directly corresponding to (45), which we might write less long-windedly as follows, augmenting the original simplest notation with a vertical "order-preserving" slash | to distinguish it from the undifferentiated nondirectional slash /:

(51) *The Order-preserving type-raising rule*

$$X \quad \Rightarrow_{\mathbf{T}} \quad \mathrm{T}|(\mathrm{T}|X)$$

The category that results from this rule can combine in either direction, but will always preserve order. Such a property is extremely desirable in a language like English, whose verb requires some arguments to the right and some to the left, but whose NPs do not bear case. The general raised category can combine in both directions, but will still preserve word order. Like Zeevat's (1988) rule, it thus eliminates what was earlier noted as a worrying extra degree of categorial ambiguity. As under that proposal, the way is now clear to incorporate type-raising directly into the lexicon, substituting categories of the form $\mathrm{T}|(\mathrm{T}|X)$, where X is a category like *NP* or *PP*, directly into the lexicon in place of the basic categories, or (more readably, but less efficiently), to keep the basic categories and the rule (51), and exclude the base categories from all combination. Most importantly, we avoid Zeevat's undesirable assumption that the English lexicon is OVS, thus ensuring continued good relations with generations of language learners to come.

Although the order-preserving constraint is very simply imposed, it is in one sense an additional stipulation, imposed by the form of the type-raising rule (45). We could have used a unique variable—say, DP_{pred}—in the crucial position in (45), unrelated to the positional condition DP_{arg} on the argument of the predicate itself, to define the distinguished position of the predicate-argument of the raised category, as in (52):

(52) $*\{X, DP_{arg}, L_{arg}, R_{arg}\} \Rightarrow$
$$\{\mathrm{T}/\{\mathrm{T}/\{X, \underline{DP_{arg}}, L_{arg}, R_{arg}\}, \underline{DP_{pred}}, L_{pred}, R_{pred}\}, -, L_{arg}, R_{arg}\}$$

However, this tactic would yield a completely unconstrained type-raising rule, whose result category could not merely be substituted throughout the lexicon for ground categories like *NP* without grammatical collapse. (Such categories immediately induce totally free word order—for example, permitting (50) on the English lexicon.)

Although it is conceivable that such non-order-preserving type-raised categories might figure in grammars for extremely nonconfigurational languages, such languages are usually characterized by the presence of *some* fixed elements. It seems likely that type-raising is universally confined to the order-preserving kind and that the sources of so-called free word order lie elsewhere.[22]

Such a constraint can therefore be understood in terms of the present proposal simply as a requirement for the lexicon itself to be consistent. It should also be observed that a single uniformly order-*changing* category of the kind proposed by Zeevat (1988) is not possible under this theory.

That is not to say that more specific order-changing categories cannot be defined in this notation. As noted earlier, in a non-verb-final language such as English the object relative-pronoun must have the category written in the basic notation as $(N\backslash N)/(S/NP)$, which is closely related to a type-raised category. In the extended notation, and abbreviating $N\backslash N$ as R, this category is the following:

(53) whom := $\{R/\{\mathrm{T}/\{X, \underline{L_{arg}}, L_{arg}, R_{arg}\}, \underline{L_{pred}}, L_{pred}, R_{pred}\}, \text{–}, L_{arg}, R_{arg}\}$

In fact, provided we constrain forward crossed composition correctly, as we must for any grammar of English, the following slightly less specific category will do for the majority dialect of English in which there is no distinction between subject and object relative-pronouns *who*, or for the un-case-marked relative-pronoun *that*:

(54) who/that := $\{R/\{\mathrm{T}/\{X, \underline{DP_{arg}}, L_{arg}, R_{arg}\}, \underline{L_{pred}}, L_{pred}, R_{pred}\}, \text{–}, L_{arg}, R_{arg}\}$

In the former category both the complement function and its argument are specified as being on the right. In the latter, the directionality of the complement argument is unspecified. Thus, we need look no further than the relative-pronouns of well-attested dialects of English to see exploited in full all the degrees of freedom that the theory allows us to specify various combinations of order-preserving and non-order-preserving type-raising in a single lexical category.

The account of pied-piping proposed by Szabolcsi (1989), to which the reader is directed for details, is also straightforwardly compatible with the present proposal.[23]

Chapter 9

Processing in Context

[After the second word of Tom wanted to ask Susan to bake a cake*] we have in the semantics a function, which we might call (*Tom want*). . . . If the parser is forced to make a choice between alternative analyses, it may make reference in this choice to semantics.*
John Kimball, "Predictive Analysis and Over-the-Top Parsing"

To account for coordination, unbounded dependency, and Intonation Structure, strictly within the confines of the Constituent Condition on Rules, we have been led in parts I and II of the book to a view of Surface Structure according to which strings like *Anna married* and *thinks that Anna married* are constituents in the fullest sense of the term. As we have repeatedly observed, it follows that they must also potentially be constituents of noncoordinate sentences like *Anna married Manny* and *Harry thinks that Anna married Manny*. For moderately complex sentences there will in consequence be a large number of nonstandard alternative derivations for any given reading.

We should continue to resist the natural temptation to reject this claim out of hand on the grounds that it is at odds with much linguistic received opinion. We have already seen in earlier chapters that on many tests for constituency—for example, the list cited in (1) of chapter 2—the combinatory theory does better than most. The temptation to reject the proposal on the basis of parsing efficiency should similarly be resisted. It is true that the presence of such semantic equivalence classes of derivations engenders rather more nondeterminism in the grammar than may have previously been suspected. Although this makes writing parsers a little less straightforward than might have been expected, it should be clear that this novel form of nondeterminism really is a property of English and all other natural languages and will be encountered by any theory with the same coverage with respect to coordination and intonational phenomena. It is also worth remembering that natural grammars show no sign of any pressure to minimize nondeterminism elsewhere in the grammar. There is therefore no a priori reason to doubt the competence theory on these grounds.

The only conclusion we can draw from the profusion of grammatical nondeterminism is that the mechanism for coping with it must be very powerful.

This chapter will argue that the most important device for dealing with nondeterminism in the human processor is a process of eliminating partial analyses whose interpretation is inconsistent with knowledge of the domain under discussion and the discourse context.

The chapter will also claim that combinatory grammars are particularly well suited to the incremental, essentially word-by-word assembly of semantic interpretations, for use with this "interactive" parsing tactic. Any attempt to argue for the present theory of competence and against any other on the basis of this last observation alone would be fallacious. The methodological priority of competence arguments remains unassailable, and none of the theories currently on offer, including this one, have yet come close to descriptive adequacy as competence theories. Since all of them are compatible in principle with incremental interpretation in this sense of the term, all bets are off until the question of descriptive adequacy has been settled.

Nevertheless, we can draw the following weaker conclusion. If the program sketched in this book is ultimately successful, and CCG, together with the view of constituency and syntactic structure that it implicates, is in the end vindicated as a descriptively adequate theory of competence grammar, then it is likely that it will also be very simply and directly related to the parser as well. In other words, if it is descriptively adequate, then it is probably explanatorily adequate as well.

9.1 Anatomy of a Processor

All language-processors can be viewed as made up of three elements. The first is a grammar, which defines how constituents combine to yield other constituents. The second is an algorithm for applying the rules of the grammar to a string. The third is an oracle, or mechanism for resolving nondeterminism. The oracle decides which rule of grammar to apply at points in the analysis where the nondeterministic algorithm allows more than one rule to apply. The following sections briefly discuss these elements in turn.[1]

9.1.1 Grammar

The strong competence hypothesis as originally stated by Bresnan and Kaplan (1982) assumes that the grammar that is used by or implicit in the human sentence processor is the competence grammar itself. It is important to be clear that this is an assumption, not a logical requirement. The processors that we design ourselves (such as compilers for programming languages) quite often do not exhibit this property. There is no logical necessity for the structures involved in processing a programming language to have anything to do with the structures that are implicated by its competence grammar—that is, the syntactic rules in the reference manual that are associated with its semantics. The compiler or interpreter can parse according to a quite different grammar, provided that there exists a computable homomorphism mapping the structures of this "covering grammar" onto the structures of the competence grammar. If the homomorphism is simple, so that the computational costs of parsing according to the covering grammar plus the costs of computing the mapping are less than the costs of parsing according to the competence grammar, then there may be a significant practical advantage in this tactic. For this reason, it is quite common for compilers and interpreters to parse according to a weakly equivalent covering grammar, mapping to the "real" grammar via a homomorphism under concatenation on a string representing the derivation under the covering grammar. For example, programming language compilers sometimes work like this, when a parsing algorithm that is desirable for reasons of efficiency demands grammars in a normal form that is not adhered to by the grammar in the reference manual (see Gray and Harrison 1972; Nijholt 1980). Such a situation also arises in artificial parsers for natural languages, when it is desired to use top-down algorithms, which can be ill suited to the left-recursive rules that commonly occur in natural grammars (see Kuno 1966 for an early example). As Berwick and Weinberg (1984, esp. 78-82) note, there is therefore no logical necessity for the structures involved in human syntactic processing to have anything to do with the structures that are implicated by the competence grammar—that is, the structures that support the semantics.

Nevertheless, similar considerations of parsimony in the theory of language evolution and language development to those invoked earlier might also lead us to expect that, as a matter of fact, a close relation is likely to hold between the competence grammar and the structures dealt with by the psychological processor, and that it will in fact incorporate the competence grammar in a modular fashion. One reason that has been frequently invoked is that language development in children is extremely fast and gives the appearance of proceeding via the piecemeal addition, substitution, and modification of individual rules and categories of competence grammar. Any addition of, or change to, a rule of competence grammar will not in general correspond to a similarly modular change in a covering grammar. Instead, the entire ensemble of competence rules will typically have to be recompiled into a new covering grammar. Even if we assume that the transformation of one grammar into another is determined by a language-independent algorithm and can be computed each time at negligible cost, we have still sacrificed parsimony in the theory and increased the burden of explanation on the theory of evolution. In particular, it is quite unclear why the development of either of the principal components of the theory in isolation should confer any selective advantage. The competence grammar is by assumption unprocessable, and the covering grammar is by assumption uninterpretable. It looks as though they can only evolve as a unified system, together with the translation process. This is likely to be harder than to evolve a strictly competence-based system.[2]

Indeed, the first thing we would have to explain is why a covering grammar was necessary in the first place. The reference grammars of programming languages are constrained by human requirements rather than the requirements of the machines that process them. Such grammars can be ill suited to parsing with the particular algorithms that we happen to be clever enough to think of and to be able to implement on that kind of machine. It is we humans who find requirements like Greibach Normal Form tedious and who prefer grammars with left-recursive rules, forcing the use of covering grammars on some artificial processors. If convenience to the available computing machinery were the only factor determining the form of computer languages, then their grammars would take a form that would not require the use of a covering grammar at all. It is quite unclear what external force could have the effect of making natural grammars ill-matched to the *natural* sentence processor.[3]

It is important to note that the strong competence hypothesis as stated by Bresnan and Kaplan imposes no further constraint on the processor. In particular, it does not limit the structures built by the processor to fully instantiated constituents. However, the Strict Competence Hypothesis proposed in this book imposes this stronger condition.[4] The reasoning behind this strict version is again evolutionary. If in order to process sentences we need more than the grammar itself, even a perfectly general "compiler" that turns grammars into algorithms dealing in other structures, then the load on evolution is increased. Similar arguments for the need for the grammar and processor to evolve in lockstep mean that a theory that keeps such extras to the minimum is preferred.

This strict version of the strong competence hypothesis has the effect of generalizing the Constituent Condition on Rules to cover the processor. The claim is that the constituents that are recognized in the grammar (and their interpretations) will be the only structures the processor will give evidence of. Anything else we are forced to postulate is an extra assumption and will require an independent explanation if it is not to count against the theory. Of course, such an explanation may be readily forthcoming. But if it is not, then it will remain a challenge to explanatory adequacy.

9.1.2 The Algorithm

If we believe that the natural processor must incorporate the competence grammar directly, what more must it include? According to the assumptions with which this section began, it must include a nondeterministic algorithm that will apply the rules of the grammar to accept or reject the string, together with some extra apparatus for simultaneously building a structure representing its analysis. Provided that the competence grammar is monotonic, this structure can be the semantic translation itself, rather than a strictly syntactic structure. Under this view (which has been standard in computational linguistics at least since Woods's (1970) ATN), a syntactic derivation is simply a trace of the way in which this interpretable structure was built.

The processor must also include an oracle (dealt with in section 9.1.3) to resolve the nondeterminism that the grammar allows (or at least rank the alternatives) for the algorithm. A theory will be successful to the extent that both of these components can be kept as minimal and as language-independent as possible. For this reason, we should be very careful to exclude the possibility that either the algorithm or the oracle covertly embeds rules of a grammar other than the competence grammar.

There are of course a great many algorithms for any given theory of grammar, even when we confine ourselves to the simpler alternatives. They may work top-down and depth-first through the rules of the grammar, or work bottom up from the string via the rules, or employ some mixture of the two strategies. For obvious reasons, most parsers with any claim to psychological realism work from the earliest elements of the sentence to the last, or (for the present orthography) leftmost-first, but alternatives are possible here, too.

Such algorithms require an automaton, including a working memory such as the chart mechanism discussed below, in addition to the competence grammar and a mechanism for eliminating nondeterminism. For context-free grammars the automaton is a pushdown automaton. For the classes of grammars treated in this book, it is Vijay-Shanker and Weir's (1990; 1994) generalization of the same device, the extended pushdown automaton, also discussed below. The question we must ask under the strict competence hypothesis is, how little more can we get away with? In particular, can we get away with nothing more than the theoretical minimum—that is, an algorithm that does not need to know about anything except rules of grammar, the string, and the state of the stack and the working memory, subject to the adjudication of the oracle?

CCGs are very directly compatible with one of the simplest classes of algorithm, namely, the binary-branching bottom-up algorithms. They are most easily understood by considering in turn: (a) a nondeterministic shift-reduce parser, and (b) a chart-based deterministic left-right incremental version of the Cocke-Kasami-Younger (CKY) algorithm (Cocke and Schwartz 1970; Harrison 1978).[5]

The nondeterministic leftmost-first shift-reduce algorithm can be stated as follows:

(1) 1. Initialize the stack to the empty stack and make a pointer point to position 0 in the string, before the first word.
   2. As long as there are any words left in the string or a combinatory rule can apply to the topmost item(s) on the stack **either**:
      a. Put on the stack (shift) a category corresponding to the word that starts at the pointed-to position, **or**:
      b. Apply the combinatory rule to the topmost categories on the stack and replace them by its result (reduce).

For a simple sentence, *Thieves love watches*, this algorithm allows an analysis via the sequence shift, shift, reduce, shift, reduce (for simplicity, NPs are shown as lexically raised):

(2)

$$\begin{array}{|c|}\hline (S\backslash NP)/NP : love' \\ \hline S/(S\backslash NP) : \lambda p.p\ thieves' \\ \hline \end{array} \qquad \begin{array}{|c|}\hline S/NP : \lambda x.love'x\ thieves' \\ \hline \end{array}$$

a. Shift, Shift b. Reduce

$$\begin{array}{|c|}\hline S\backslash(S/NP) : \lambda p.p\ watches' \\ \hline S/NP : \lambda x.love'x\ thieves' \\ \hline \end{array} \qquad \begin{array}{|c|}\hline S : love'\ watches'\ thieves' \\ \hline \end{array}$$

c. Shift d. Reduce

One (very bad) way of making this algorithm deterministic is to choose a default strategy for resolving shift-reduce ambiguities—say, "reduce-first"—and to keep a trail of parse states including alternatives not taken, backtracking to earlier states and trying the alternatives when the analysis blocks. This will cope with the fact that all three words in the sentence are ambiguous between nouns and verbs. However, if we want to be sure that we have found not only *an* analysis, but in fact *all possible* analyses of the sentence, we must restart the process and backtrack again until all possible avenues have been examined and no choices are left on the trail. Because naive backtracking parsers examine all possible paths in the search space, they are time-exponential in the number of words in the sentence. The source of this exponential cost lies in the algorithm's tendency to repeat identical analyses of the same substring, as when, the algorithm having mistakenly chosen the auxiliary category for the first word of the following sentence and having failed to find an analysis at the word *take*, the entire analysis of the arbitrarily complex subject NP has to be unwound and then repeated once the alternative of analyzing the first word as a main verb is taken (the example is from Marcus (1980)):

(3) Have *the students who missed the exam* take the makeup.

Because of the extra nondeterminism induced by type-raising and the associative composition rules, there is even more nondeterminism in CCG than in other grammars, so even for quite small fragments, particularly those that involve coordination, naive backtracking parsers are in practice unusable. Unlike standard grammars and parsing algorithms, because of the associativity of functional composition and the semantics of combinatory rules, CCG derivations fall into equivalence classes, with several derivations yielding identical interpretations.[6] Of course, it does not matter which member of any equivalence class we find, so long as we find some member of each. However, the search space is unacceptably large, and to ensure that we have found at least one member of all possible equivalence classes of derivation for a string of words, we are still threatened by having to search the entire space.

Karttunen (1989), Pareschi and Steedman (1987), and Pareschi (1989) discuss the use of a "chart" (Kay 1980) to reduce the search space for combinatory parsers. Chart parsing is by origin a technique for parsing CFPSG using a data structure in which all constituents that have been found so far are kept, indexed by the position of their left and right edge or boundary in the string. Each chart entry identifies the type of the constituent in question. It is common to refer to chart entries as "arcs" or "edges" and to represent the chart as a graph or network. We will be interested in the possibility that the chart may also associate other information with an arc or entry, such as the predicate-argument structure of the constituent. (For a sentence of length n, this chart can be conveniently represented as an $n \times n$ half-matrix with sets of categories as entries.)

Using a chart overcomes the main source of exponential costs in naive backtracking, arising from repeated identical analyses of a given substring. In the case of mere recognition, it is enough to make one entry in the table for a constituent of a given type spanning a given substring. For the task of finding all distinct parses of a sentence, the chart must include an entry for each distinct analysis spanning that substring.

Since even context-free grammars can approach bracketing completeness, and do so in practice for constructions like multiple noun compounding, to similarly reduce the worst-case complexity of the parsing problem to n^3 requires the use of structure-sharing techniques to produce a "shared forest" of analyses using linked lists (see Cocke and Schwartz 1970; Earley 1970; Pratt 1975; Tomita 1987; Billot and Lang 1989; Dörre 1997).

As noted earlier, because of the associative nature of function composition, CCG parsers will potentially deliver structurally distinct derivations for a constituent of a given type and interpretation spanning a given substring—the property that is misleadingly referred to as "spurious" ambiguity.[7] If multiple equivalent analyses are entered into the chart, then they too will engender an explosion in computational costs. To the extent that CCGs approximate the bracketing completeness of the Lambek calculus version, the number of derivations will proliferate as the Catalan function of the length of the sentence—essentially exponentially.

Pareschi, following Karttunen, proposed to eliminate such redundancies via a check for constituents of the same type and interpretation, using unification. Any new constituent resulting from a reduction of existing constituents whose predicate-argument structure was identical to one already existing in the chart would not be added to it. This reduces the worst-case complexity of combinatory parsing/recognition for the context-free case to the same as that for standard context-free grammars without "spurious" ambiguity—that is, to n^3, with the same proviso that interpretation structures are shared, and with a constant overhead for the redundant reductions and for the unification-based matching entry check.[8]

Vijay-Shanker and Weir (1994) discuss the problem of generalizing the context-free algorithms to mildly context-sensitive formalisms including the present one. Because such grammars potentially introduce infinitely many nonterminal categories, generalizing the CKY algorithm discussed below to deal with them potentially makes it worst-case exponential, unless a technique of structure sharing of category entries that they describe is used. It should not be forgotten that this is strictly a worst-case complexity result. As always, caution is needed in drawing conclusions for practical average-case complexity. Komagata (1997a) suggests that average-case recognition complexity for significant practical grammar fragments for Japanese and English is roughly cubic, so that the overhead of Vijay-Shanker and Weir's technique may not be worthwhile in practical applications.

As a first step toward defining a psychologically reasonable parser, it is instructive to see a trace of an algorithm of this kind, the left-to-right breadth-first bottom-up context-free chart parser. This can be defined in terms of an algorithm that can be informally stated as follows.[9]

(4) 1. Initialize the chart to the empty chart, and make a pointer point to position 0 in the string, before the first word.
  2. Until the end of the sentence is reached:
    a. Add entries corresponding to all categories of the word that starts at the pointed-to position. Make the pointer point to the next position in the sentence.
    b. As long as there is a pair of entries in the chart that can reduce, do the reduction and add an entry representing the result to the chart, unless the matching-entry test reveals that an equivalent entry is already present.

Since by definition shifting a new lexical category for the jth word can only induce new reductions to give categories whose right boundary is at position j in the sentence, an efficient way of carrying out step 2 is to ask for all i where $0 \leq i \leq (j-2)$ whether there are any such reductions. This in turn means asking for all k where $i < k < j$ whether there are entries spanning (i,k) and (k,j) that reduce. Since adding a new entry (i,j) during this process may itself enable further reductions, it is necessary to compute the new entries (i,j) bottom up—that is, by starting with $i = j-2$ and stepping down to $i = 0$. Hence, we can state the algorithm more completely and formally as follows, where *present* is the test for a matching entry already in the table, and A, B, C are category-interpretation pairs of the form $\Sigma : \Lambda$ where Σ is a syntactic category such as *NP* or S/NP, and Λ is a predicate-argument structure:

```
(5) 1. for $j := 1$ to $n$ do
       begin
       $t(j-1, j) := \{A | A \text{ is a lexical category for } a_j\}$
    2. for $i := j-2$ down to 0 do
       begin
    3. $t(i,j) := \{A | \text{there exists } k,\ i < k < j, \text{ such that } B\ C \Rightarrow A \text{ for some}$
                  $B \in t(i,k),\ C \in t(k,j), \text{ and not } present(A,i,j)\}$
       end
       end
```

This algorithm is complete, in the sense that it finds all possible grammatical constituents and all complete analyses for the sentence.

. Dexter . must . know . Warren . well .

Figure 9.1
The start-state: string and empty chart

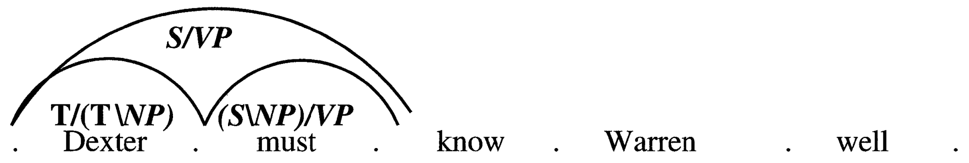

Figure 9.2
Shift *Dexter*, shift *must*, reduce

Imagine that such a parser is faced with the sentence *Dexter must know Warren well*, and suppose (simplifying) that all words have a unique category, including the adverb *well* which is a VP modifier with the single category $VP\backslash VP$. The reduce-first strategy goes through the following stages. Since the chart is initially empty, as in figure 9.1, nothing can happen until categories have been shifted to the chart for the first two words, *Dexter must*. At that point a single reduction is possible via the composition rule, leaving the chart in the state shown in figure 9.2. No further reductions are possible, so we shift a category for the next word, *know*. Two new reductions are now permitted, again via the composition rule. One of these is with the previously shifted modal, $(S\backslash NP)/VP$, and one with the result of the previous reduction, S/VP. The first induces a result that can further reduce with the subject, but this yields a result equivalent to the second in predicate-argument structural terms, so one or the other is detected to be redundant. No further reductions are possible, so the state is as in figure 9.3. We must shift a category for the word *Warren*, a shift that precipitates reductions and new entries as shown in figure 9.4. These include a number of redundant constituents that will take part in no grammatical analysis, including the *S, Dexter must know Warren*. Many of them have multiple analyses that must be detected by the matching check (or preempted by some other mechanism such as the normal form parsers that Hepple and Morrill (1989), Hendriks (1993), König (1994), and Eisner (1996) have proposed). Finally we shift a category for the adverb and halt in the state shown with a single S spanning positions 0 through 5, as in figure 9.5.

This derivation reveals the tendency for bottom-up parsers to build unnecessary constituents, here typified by the spurious *S, Dexter must know Warren*. Even the comparative simplicity of the derivation described above is mislead-

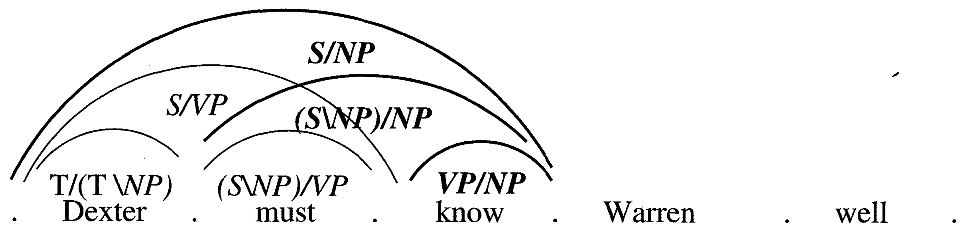

Figure 9.3
Shift *know*, reduce, reduce

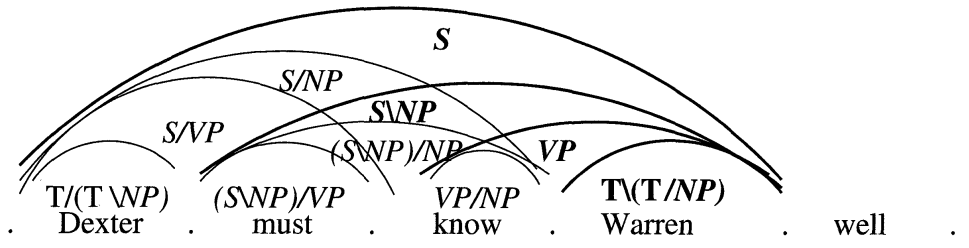

Figure 9.4
Shift *Warren*, reduce, reduce, reduce

ing in this respect. There is at least one other category for *Warren* (namely, the *subject* type-raised category), and it can combine with another category for *know* (namely, VP/S). In more complex sentences such fruitless analyses will proliferate.

However, this example serves to reify some of the main components of a practical parser, in preparation for the discussion of how this sort of device could be made more like a human processor. Human beings are rarely aware of lexical or global syntactic ambiguity in the sentences that they process, and they rarely encounter difficulty arising from nondeterminism in the grammar. How can this be? There was a broad hint in chapter 5, where we saw that intonational boundaries can on occasion reduce the ambiguity of CCG derivation. Although such indicators are frequently missing, the occasions on which they *are* missing can plausibly be argued to be exactly the occasions on which the Information Structure, and therefore some important aspects of Surface Structure, can be assumed to be known to the hearer. Perhaps there are other sources of information that mean that redundant structure building and proliferation of categories exemplified above can be eliminated for the benefit of the parser.

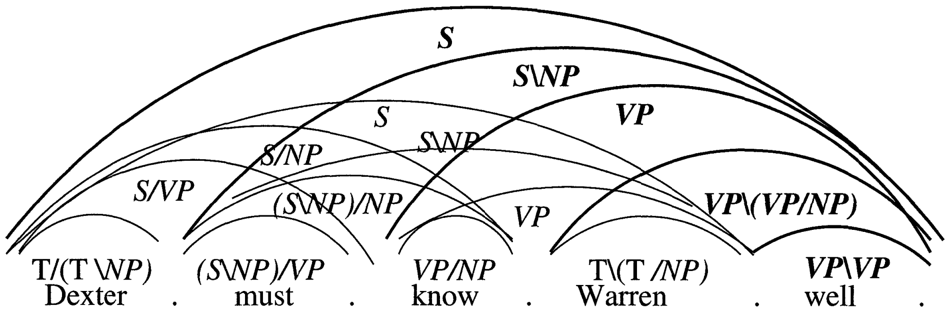

Figure 9.5
Shift *well*, reduce, reduce, halt

9.1.3 The Oracle

Nothing about the expected close relation between syntax and semantics entails that the mapping should be unambiguous, even though the grammars for artificial languages we design ourselves typically permit only the most local of nondeterminism, because they are designed for use as formal calculi. Expressions in natural languages seem to be remarkably free with ambiguity, both global and local, as in the following famous examples;

(6) Flying planes can be dangerous.

(7) a. Have the students taken the exam?
 b. Have the students take the exam!

In the latter example, from Marcus 1980, the substring *have the students* is (syntactically and semantically) locally ambiguous, in the sense that a processor cannot immediately know which rule of grammar to apply after encountering it. Human beings seem to be remarkably adept at resolving such ambiguities, which are astonishingly profuse in natural language.[10]

Probably for the same reason, we do as a matter of fact tend to design our artificial languages in ways that make their symbols "locally" ambiguous, either in terms of which rule of syntax should apply, or in terms of which rule of semantics should apply. (An example of the latter is the "overloading" of an operator like + to denote distinct operations applying to integers, real numbers, and complex numbers.) The one lesson that we can derive from our experience with artificial languages and processors like compilers is that such ambiguities must be resolvable quickly and locally if the computational complexity of processing is to be contained.

In order to facilitate this requirement, programming languages are invariably carefully designed so that local ambiguity can be resolved immediately, either syntactically by examining the next symbol in the string, or semantically by examining the types of functions and arguments (as in the case of overloading above). However, natural language shows no sign of any such constraint from within grammar. For example, although the locally ambiguous substring *Have the students* ... in (7) is disambiguated by the phrase *take/taken the exam*, an indefinite amount of further linguistic material may intervene between the ambiguous substring and the disambiguating information, as when the sentences begin *Have the students who were late with the homework* ..., *Have the students who were late with the homework that I assigned last Monday* ..., and so on. This apparent nondeterminism in the grammar is an anomaly that requires further explanation, for if we allow the ambiguities to proliferate, then the costs of maintaining the alternatives will explode. Indeed, as Marcus points out, we must be able to eliminate all but some bounded number of alternative paths on the basis of purely local evidence, since there is no evidence that processing load increases as a worse-than-linear function of sentence length. I will call the device that eliminates nondeterminism, and decrees which rule of the grammar should be invoked at any point in the derivation, an "oracle." However this device works, it is clear that it must be very effective in order to deal with the degree of nondeterminism that natural grammars exhibit. Moreover, as noted earlier, it must also be entirely language-independent, if it is not to compromise the parsimony and modularity of the theory of the processor.

Most accounts of the human sentence-processing mechanism have assumed that local attachment ambiguity resolution is based on structural criteria, such as parsing "strategies" (Fodor, Bever and Garrett 1974; Kimball 1973), structural preferences (Frazier 1978), rule orderings (Wanner 1980), lexical preferences (Ford, Bresnan and Kaplan 1982), or lookahead (Marcus 1980). Such accounts have been claimed to explain a wide range of sentence-processing phenomena, the most spectacular of which is undoubtedly the identification by Bever (1970) of the "garden path phenomenon"—that is, the existence of sentences like the following, for which a local ambiguity is *mis*resolved in a way that makes a perfectly grammatical sentence unanalyzable:

(8) The horse raced past the barn fell.

However, such accounts have generally been characterized either by empirical shortcomings or by proliferation of devices and degrees of freedom in the theory (see e.g. the exchange between Frazier and Fodor (1978), and Wanner

(1980)). In particular, since the earliest stages of the inquiry, it has been clear that all human parsing phenomena are extremely sensitive to the influence of semantics and especially referential context. Bever (1970) notes a difference in the strength of the garden path effect in minimal pairs of sentences analogous to the following, raising the possibility of an influence either from different word transition probabilities or from the related differing pragmatic plausibility of analyzing the initial NP as a subject of the ambiguous verb/participle *sent*:

(9) a. The doctor sent for the patient arrived.
b. The flowers sent for the patient arrived.

Various computational proposals have been made for how pragmatic plausibility might have this effect via a "weak" interaction between syntax and semantics, using a filtering process of comparing rival partial analyses on the basis of their success or failure in referring to entities in the model or discourse context (see Winograd 1972 and Hirst 1987). In particular, Crain and Steedman (1985) and Altmann and Steedman (1988) proposed a criterion for selecting among analyses called the "Principle of Parsimony," which can be stated as follows:

(10) *The Principle of Parsimony*
The analysis whose interpretation carries fewest unsatisfied but accommodatable presuppositions or consistent entailments will be preferred.

These authors use the term "presupposition" in the "pragmatic" sense of Stalnaker (1974) and Lewis (1979), and explain this principle in terms of the associated notion of accommodation of unsatisfied presuppositions. They point out that the two analyses of sentence (8), which differ according to whether it begins with a simple NP *the horse* or a complex NP *the horse raced past the barn*, also differ in the number of horses whose existence in the model they presuppose (one or more than one) and in the number of properties that they assume to distinguish them—none in the case of the singleton horse, and being caused to race along a given path in contrast to some other property in the case of multiple horses. They argue that contexts which already support one or the other set of presuppositions—say, because a single horse has previously been mentioned, or several horses and some racing—will favor the related analysis at the point of ambiguity and thereby either induce or eliminate the garden path for this sentence under the Principle of Parsimony. Crucially, they also argue that the empty context, in which *no* horses and no racing have been mentioned, will favor the simplex NP analysis, because its interpretation carries fewer unsatisfied but consistent presuppositions and is therefore easiest to accommodate. The principle accordingly predicts a garden path in the empty context.

In support of this view, Crain and Steedman (1985) offer experimental evidence that attachment preferences are under the control of referential context. Subjects were presented with minimal pairs of target sentences displaying local attachment ambiguities, preceded by contexts that established either two referents, respectively with and without a distinguishing property, or one referent with that property. Examples (modified from the original) are as follows:

(11) a. *Contexts:*
- i. A psychologist was counseling two women. He was worried about one of them, but not about the other.
- ii. A psychologist was counseling a man and a woman. He was worried about one of them, but not about the other.

b. *Targets:*
- i. The psychologist told the woman that he was having trouble with *her husband.*
- ii. The psychologist told the woman that he was having trouble with *to visit him again.*

Both target sentences have a local ambiguity at the word *that*, which is resolved only when the italicized words are encountered. Frazier's (1978) Minimal Attachment Principle would predict that the second target would always cause a garden path. In fact, however, this garden path effect is eliminated when the sentence is preceded by the first context, which satisfies the presupposition of the relative-clause analysis. Moreover, a garden path effect is induced in the first target when it is preceded by the same context, because by the same token it fails to support the presupposition that there is a unique woman. Crain and Steedman (1985) also confirmed certain related predictions concerning the effect of definiteness on garden paths in the null context. The experiments were repeated and extended with improved materials by Altmann (1988) and Altmann and Steedman (1988), and the effect has been show to be robust across a number of experimental measures of processing load including brain-imaging and Event-Related Potential (ERP) measures (van Berkum, Brown and Hagoort 1999).

Whereas examples like (9) are compatible with an alternative explanation based on word transition probabilities and higher-order statistics of the language, these experiments showing effects of referential context with minimal pairs of targets are much harder to plausibly account for in this way.

The majority of early psycholinguistic experiments on processing loads used only empty contexts, and therefore failed to control for this sort of effect. However, more recent experiments (see e.g. Carroll, Tanenhaus and Bever 1978; Tanenhaus 1978; Marslen-Wilson, Tyler and Seidenberg 1978; Swinney 1979; Tanenhaus, Leiman and Seidenberg 1979; Crain 1980; Altmann 1985; Trueswell, Tanenhaus and Kello 1993; Trueswell, Tanenhaus and Garnsey 1994; Spivey-Knowlton, Trueswell and Tanenhaus 1993; Sedivy and Spivey-Knowlton 1993; and van Berkum, Brown and Hagoort 1999) have now built up a considerable body of evidence that effects of semantics, knowledge-based plausibility, and referential context are extremely strong. Indeed, almost all theories of performance nowadays admit that some such component, in the form of a "thematic processor" (Frazier 1989), "construal" (Frazier and Clifton 1996), or the equivalent, can intervene at an early stage of processing. The only remaining area of disagreement is whether anything *else* besides this potentially very powerful source of ambiguity resolution is actually required. (See the exchange between Clifton and Ferreira (1989) and Steedman and Altmann (1989)). For, if interpretations are available at every turn in sentence processing, then there is every reason to suppose that the local syntactic ambiguities that abound in natural language sentences may be resolved by taking into account the appropriateness of those interpretations to the context of utterance, even when the rival analyses are in traditional terms incomplete. Indeed, the possibility that human language-processors are able to draw on the information implicit in the context or discourse model seems to offer the only mechanism powerful enough to handle the astonishing profusion of local and global ambiguities that human languages allow and to explain the fact that human language users are so rarely aware of them. Such a selective or "weak" interaction between syntactic processing and semantic interpretation is entirely modular, as J.A. Fodor (1983, 78 and 135) points out.

If interpretation in context is the basis of local ambiguity resolution, then a number of further properties of the parser follow. The felicity of an interpretation with respect to a context is not an all-or-none property, comparable to syntactic well-formedness. Utterances are often surprising—indeed, they are infelicitous if they are *not* at least somewhat novel in content. It follows that evaluation in context can only yield information about the *relative* good fit of various alternatives. We might therefore expect the parser to use a tactic known as "beam-search," whereby at a point of local ambiguity, all alternative analyses permitted by the grammar are proposed in parallel, and their interpretations are then evaluated in parallel. Readings that fail to refer or are otherwise implausible are discarded or ranked lower than ones that are consistent with what is known, along the lines suggested earlier (see Gibson 1996; Collins 1997, 1998; Charniak, Goldwater and Johnson 1998.) The parsing process then proceeds with the best candidate(s), all others being discarded or interrupted. (A similar tactic is widely used in automatic speech processing to eliminate the large numbers of spurious candidates that are thrown up in word recognition; see Lee 1989).

On the assumption that the number of alternative analyses that can be maintained at any one time is strictly limited, we can also assume that the process of semantic filtering occurs very soon after the alternatives are proposed. It should at least be completed before the next point of local ambiguity, for otherwise we incur the penalties of exponential growth in the number of analyses. Given the degree of nondeterminism characteristic of natural grammars, this means that the interplay of syntactic analysis and semantic adjudication must be extremely intimate and fine-grained. Since most words are ambiguous, semantic adjudication will probably be needed almost word by word.

For example, consider (9), repeated here:

(12) a. The doctor sent for the patient arrived.
     b. The flowers sent for the patient arrived.

The garden path effect in (12a) is reduced in (12b), because flowers, unlike doctors, cannot send for things. The very existence of a garden path effect in (12a) suggests that this knowledge must be available early. If the processor were able to delay commitment until the end of the putative clause *the flowers sent for the patient*, then it would have got to the point of syntactic disambiguation by the main verb, and there would be no reason not to expect it to be able to recover from the garden path in (12a). It follows that to explain the lack of such an effect in (12b), we must suppose that the interpretation of an *incomplete* proposition *the flowers sent for* ... is available in advance of processing the rest of the PP, so that its lack of an extension can cause the garden path analysis to be aborted.[11]

However, the proposal to resolve nondeterminism by appeal to such interpretations immediately leads to an apparent paradox. If the processor resolves nondeterminism in midsentence, more or less word by word, on the basis of contextual appropriateness of interpretations, then those interpretations must be available in mid-sentence, also more or less word by word. However, under the rule-to-rule hypothesis and the strict competence hypothesis, only *constituents* have interpretations, and only constituents are available to the processor. Now, there is no particular problem about constructing a grammar according to which every leftmost string is a constituent, so that processing can proceed in this incremental fashion. Any left-branching grammar provides an example. For such grammars, the assumption of a rule-to-rule compositional semantics means that, for each terminal in a left-to-right pass through the string, as soon as it is syntactically incorporated into a phrase, the interpretation of that phrase can be provided. And since the interpretation is complete, it may also be evaluated; for example, if the constituent is a noun or an NP, then its extension or referent may be found.

A right-branching context-free grammar, on the other hand, does not have this property for left-to-right processors. In the absence of some further apparatus going beyond rule-to-rule processing and rule-to-rule semantics, all comprehension must wait until the end of the string, when the first complete constituent is built and can be interpreted. Until that point any processor that adheres to the strict competence hypothesis must simply pile up constituents on the stack. It therefore seems that we should, under the strict competence hypothesis, expect the languages of the world to favor left- branching constructions, at least wherever incremental interpretation is important for purposes of resolving nondeterminism. However, the languages of the world make extravagant use of *right*-branching constructions—the crucial clause in (12), *The flowers sent for the patient*, being a case in point. The availability of an interpretation for what are in traditional terms nonconstituents (e.g. *the flowers sent* ... and/or *the flowers sent for* ...) therefore contradicts the strict competence hypothesis, if we assume the orthodox grammar.

It is therefore interesting that CCG makes such fragments as *the doctor/flowers sent for* ... available in the competence grammar, complete with an interpretation, and comparable in every way to more traditional constituents like the clause and the predicate. To the extent that the empirical evidence—for example, from comparison of the garden path effect in similar minimal pairs of sentences by Trueswell and Tanenhaus and colleagues—suggests that interpretations are available to the processor for such fragments, it follows that the present theory of grammar delivers a simpler account of the processor, without compromising the strict competence hypothesis. Derivations like (56) in chapter 6 show that this claim extends to the SOV case.[12]

It is important to be clear that this problem for traditional right-branching grammars is independent of the particular algorithms discussed in section 9.1.2. It applies to bottom-up and top-down algorithms alike, so long as they adhere to the strict competence hypothesis. Top-down algorithms have the apparent advantage of being syntactically predictive, a fact whose psychological relevance is noted by Kimball (1973) and Frazier and Fodor (1978). However, neither algorithm of itself will allow an interpretation to be accessed for the leftmost substring, in advance of their being combined into a constituent. Therefore, neither algorithm unaided will allow word-by-word incremental semantic filtering as a basis for the oracle within right-branching constructions.

To say this is not of course to deny that incremental interpretation is possible for right-branching grammars if they do *not* adhere to the strict competence hypothesis in this extreme form. In fact, the requisite information is quite easy to compute from the rules of the grammar. It would not be unreasonable to postulate a language-independent mechanism using functional composition to map traditional grammar rules onto new rules defining parser-specific entities like S/NP.

For example, Pulman (1986, 212–213) proposes a bottom-up shift-reduce processor that includes a rule "Clear" that combines subjects like *the flowers* and a transitive verb *sent* (with the "summon" reading) on a stack, thus:[13]

(13)
$$\begin{array}{|c|}\hline VP/PP : \lambda x.summon'x \\ \hline S/VP : \lambda p.p\,flowers' \\ \hline\end{array} \quad\Longrightarrow\quad \boxed{S/PP : \lambda x.summon'x\,flowers'}$$

The rule Clear corresponds to an operation of semantic composition on categories on the parser's stack, as distinct from a grammatical rule. It therefore violates the strict competence hypothesis. If such violations are permitted, then it is clearly easy for a processor to gain access to interpretations more incrementally than the grammar would otherwise allow.

However, this argument cuts two ways. If such entities can be associated with semantic interpretations, why are they *not* grammaticalized? Under the assumption that grammar is just the reification of conceptual structure, why are these apparently useful concepts getting left out?

Of course, there is a lot that we don't know about the conceptual infrastructure of grammar. We are not yet in a position to say whether or not there is anything odd about those concepts that causes them to be left out. However, given that such conceptual objects seem to be accessible to the parser for resolving nondeterminism, it is interesting to remember at this point that categorial grammars of the kind discussed here already *do* grammaticalize the fragments in question. Since they do so by including composition as a component of competence grammar, they predict that the same operation should be available to the processor, under the strict competence hypothesis, rather than requiring it as an extra stipulation and thereby violating that principle.

There is in fact no sense in which a parser using a right-branching grammar under strict competence can access interpretations for substrings that are not constituents. In the case of (12b), this means that the anomaly of the clausal reading of the substring *the flowers sent for the patient* cannot be detected until after the word *patient*. However, this is rather late in the day. The very next word in the sentence is the disambiguating main verb *arrived*. Since we know that there is a garden path effect in (12a), and we know that in context the grammatical reading can be comprehended, we have reason to believe that the human disambiguation point must be earlier, around the verb or the preposition. If so, then the degree of incremental interpretation permitted under the strict competence hypothesis for standard right-branching grammars for this construction is of insufficiently fine grain.

Stabler (1991), criticizing an earlier version of this proposal (Steedman 1989), has argued that the present claims are in error, and that incremental interpretation of right-branching constituents is in fact possible without violating the strong competence hypothesis in the strict sense used here.

Stabler does not in fact adhere to the strict form of the strong competence hypothesis. It is clear that he is assuming a weaker form of the competence hypothesis, although he gives no explicit definition (see Stabler 1991, 233, n.1). In particular, in his first worked example, (p. 226) he binds a variable *Subj* in the interpretation of the sentence to the interpretation of the actual subject, via Prolog-style partial execution in the rule *I1* (p. 208). This is possible only because he is using the grammar as a predictive parser. He uses this information to identify the fact that since the context includes only one predication over this subject, that must be the one that is to come, under a caricature of incremental evaluation similar to that used above.[14]

However, we have seen that this much is merely information that could legitimately have been built into the grammar itself, via type-raising. (In fact, this analogy seems to be effectively embodied in Stabler's second example using an LR parser (Aho and Johnson 1974), although the details here are less clear.) As in the example offered above, Stabler's processor has not actually handled any interpretations that correspond to nonconstituents. It is therefore more important to ask whether it adheres to the strict competence hypothesis in all other respects by entirely avoiding interpretation of nonconstituents of the *the flowers sent* variety, or whether it violates the hypothesis by covertly constructing interpreted objects that are not merely constituents according to the competence grammar, either in the form of dashed categories or in the form of partially instantiated semantic interpretations.

Curiously, since his paper is addressed to a predecessor of the present proposal, and even though he technically allows strict competence to be compromised, all of Stabler's examples take the first tactic. Thus, in his exegesis of the (right-branching) sentence *The joke is funny*, there is no sense in which there ever exists an interpretation of the nonconstituent *the joke is*, or indeed anything comparable to *The flowers sent* (see Stabler 1991, 215, and 232).[15] His parser offers no help with the question raised by (12). It is neither consistent with the strict competence hypothesis nor incrementally interpretative in the sense argued for here.

Shieber and Johnson (1993) have also argued against the same earlier version of the present proposal on a rather different ground. They freely admit (p. 29) that their proposal for incorporating a version of incremental interpretation in a more or less traditional grammar violates of the strict competence hypothesis. (The violation arises when they exploit the fact that the state of their LR parser encodes a shared forest of possible interpretable partial analyses in much the same way as Pulman's (1986) parser discussed above; see pp. 18–22.) However, they claim that within such a not strictly competence-based parser, incremental interpretation is actually simpler than strictly constituent-based interpretation. The reasoning behind this interesting claim is that, once interpretation of nonconstituents is allowed by the addition of extragrammatical apparatus, imposing strict competence on the system requires the reimposition of synchrony, via further additional mechanisms such as a clock or switch.

As in the simpler examples discussed earlier of incrementally interpreting parsers using categories and the stack as interpretable objects, the real force of this argument depends upon the extent to which the apparatus for interpreting LR states as encoding partial analyses can be given some independent motivation. All the earlier questions about why these interpretations are *not* grammaticalized also remain to be answered, especially when it is recalled that other competence phenomena (e.g. coordination, considered in part II) suggest that similar interpretations indeed behave like grammatical entities. Such questions are simply open, and they will remain so until the rival competence theories attain something closer to descriptive adequacy than any of them do today.

The resolution of the apparent paradox of incremental interpretation does not lie in Stabler's or Shieber and Johnson's parsers, but in the observation that strings like *Anna married* and *the flowers sent* are in the fullest sense constituents of competence grammar. We can retain the strict version of the strong competence hypothesis and continue to require the grammar to support incremental interpretation, if we also take on board the combinatory theory of grammar. This theory offers a broader definition of constituency, under which more substrings, and in particular more left prefixes, are associated with interpretations as a matter of grammar. The interpretations of such nonstandard constituents can therefore be used to compare rival analyses arising from nondeterminism on the basis of their fit to the context, without violating the strict competence hypothesis.

9.2 Toward Psychologically Realistic Parsers

How could a reasonably efficient parser of this kind be built? One possibility is a very simple modification of the breadth-first incremental CKY parser sketched in section 9.1.2. The modification is that when a new constituent (i, j) is found, we not only check that it is not already present in the chart before adding it. We also check that it makes sense by evaluating it either with respect to a priori likelihood with respect to the knowledge base, as Winograd (1972) suggests for disambiguating compound NPs like *water meter cover adjustment screw* or (if it is a main-clause prefix) with respect to referential context, as in the flowers/doctor example (12).

We noted in the earlier discussion that we need only consider new arcs ending at j when the categories of the jth word a_j are shifted, and that it is necessary to compute the new entries (i, j) bottom up—that is, by starting with $i = j - 2$ and stepping down to $i = 0$. For each i we ask for all k where $i < k < j$ whether there are entries spanning (i,k) and (k, j) that reduce. If so, then the result is added to the table t as $t(i, j)$ if it survives the matching check. The algorithm can be defined as follows. (Compare Harrison 1978, 433, and example (5) above—the present version differs only in assuming that categories are accompanied by interpretations and that all reductions are considered in the innermost loop.) A, B, C are category-interpretation pairs $\Sigma : \Lambda$ as before:

```
(14) 1. for $j := 1$ to $n$ do
        begin
        $t(j-1, j) := \{A | A \text{ is a lexical category for } a_j\}$
     2. for $i := j-2$ down to 0 do
        begin
     3. a. $t(i, j) := \{A | \text{ there exists } k,\ i < k < j, \text{ such that } B\ C \Rightarrow A \text{ for some}$
                 $B \in t(i,k), C \in t(k, j), \text{ and not } present(A,i,j)\}$
        b. $t(i, j) := rank(t(i,j))$
        end
        end
```

The function *rank* is assumed to order the constituents in $t(i, j)$ according to plausibility, either intrinsically or in terms of the state of the context or database and the Principle of Parsimony (10). For the sake of simplicity I will assume in what follows that the highest-ranked element is assigned a plausibility value of 1 and the rest are assigned a plausibility value of 0, although more realistically a range of values summing to 1 and/or a threshold for entry to the chart could be used.

To make the proposal more concrete, I will again assume a very simplified account of the discourse model related to an extensional version of the Alternative Semantics of Rooth (1985, 1992) used in chapter 5.[16] In particular, I will assume that a context is a database containing modal propositions as individuals, corresponding to the fact that it is possible for a person to send anything for a person, that it is possible for a person to summon a person, that it is possible for anything to arrive, that doctors and patients are persons, and that flowers are not. To further simplify, I will ignore the "send into raptures" sense of *send*. In the null context the discourse model might look something like the following, where the $\Diamond$ prefix means that the event to its right is possible and can be accommodated in the sense defined earlier, and where I use the logic-programming convention that variables are implicitly universally quantified:

(15) $person'x \wedge person'z \rightarrow \Diamond send'xyz$
$person'x \wedge person'y \rightarrow \Diamond summon'xy$
$\Diamond arrive'x$
$doctor'x \rightarrow person'x$
$patient'x \rightarrow person'x$
$flowers'x \rightarrow \neg person'x$

This database rather crudely represents the fact that propositions about people sending and summoning can be readily accommodated or added to the hearer's representation of the situation, but that in our simplified example no propositions with flowers as the subject of sending or summoning can be accommodated.

As far as the grammar goes, we have the usual problem of deciding whether nouns like *flowers* optionally subcategorize for modifiers like relatives and past participials or not. On the argument given in section 4.3.2 to the effect that anything out of which something can be right node raised must be an argument, examples like the following suggest that such modifiers are arguments:

(16) a. a few *men that I gave and women that I sold* flowers
b. the *flowers sent for and chocolates given to* the patient

Continuing to simplify, I will represent this by simple lexical ambiguity on the noun *flowers*. For the same reason I will also continue to assume that type-raising applies lexically.

Consider what happens when the sentence *The flowers sent for the patient arrived* is processed in this context. We shift the definite article *the* and the two categories for the noun *flowers*, which can then reduce to yield a subject (among other irrelevant raised categories) meaning something like the following, in which ι is Russell's definite existential quantifier (Russell 1905—see van der Sandt 1988; Beaver 1997):

(17) a. $S/(S\backslash NP) : \lambda p.\iota x.flowers'x \wedge px$
  b. $(S/(S\backslash NP))/(N\backslash N) : \lambda q.\lambda p.\iota x.(flowers'x \wedge qx) \wedge px$

The ι operator in the first category requires the existence of exactly one entity of type *flowers′* The ι operator in the interpretation of the second category requires the existence of exactly one entity of type *flowers′* having one other property *q*

There are no such entities in the database, but they can be consistently accommodated. Since the first category requires accommodating one proposition and the second requires accommodating two, the first is more plausible. It is therefore ranked 1, and the accommodation is carried out. The second is ranked 0 and not accommodated. Importantly, both categories remain in the table.

Let us represent the accommodation by existentially instantiating the flowers with an arbitrary constant—say, $gensym'_1$—and adding the following fact to the database:

(18) $flowers'gensym'_1$

We next encounter the word *sent*, which has three categories:

(19) a. $((S\backslash NP)/PP)/NP : \lambda x.\lambda y.\lambda z.send'yxz$
  b. $(S\backslash NP)/PP : \lambda x.\lambda y.summon'xy$
  c. $(N\backslash N)/PP : \lambda x.\lambda p.\lambda y.p\ y \wedge send'yxsomeone'$

The raised subject (17a) can compose with the categories (19a,b) to yield the following categories:

(20) a. $S/PP : \lambda y.\iota x.flowers'x \wedge summon'yx$
  b. $(S/PP)/NP : \lambda y.\lambda z.\iota x.flowers'x \wedge send'zyx$

The other subject category, (17b), can compose with the last verbal category, (19c), to yield the following category:

(21) $(S/(S\backslash NP))/PP : \lambda y.\lambda p.\iota x.(\mathit{flowers}'x \wedge \mathit{send}'yx\mathit{someone}') \wedge px$

To assess the plausibility of (20a,b), the processor must ask if it is possible for flowers to send things for people or send for people:

(22) a. $\Diamond \mathit{flowers}'x \wedge \mathit{summon}'yx$
b. $\Diamond \mathit{flowers}'x \wedge \mathit{send}'zyx$

Neither possibility is supported by the database, so both of these categories are associated with a low probability by the ranking function *rank* when entered in the chart.

The plausibility of category (21) depends on there being just one thing around of type *flowers'* with the property that they were sent. Although there is no corresponding proposition in the database, the knowledge base does at least support the possibility of sending flowers:

(23) $\Diamond \mathit{flowers}'y \wedge \mathit{send}'zyx$

Category (21) therefore ends up as the highest-ranked category for *The flowers sent* . . . , ranked 1 on entry to the chart. Its presuppositions are accommodated by adding the following facts to the database about the already present arbitrary flowers gensym_1:

(24) $\mathit{send}'z\ \mathit{gensym}'_1\mathit{someone}'$

The next word to shift is the preposition *for*, which first reduces with (20b) and (21) by composition. Since flowers cannot send for anything, and can be sent for people, the first is ranked 0 and the second 1 on entry to the chart:

(25) a. $S/NP : \lambda y.\iota x.\mathit{flowers}'x \wedge \mathit{summon}'yx$
b. $(S/(S\backslash NP))/NP : \lambda y.\lambda p.\iota x.(\mathit{flowers}'x \wedge \mathit{send}'yx\mathit{someone}') \wedge px$

Note that this preference for the modified subject reverses that on the subject alone. (Other reductions, which we will come back to later, are possible at this stage.)

Next we shift *the*, shift *patient*, and reduce to yield a number of categories as in the case of the subject, of which the following (where $NP^{\uparrow}$ schematizes as usual over various raised categories) carries fewest presuppositions/entailments and is highest ranked:

(26) $NP^{\uparrow} : \lambda p.\iota x.\mathit{patient}'x \wedge px$

A unique patient must therefore be accommodated using another arbitrary constant:

(27) $patient' gensym'_3$

The raised NP can combine with both categories in (25) for *The flowers sent for* .... Since patients are people and can be summoned and sent things, two categories go in the chart for *The flowers sent for the patient* ... :

(28) a. $S : \iota y.patient' y \wedge \iota x.flowers' x \wedge summon' yx$
 b. $S/(S\backslash NP) : \lambda p.\iota y.patient' y \wedge \iota x.(flowers' x \wedge send' xysomeone') \wedge px$

The first of these is again implausible. The second is plausible to the extent that it is possible to send flowers for a patient, and that there is exactly one thing with the property *flowers′* and one with the property *patient′*, and that a proposition subsuming the following one—namely, (24)—is already accommodated:

(29) $send' gensym'_2 gensym'_1 someone'$

The subject in turn can combine with the main verb *arrived* and complete the analysis, since anything can arrive.

This version of the CKY parser is complete, in the sense that it builds all legal constituents, even when they are zero-ranked for likelihood. Other constituents will be built, some of which will be rejected under the matching-entry test as being redundant without any further evaluation and without affecting the rankings already assigned to the equivalent constituents already in the chart. The important thing to note is that the anomaly of the tensed-verb reading is apparent as soon as the ambiguous word *sent* is encountered.

The analysis of the sentence *The doctor sent for the patient arrived* is identical, except that because of the plausibility of doctors sending for people, and the lesser presuppositional demand of the simple NP, the tensed-verb analysis is favored over the modifier analysis until the disambiguation point:

(30) $S/(S\backslash NP) : \lambda p.\iota y.doctor' y \wedge \iota x.(patient' x \wedge send' xysomeone') \wedge px$
 $S : \iota y.doctor' y \wedge \iota x.patient' x \wedge \wedge summon' xy$

Now suppose that a single doctor is already identified in the context.

(31) $doctor' dexter'$

Here the analysis of *The doctor sent for the patient arrived* will proceed exactly as in the null context, except that the unique doctor will no longer need to be accommodated. The analysis will receive a low rank for the same reason.

However, consider the case where there are two known doctors in the context:

(32) *doctor′dexter′*
doctor′warren′

Even if the entailment of the restrictor (that someone sent one of them for the patient) is not known and must be accommodated, the simple NP will fail to refer from the start and the complex NP will be highly ranked, as in the case of *The flowers sent for the patient arrived.*

We have already noted that the CKY algorithm as described here is complete. This means that it does not of itself predict exactly which sentence-context pairs will lead to unrecoverable garden paths. Moreover, we have unrealistically assumed that the ranking function does not need to take into account the preference values of the inputs to combinations. However, the algorithm shows that alternatives can be ranked consistently with the observed effects up to the point of disambiguation. This means that less conservative algorithms such as the beam-searching CCG parser of Niv (1993, 1994), the best-first chart-parser of Thompson (1990), or a version of CKY in which continuous probabilities are used to calculate exact likelihood values—say, using methods discussed by Collins (1996, 1997, 1998)—and subjected to a threshold, can be used to make more precise predictions.

9.3 CCG Parsing for Practical Applications

As noted earlier, the CKY algorithm has worst-case time complexity n^3 for recognition in the context-free case (because it involves three nested loops of complexity order n), and the n^6 worst-case complexity of recognition for Vijay-Shanker and Weir's (1990; 1993; 1994) generalization to CCG depends upon a complex structure-sharing technique for categories. Moreover, polynomial worst-case complexity for the corresponding parsers depends in both cases upon similarly subtle techniques for structure sharing among parse trees or interpretable structures using devices like "shared forests" (Billot and Lang 1989).

However, experiments by Komagata (1997a, 1999) with a CKY parser for hand-built CCG grammar fragments of English and Japanese for real texts from a constrained medical domain suggests that average-case parsing complexity for practical CCG-based grammars can in practice be quite reasonable even in the absence of semantic disambiguation or statistically-based optimization, despite its worst-case exponentiality.[17] To the extent that the psychological processor limits attachment ambiguities by the kind of semantic strategy outlined here, or by the related probabilistic techniques discussed in section 9.2, psychologically implausible mechanisms to manage shared forests as a representation of the alternative parsers may also be eliminated.

The modified CKY algorithm is only one among a number of possibilities, including parsers based on more predictive algorithms such top-down and mixed top-down and bottom-up algorithms, such as that of Earley (1970). The important point in terms of psychological realism is that by using CCG as the grammatical module, all such algorithms can be semantically incremental, while remaining entirely neutral with respect to the particular theory of grammar involved and exactly as incremental as the grammar itself allows, in keeping with the strict competence hypothesis.

Nevertheless, for many applications this kind of algorithm will always be vulnerable to well-known limitations on our ability to represent our everyday knowledge about practical domains in ways that will support adequate assessment of plausibility. For many applications such a parser will therefore benefit very little from the pruning step. Moreover, as we saw at the end of chapter 8, this particular algorithm, being bottom-up, inherits the disease of building useless constituents.

Many of the worst effects of the disease can be eliminated by being more careful about which categories for a_j are input to the algorithm in step 1. For example, almost all nouns like *thieves* in English can be either N or NP. However, when preceded by an article, as in *the thieves*, the relevant category is, with overwhelmingly high probability, N rather than NP. This fact provides the basis for a number of low-level, purely stochastic, syntax-independent "part-of-speech-tagging" (POS) methods for disambiguating lexical form-class (Jelinek 1976; Merialdo 1994), based on algorithms that can be automatically trained on text or speech corpora. POS tagging can be used to limit the candidate categories input to the CKY algorithm to the "*n*-best" or most likely categories, as de Marcken (1990) points out, eliminating much of the disadvantage of bottom-up techniques.

Indeed, it is likely that CCG and other lexicalized grammars, such as TAG, will benefit more from such stochastic filtering. POS tagging is commonly based on around 60 form classes (Francis and Kučera 1964, 23–25), some of which, such as VBZ (verb, 3rd person singular present), are not as informative as they might be. By contrast, CCG and TAG have several distinct categories or elementary trees corresponding to VBZ, distinguishing intransitive and a number of distinct varieties of transitive and ditransitive verbs. This suggests that better POS-tagging algorithms could be developed by using CCG or TAG categories in place of the standard POS categories, a proposal that has been investigated by B. Srinivas and Joshi (1994). Experiments of this kind are reported by B. Srinivas (1997) and Doran and B. Srinivas (to appear), with promising results. One interesting question for this research is whether stochastic methods will be effective in disambiguating type-raising.

Recent work by Collins (1996, 1997, 1998) points to the advantages of an even greater integration of probabilistic information with syntax in parsers for lexicalized context-free grammars, including CCG. Collins (1997, 1998) presents a technique for supervised learning of probabilistic Dependency Grammars, in which the production rules are induced from a tree-bank and probabilities are found for a given rule applying with a given lexical "head" and arguments with other given heads. Some of the distinctive features of the procedure for assigning these probabilities are a method based on maximum likelihood estimation and a "backing off" method for use in the face of sparse data. The probabilities can be used to guide search in any one of a number of standard parsing algorithms, including shift-reduce, beam search, and CKY. This parser was at the time of writing the most accurate wide coverage parser by the standard measures, with precision/recall figures better than 88% on unseen Wall Street Journal text.

Because of its simultaneous clean theoretical separation between competence grammar, parsing algorithm, and probability, and because of its close coupling of these elements in processing, Collins's method is extremely general. More or less any class of lexicalized grammar can be made to yield dependency structures, and so probabilistic parsers can in principle be induced for them too, provided that the tree-bank that is used records the relevant dependencies. Combinatory Categorial Grammar is a particularly interesting case to consider, not only because of the historically close relation between Categorial Grammar and Dependency Grammar, and because the Logical Forms that CCGs build capture the dependencies in question, but also because of the simple way in which they project lexical dependencies, and hence the associated head-dependency probabilities, onto unbounded and fragmentary constructions including coordination. (The advantages of this property for grammar induction have already been mentioned in connection with human language acquisition.)

It is not clear what psychological reality such stochastic methods can lay claim to. It does not seem likely that the semantic methods I have advocated for resolving attachment ambiguities will solve the problem of lexical ambiguity. On the other hand, it does seem possible that human processors could derive plausibility measures directly from properties of the syntax and associative properties of the memory for concepts underlying word meanings, rather than by collecting higher-order statistics over large volumes of data. The rule-based POS taggers of Brill (1992) Voutilainen (1995), Kempe and Karttunen (1996), and Cussens (1997), and related sense-disambiguation work by Resnik (1992) using WordNet (Miller and Fellbaum 1991) are suggestive in this respect. However, to the extent that very transient changes to the sets of referents that are available in the discourse model can affect processing load and garden path effects, as in the experiments discussed earlier, processes of active interpretation, including limited amounts of inference, seem to be the only plausible basis for resolution of structural or attachment ambiguities by the psychological processor.

Chapter 10
The Syntactic Interface

Lofty designs must close in like effects.
Robert Browning, "A Grammarian's Funeral"

This book began by stating some uncontroversial assumptions in the form of the rule-to-rule condition and the competence hypothesis, deducing the even more widely accepted Constituent Condition on rules of competence grammar. The Introduction also endorsed the methodological priority of investigating competence syntax over performance mechanisms. Having noted the difficulties presented by coordination and intonation in relation to the Constituent Condition on Rules, part I of the book went on to advance an alternative combinatory view of competence grammar under which these apparently paradoxical constructions were seen to conform to that condition after all. After putting the theory through its syntactic paces in part II, the progression has been brought full circle in part III by deriving some consequences for the theory of performance under a "strict" version of the competence hypothesis.

10.1 Competence

The competence theory that was developed along the way is conveniently viewed in terms of a third and final version of the by-now familiar Y-diagram in figure 10.1, which combines figures 4.1 and 5.3, again including mnemonic exemplars of the constructs characteristic of each module of the theory. According to this theory, lexical items and derived constituents (including sentences) pair a phonological representation with a syntactic category (identifying type and directionality only) and an interpretation. Chapter 5 showed that the interpretations of the principal constituents of the sentence correspond to the information structural components called theme and rheme. These in turn combine by function application or "β-normalization" to yield fairly standard quantified predicate-argument structures or Logical Forms. Predicate-argument structures preserve fairly traditional relations of dominance and command. In par-

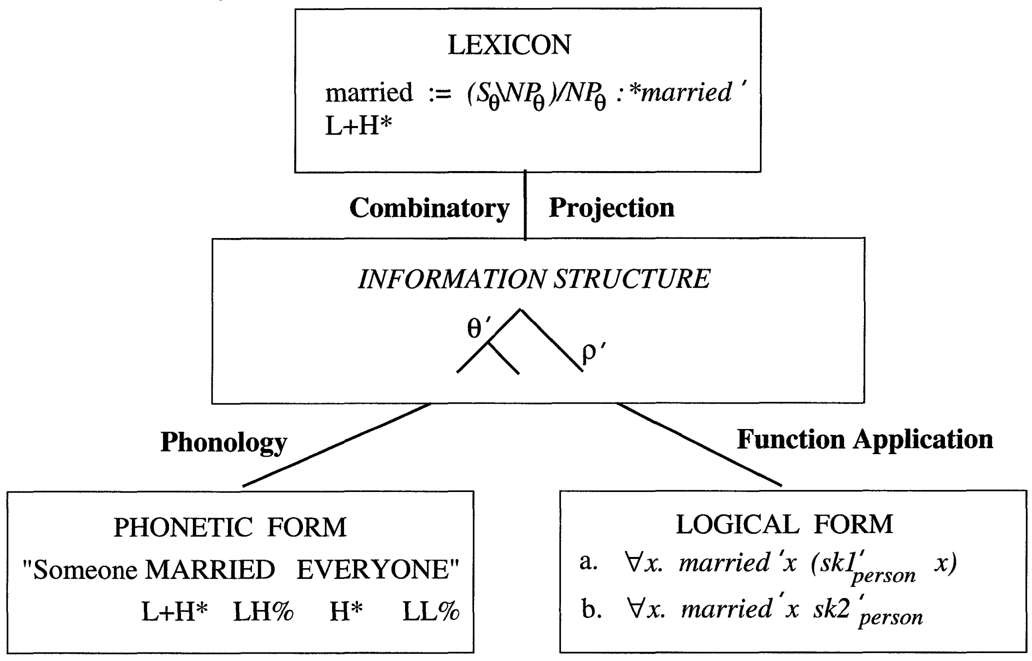
LEXICON
married := $(S_\theta\backslash NP_\theta)/NP_\theta$: *married′
L+H*
Combinatory
Projection
INFORMATION STRUCTURE
θ′
ρ′
Phonology
Function Application
PHONETIC FORM
"Someone MARRIED EVERYONE"
L+H* LH% H* LL%
LOGICAL FORM
a. ∀x. married′x (sk1′person x)
b. ∀x. married′x sk2′person

Figure 10.1
Architecture of Combinatory Categorial Grammar III

ticular, they embody the obliqueness hierarchy on grammatical relations over arguments. The order of combination that is defined by the syntactic category need not conform to the obliqueness hierarchy, and in VSO and SVO languages cannot conform to it.

Traditional notions of command and dominance have nothing to do with derivation in this sense. Instead, derivations capture directly the notion of constituency relevant to relativization, coordination, and phrasal intonation, without the invocation of empty syntactic categories or syntactic operations of "movement," "deletion," "copying," or "restructuring," distinct from those implicit in the automatic construction of the appropriate Logical Form. This notion of structure should be identified with Information Structure, rather than traditional Surface Structure. Although it is convenient to represent Information Structures as trees, they do not constitute a level of representation in the theory. In contrast to Logical Form and the associated predicate-argument structural domain of the binding theory, no rule or relation is predicated over such structures.

The responsibility of the combinatory rules is to "project" both components of the lexical categories, synchronously and in lockstep, onto the corresponding constituent components of the derivation.[1] The types of the constituents that they yield are considerably more diverse than those implicated in traditional Surface Structures or GB S-Structures. They provide the input to rules of coordination, parentheticalization, and extraction, all of which are thereby brought under the Constituent Condition on Rules. They also provide the input to purely local phonological processes, such as vowel harmony or *liaison* and the Rhythm Rule (Selkirk 1984), which directly map information-structural constituents onto Phonetic Form proper.

There is no conflict between such a view of surface constituency and more traditional theories of grammar. In categorial terms, such theories can be seen as predominantly concerned with predicate-argument structure and hence with elements of semantic interpretation or Logical Form, rather than syntax proper. To the extent that such theories provide a systematic account of the relation between interpretations in this sense and syntactic categories, they provide what amounts to a theory of the categorial lexicon—a component of the present theory that continues to be lacking in this and preceding discussions of CCG.

By contrast, the normalized Logical Form or quantified predicate-argument structure, which is the exclusive domain of the binding theory, provides the input to such systems as reference and the binding of pronouns. It is presumably at this level that the effects associated with "weak crossover" and "subjacency" make themselves felt. Although we may find it convenient to think about these processes in terms of a further structural level of Logical Form, such a representation is not in principle necessary, for the reasons discussed by Montague (1970), and in fact this level is eschewed in other versions of Categorial Grammar. In chapter 4, I discussed how such systems should capture ambiguities of quantifier scope without movement at LF or the equivalent, drawing on work by VanLehn (1978), Webber (1978, 1983), Reinhart (1991, 1997), Park (1995, 1996), Winter (1997), and Schlenker (to appear).

The question of how well the theory generalizes to more parametrically diverse languages than English and its Germanic relatives, and in particular to languages with freer word order and those that use morphological markers of Information Structure rather than intonational ones, goes beyond the scope of the present book. However, Kang (1988), Segond (1990), Foster (1990) Nishida (1996), Hoffman (1995a,b, 1996, 1998), Bozsahin (1998), Komagata (1997b, 1999), Baldridge (1998, 1999), and Trechsel (to appear) offer CCG analyses for the grammar and Information Structure of Korean, French, Spanish, Old Spanish, Turkish, Japanese, Tagalog and Tzotzil.

10.2 Acquisition

The explanatory adequacy of the theory will also depend on its compatibility with a reasonable account of language acquisition. This question also lies beyond the scope of the present book, and the following remarks are restricted to the briefest of preliminary sketches. (See Briscoe 1997, forthcoming, Osborne and Briscoe (1997), and Watkinson and Manandhar 1999 for specific proposals for acquiring categorial grammars in the face of noise and situational ambiguity, and see Kanazawa 1998 on the computational complexity of the problem.)

The considerations discussed in chapters 1 and 8 suggest that language acquisition mainly reduces to the problem of learning the categorial lexicon and the language-specific instances of the combinatory rule types that are involved. Lexical learning must in the earliest stages depend upon the child's having access to mental representations of the concepts underlying words, in a form more or less equivalent to the lexical Logical Forms assumed here, perhaps along lines suggested in Pinker 1979, Fisher et al. 1994, Steedman 1994, and Siskind 1995, 1996. Under the assumptions inherent in the Principle of Categorial Type Transparency, the semantic type of such concepts defines the syntactic type in every respect except directionality. The Principle of Head Categorial Uniqueness ensures that in most cases the child need have access only to combinatory rules of functional application in order to deduce the latter property, and hence the lexical category or categories of each word. The tendency of languages toward consistency in head-complement orders suggests that this search is constrained accordingly.

As far as the combinatory rules go, it seems likely that the repertoire of semantic combinators is fixed as composition, substitution and (possibly as a lexical rule) type-raising over the categories that are actually encountered in the grammar acquired so far. Once some lexical categories are known, the child is therefore immediately in a position to master constructions like relatives by inducing the particular instances of combinatory rules that the grammar includes—principally those of the composition family— and the categories to which they apply, working on the basis of the lexical categories learned in the manner sketched above and contextually available compound concepts, perhaps along lines sketched in Steedman 1996a. (The fact that lexical learning generalizes in this way is an important reason why natural languages should adhere as closely as possible to the Principle of Head Categorial Uniqueness.) The simplest way to do this would be to include only the most specific instance of a combinatory rule that supports a combination that yields the concept in question. However, such an assumption raises the same questions of inductive generalization and stability in the face of noise and ambiguity that arise in other frameworks.

The most serious problem that this account faces arises from the inclusion of exceptions to the Principle of Head Categorial Uniqueness, such as the subject extraction category stipulated in chapter 4 at example (20) for verbs like *think*. When children who are acquiring English encounter subject extraction, they have three options. They might wrongly assume that the grammar of English includes the rule of crossed composition that was rejected in chapter 4—in which case they will begin to overgenerate wildly. Or they might rightly assume that this counts as a different construction headed by *think*, specified by a separate lexical entry, but wrongly assume that this lexical entry conforms to the Principle of Head Categorial Uniqueness—in which case they will begin to overgenerate sentences like **I think fell the horse*. Or they might correctly further assume that this lexical entry is independent, violating the Uniqueness principle.

One way to ensure that the child makes the right choice, despite the penalty associated with violating this principle, is to stipulate that categories that are not induced via an application-only derivation of the kind described earlier, but (like this one) are first encountered under extraction, are assigned the most conservative category that will allow the sentence—that is, one confined to antecedent government, an assumption analogous to Baker's (1979) proposal for conservative acquisition of dative shift.[2] (Similar remarks may apply to acquisition of the antecedent government–restricted combinatory rules for phenomena like Heavy NP shift, discussed in chapters 4 and 6.) Such a procedure seems to be one that could only be safely applied in the last stages of fine-tuning a stable grammar based on a sizable corpus. Stromswold's (1995) results showing that complement subject extraction is one of the last constructions to be acquired in English (see section 4.2.1) are consistent with this procedure.[3]

10.3 Performance

The architecture schematized in figure 10.1 embodies the strongest possible relation between Surface Structure or derivation, Intonation Structure and Information Structure. The evidence for it is entirely based on linguistic argumentation, and it must in the first place be judged on those grounds. Nevertheless, this property of the theory has significant implications for processing under the strict competence hypothesis. The fact that syntactic constituency subsumes intonational constituency in the sense discussed in chapter 5 implies that modular processors that use both sources of information at once should be easier to devise. Such an architecture may reasonably be expected to simplify the problem of resolving local structural ambiguity in both domains.

However, we have noted that a considerable amount of nondeterminism remains in the grammar, for both spoken and written language. Although this nondeterminism can be kept within polynomial complexity bounds using techniques discussed in chapter 9, the associativity implicit in functional composition means that the average-case complexity potentially remains serious. The properties of the grammar are consistent with the suggestion that the basis for the oracle that renders the process as a whole deterministic is the incremental availability of semantic interpretations (possibly compiled in the form of related head-dependency probabilities of the kind discussed by Collins (1998).)

The generalized notion of constituency that is engendered by the combinatory rules ensures that many leftmost substrings are potentially constituents with interpretations, subject of course to the limitations of the grammar and any further information that may be available from intonation. Such a theory of grammar may therefore have the added advantage of parsimony, in being compatible with such a processor without compromising the strict competence hypothesis.

Indeed, we can stand this argument on its head. If we believe that the parser has to know about interpretations corresponding to strings like *The flowers sent for*..., and we identify such interpretations with the notion of abstraction, then advocates of more traditional notions of constituency must ask themselves why their grammar does *not* accord such useful and accessible semantic concepts the status of grammatical constituents.

The claim is strengthened by the observation that the residual nondeterminism in the grammar of intonation, arising in part from the widespread presence of unmarked themes, as discussed in connection with example (55) in chapter 5, is confined precisely to those occasions on which the topic or theme is believed by the speaker to be entirely known to all parties, and to be recoverable by comparing the interpretation of a (usually leftmost) substring with the contextual open proposition or theme. It would be surprising if the mechanism for disambiguating written language were very different from its ancestor in the processor for spoken language.

It is of course unlikely that we will ever know enough about the biological constraints to evaluate the assumptions on which the "strict" version of the competence hypothesis is based with any certainty. In the absence of such certainty, we must beware of falling into the error of evolutionary Panglossism. However, it is appropriate to speculate a little further upon the implications of the Strict Competence Hypothesis for the theory as a whole, for the following reason.

Competence grammar and performance mechanism originally evolved as components of a single biological system. The methodological priority of competence that has been continually endorsed in the present work is no more than a research strategy. Any claim about competence grammar is ultimately a claim about the entire computational package. As soon as our linguistic theories have attained the level of descriptive adequacy, they will have to be judged not merely on their purity and parsimony as theories of competence, but on their explanatory value as part of a psychologically and biologically credible performance system. Chapter 9 noted that all theories will require *something* more, in the form of a language-independent mechanism for resolving local ambiguity, or grammatical nondeterminism, together with a language-independent algorithm and automaton. But if a theory of competence requires much more than that, or if that mechanism in turn implicates a notion of structure that is not covered by the competence grammar, then those assumptions will weigh against it. If there is another descriptively adequate theory that requires fewer such assumptions, perhaps even no further assumptions beyond the mechanism for resolving nondeterminism and the minimal bottom-up algorithm, by virtue of having a different notion of surface syntax, then the scales may tilt in its favor.

None of the current theories of grammar, including the present one, have yet attained the full descriptive adequacy that would allow us to weigh them in the balance in this way. But if it is true that the principal responsibility for local ambiguity resolution lies with word-by-word incremental interpretation (or with correspondingly fine-grain probabilistic evaluation), then any theory that does not make assumptions similar to those of CCG concerning constituency in the competence grammar will, as we saw in chapter 9, have to make some strikingly similar structures available to the processor, complete with interpretations. Such additional assumptions could not by definition be inconsistent with the pure competence theory itself. However, they compromise the Strict Competence Hypothesis. To the extent that a combinatory grammar can achieve the same result without any additional assumptions, and to the extent that it is descriptively correct to include identical structures and interpretations in the competence grammar of coordination and intonation, the combinatory theory may then be preferred as an explanatory account.

BLANK PAGE

Notes

Chapter 1

1. The HOLD-register analysis of *wh*-movement was in part anticipated in earlier work by Thorne, Bratley and Dewar (1968), who called their register *.

2. Wood (1993) provides a useful review of theories by Lambek (1958), Ades and Steedman (1982), Bach (1979), Dowty (1979), Steedman (1987), Oehrle (1988), Hepple (1990), Jacobson (1990, 1992b), Szabolcsi (1989, 1992), and Wood (1988), although my colleagues should not be assumed to endorse all the assumptions of the version that is outlined here. The present proposal is more distantly related to a number of other generalizations of the early categorial systems of Ajdukiewicz, Bach, Bar-Hillel, Dowty, Lambek, Geach, Lewis, Montague, van Benthem, Cresswell, and von Stechow, to many of which the conclusions of this book also apply. In particular, Oehrle (1987), Moortgat (1988a), and Morrill (1994) explicitly relate Lambek-style categorial grammars to prosody.

3. Marr expressed some doubt about whether natural language is in fact a modular system, apparently because he was aware of the way knowledge and inference interact with language understanding. I will argue against this conclusion in chapter 9.

Chapter 2

1. This claim should not be taken as denying that such learning can be usefully thought of in terms of supervised machine learning techniques, or as excluding the possibility that the substrate of such conceptual representations may be associative or probabilistic.

2. The "Standard Theory" presented in Chomsky 1965 did not explicitly recognize any level of Logical Form distinct from Deep Structure. However, had it done so, it would have had to derive it from Deep Structure. The fact that later "Extended," "Revised Extended," and "Principles and Parameters" or "Government-Binding (GB)" versions of Chomsky's theory derived Logical Form from a level called "S-Structure" should not be allowed to confuse the point. S-Structure is not the same as Surface Structure, as will become clear when this level is discussed in more detail below. The rather different view of Logical Form sketched in Chomsky 1971 is discussed in chapter 5.

3. There are a number of well-known exceptions to this generalization, which I will